

USER GUIDE

# Essential Studio

## for EJ2 Vue

---

Version - v24.1.41 | Release Date - December 18, 2023

Syncfusion Vue UI Components (Essential JS 2) .....	99
Components list .....	99
table .....	99
How to best read this user guide .....	101
Getting help .....	101
See also .....	101
System Requirements for Vue UI Components .....	101
Browser support .....	102
Vue supported versions .....	102
Browser Support .....	102
Required polyfills .....	102
Using CDN .....	102
Node.js .....	103
Getting started .....	103
Vue 3 .....	103
Getting Started with Vue UI Components with JavaScript and Composition API .....	103
Getting Started with Vue UI Components with TypeScript and Composition API .....	108
Getting Started with Vue UI Components with JavaScript and Options API .....	114
Getting Started with Vue UI Components with TypeScript and Options API .....	119
Getting Started with Syncfusion Vue UI Components in Vue 3 .....	126
Vue 2 .....	133
Getting Started with Syncfusion Vue UI Components and Vue CLI .....	133
Getting Started with Syncfusion Vue UI Components and Vue CLI .....	137
Getting Started with Vue UI Components with the Nuxt Framework .....	141
Prerequisites .....	142
Set up the Nuxt project .....	142
Add Syncfusion Vue packages .....	142
Import Syncfusion CSS styles .....	142
Add the Syncfusion Vue component .....	143
Run the project .....	144
Getting Started with Vue UI Components with Vite and PNPM .....	145
Prerequisites .....	145
Set up the Vite project using PNPM .....	145
Add Syncfusion Vue packages .....	146
Import Syncfusion CSS styles .....	146

Add Syncfusion Vue component .....	147
Run the project .....	149
See also .....	149
Getting started with testing Vue UI components in the Vitest project .....	149
Prerequisites .....	149
Set up the Vitest project .....	149
Add the Syncfusion packages.....	150
Add the Syncfusion Vue component.....	151
Run the project .....	152
Getting Started with Syncfusion Vue UI Components using direct scripts in a quickstart application	153
Prerequisites .....	153
Set up the Vue project .....	153
Import Syncfusion CSS styles .....	154
Import Syncfusion Vue scripts .....	154
Add Syncfusion Vue component.....	154
Run the project .....	157
Installation and Upgrade.....	157
Installation .....	157
Install by using npm CLI.....	157
Install by using package.json.....	158
Download JavaScript – EJ2 Installer .....	158
Download the trial version.....	158
Installation using Web Installer .....	161
Overview .....	161
Installation .....	161
Uninstallation.....	168
Installation using Offline Installer .....	174
Installing with UI .....	175
Installing in silent mode .....	182
Installing Syncfusion JavaScript – EJ2 Mac Installer.....	183
Steps to resolve the warning message in Catalina OS or later .....	183
Step-by-Step Installation.....	184
License key registration in samples .....	189
Linux Installer .....	189
Download Syncfusion JavaScript Linux Installer .....	189

Installing Syncfusion JavaScript Linux installer .....	193
Common Installation Errors .....	194
Unlocking the license installer using the trial key.....	194
License has expired .....	194
Unable to find a valid license or trial .....	195
Unable to install because of another installation.....	196
Unable to install due to controlled folder access .....	197
Upgrade.....	199
Syncfusion Vue supported versions .....	199
Vue version compatibility .....	199
Syncfusion version information .....	199
See also .....	199
Upgrading Syncfusion JavaScript (Essential JS2).....	199
Upgrading to the latest version .....	199
Upgrade from trial version to license version.....	200
Licensing.....	200
Syncfusion Licensing Overview .....	200
Difference between unlock key and license key.....	200
Registering Syncfusion license keys in Build server .....	201
See also .....	201
Generate Syncfusion Vue License key .....	201
Claim license key.....	201
See also .....	203
Register Syncfusion License key in Vue application.....	203
Register Syncfusion license key in the project.....	204
Register Syncfusion license key in the Nuxt project .....	204
Register Syncfusion license key using the npx command.....	204
Activate the license.....	208
See also .....	209
Syncfusion Licensing Errors.....	209
Licensing errors.....	209
Licensing errors from version 16.2.0* to 20.3.0* .....	212
Licensing FAQ.....	216
Is an internet connection required for license validation.....	216
Upgrade from the trial version after purchasing a license .....	216



Where can I get a license key.....	216
Will the registered license key expire .....	217
When to generate new license key while upgrading.....	217
License registration for multiple developers on your project .....	217
Can I use the same key for all the web apps under the project .....	217
Does the license registration access any resources or data .....	217
License & Downloads shows the "Essential Studio Enterprise Edition Binary with Test Studio" and the "Project License". Which license to use.....	218
If I registered the license key in both the application and the license text file .....	218
Potential causes of licensing errors in applications. ....	218
Appearance .....	219
Theme in Vue Appearance component .....	219
Referring individual control theme .....	220
Common variables .....	221
Size modes in Vue Appearance component .....	233
Size mode for application .....	233
Size mode for a component .....	234
Change size mode for application at runtime.....	234
Change size mode for a component at runtime .....	235
See also .....	236
Icons in Vue Appearance component.....	236
Referring icons in the Vue application.....	236
Steps to use icons library .....	237
Available icons .....	241
Theme studio in Vue Appearance component .....	242
Customizing theme color from theme studio.....	242
Import previously changed settings into the theme studio .....	250
Material 3 Theme.....	253
Syncfusion Material 3 Theme .....	253
What are CSS Variables? .....	254
Dark mode support .....	255
ThemeStudio application .....	257
Common.....	258
Accessibility in Syncfusion Vue Components.....	258
Accessibility overview .....	258

Accessibility standards .....	258
Accessibility compliance .....	258
Ensuring accessibility .....	259
Accessibility support for specific components.....	259
table .....	260
Right-To-Left support in Syncfusion Vue Components.....	261
Enable RTL for all components .....	261
Enable RTL for an individual component .....	263
State Persistence in Syncfusion Vue components .....	264
State Persistence supported components and properties .....	265
Event Binding .....	267
Custom events .....	267
calendar { .....	268
Native events .....	268
Model Binding.....	269
Integrating Vue model binding in Syncfusion Vue UI components .....	269
Templates in Syncfusion Vue Components.....	270
Types of templates.....	270
Slot template.....	270
Inline template.....	273
External template.....	274
External modules in templates .....	276
Provide/Inject in templates .....	278
Animation in Vue .....	279
Animation effects.....	280
Animation duration.....	280
Animation delay .....	281
Enable or disable animation globally .....	282
Getting started with Localization.....	282
Loading translations.....	283
Changing current locale .....	284
Internationalization.....	284
Loading culture data .....	284
Changing Global Culture and Currency Code.....	285
Manipulating numbers.....	285

Manipulating DateTime .....	289
Drag and Drop in Vue.....	294
Draggable .....	294
Droppable .....	297
See also .....	298
Deployment .....	298
CDN .....	298
Packages.....	299
How To .....	300
Updating Syncfusion npm packages .....	300
How to resolve Content Security Policy (CSP) errors.....	300
Troubleshoot.....	301
Content Security Policy .....	301
Visual studio code integration .....	302
Visual Studio Code Integration .....	302
Overview .....	302
Visual Studio Code Extensions .....	302
Download and Installation .....	302
Visual Studio Code Extensions .....	304
Intellisense of Syncfusion Vue Components.....	308
Accordion .....	309
Getting Started with the Vue Accordion Component in Vue 2.....	309
Prerequisites .....	309
Dependencies.....	309
Setting up the Vue 2 project .....	310
Adding syncfusion packages .....	310
Add Syncfusion Vue component.....	311
Running the Application.....	312
Initialize the Accordion using HTML elements.....	313
Getting Started with Vue Accordion Component in Vue 3 .....	316
Prerequisites .....	316
Setup the Vite project .....	316
Adding Syncfusion Vue packages.....	317
Import Syncfusion CSS styles .....	318
Adding Syncfusion Vue component .....	318

Run the project .....	321
See also .....	321
Expand mode in Vue Accordion component .....	321
Single .....	321
Multiple .....	322
See Also .....	323
Accessibility in Vue Accordion component .....	323
ARIA attributes .....	324
Keyboard interaction .....	325
Ensuring accessibility .....	325
See also .....	325
Style in Vue Accordion component .....	325
Customizing Accordion .....	325
Customizing the list items .....	326
Customizing Accordion's header .....	326
Customizing Accordion's expand and collapse icons .....	326
Customizing the hover state of Accordion control .....	326
Customizing selected item of Accordion control .....	326
How To .....	327
Set the nested accordion in Vue Accordion component .....	327
Load content through post in Vue Accordion component .....	329
Set custom animation in Vue Accordion component .....	330
To keep single pane open always in Vue Accordion component .....	332
Create wizard using accordion in Vue Accordion component .....	333
Load accordion with data source in Vue Accordion component .....	339
Customize expand collapse actions in Vue Accordion component .....	340
Render other components in accordion using template in Vue Accordion component .....	342
Accumulation Chart .....	343
Getting Started with the Vue Accumulation chart Component in Vue 2 .....	343
Prerequisites .....	343
Dependencies .....	344
Setting up the Vue 2 project .....	344
Add Syncfusion Vue packages .....	345
Add Syncfusion Vue component .....	345
Adding Chart Component .....	346

Run the project .....	347
Getting Started with the Vue Accumulation Chart Component in Vue 3 .....	347
Prerequisites .....	347
Set up the Vite project .....	348
Add Syncfusion Vue packages.....	349
Add Syncfusion Vue component.....	349
Run the project .....	352
See also .....	353
Pie dough nut in Vue Accumulation chart component.....	353
Pie Chart.....	353
Radius Customization.....	354
Pie Center.....	355
Various Radius Pie Chart .....	356
Doughnut Chart.....	357
Start and End angles .....	358
Color and Text Mapping.....	359
Multi-level pie chart.....	360
Pyramid in Vue Accumulation chart component.....	363
Mode.....	364
Size .....	365
Gap Between the Segments.....	366
Explode.....	367
Customization .....	367
Funnel in Vue Accumulation chart component .....	368
Size .....	369
Neck Size .....	370
Gap between the segments .....	371
Explode.....	372
Smart Data Label.....	373
Customization .....	374
See Also .....	375
Data label in Vue Accumulation chart component.....	375
Positioning.....	376
DataLabel rotation .....	377
Smart labels.....	378

Datalabel template .....	379
Connector Line .....	380
Text Mapping .....	382
Format.....	382
Customization .....	384
Text wrap .....	385
Show percentages in data labels of pie chart .....	386
Grouping in Vue Accumulation chart component .....	388
Broken slice .....	389
GroupMode.....	390
Customization .....	391
Empty points in Vue Accumulation chart component.....	392
Customization .....	393
Annotation in Vue Accumulation chart component.....	394
Region .....	395
Co-ordinate Units.....	396
Alignment.....	397
Tooltip in Vue Accumulation chart component.....	398
Header.....	399
Format.....	400
Tooltip template .....	400
Fixed tooltip .....	401
Customization .....	402
To customize individual tooltip.....	403
Tooltip mapping name.....	404
Legend in Vue Accumulation chart component .....	405
Position and Alignment.....	406
Legend Reverse .....	407
Legend Shape .....	409
Legend Size.....	409
Legend Item Size .....	410
Paging for Legend.....	411
Legend Text Wrap .....	412
Enable Animation.....	413
Legend Title .....	414

Arrow Page Navigation .....	415
Legend Item Padding .....	416
Center Label in Vue Accumulation chart component.....	417
Center label.....	417
Hover text .....	418
Customization .....	418
Title and sub title in Vue Accumulation chart component .....	420
Title .....	420
Title customization.....	421
SubTitle .....	422
SubTitle Customization .....	423
Chart print in Vue Accumulation chart component.....	424
Print.....	424
Export.....	425
Appearance in Vue Accumulation chart component.....	426
Custom Color Palette .....	426
Animation.....	427
Accessibility in Vue Accumulation chart component .....	429
WAI-ARIA attributes.....	430
Keyboard interaction .....	430
Ensuring accessibility .....	431
See also .....	431
AppBar.....	431
Getting Started with the Vue AppBar Component in Vue 2 .....	431
Prerequisites .....	431
Dependencies.....	431
Setting up the Vue 2 project .....	431
Add Syncfusion Vue packages.....	432
Import Syncfusion CSS styles .....	433
Add Syncfusion Vue component .....	433
Run the project .....	435
Getting Started with Vue AppBar Component in Vue 3 .....	435
Prerequisites .....	435
Setup the Vite project .....	435
Add Syncfusion Vue packages.....	437

Import Syncfusion CSS styles .....	437
Adding Syncfusion Vue AppBar component in the application .....	437
Run the project .....	439
See also .....	440
Size and color in Vue Appbar component.....	440
Size .....	440
Color .....	443
Position in Vue Appbar component.....	446
Top AppBar .....	446
Bottom AppBar .....	447
Sticky AppBar .....	448
Design in Vue Appbar component .....	450
Spacer.....	450
Separator.....	450
Media Query .....	451
Designing AppBar with Menu .....	452
Designing AppBar with Buttons .....	454
Designing AppBar with SideBar.....	455
Style and appearance in Vue Appbar component .....	458
CssClass .....	458
HtmlAttributes .....	459
AutoComplete .....	460
Getting Started with the Vue Auto compleye Component in Vue 2 .....	460
Prerequisites .....	460
Dependencies.....	460
Setting up the Vue 2 project .....	460
Add Syncfusion Vue packages.....	461
Import Syncfusion CSS styles .....	461
Add Syncfusion Vue component.....	462
Binding data source .....	462
Run the project .....	464
Custom values.....	465
Configure the suggestion list .....	466
See Also .....	468
Getting Started with the Vue AutoComplete Component in Vue 3 .....	468



Prerequisites .....	468
Set up the Vite project .....	468
Add Syncfusion Vue packages.....	470
Import Syncfusion CSS styles .....	470
Add Syncfusion Vue component.....	470
Run the project .....	472
Custom values.....	472
Configure the suggestion list .....	474
See Also .....	475
Data binding in Vue Auto complete component .....	475
Bind to local data .....	476
Array of complex object.....	478
Bind to remote data.....	479
See Also .....	480
Templates in Vue Auto complete component.....	480
Item template .....	480
Group template.....	481
Header template .....	483
Footer template .....	484
No records template .....	486
Action failure template .....	487
See Also .....	489
Virtualization in AutoComplete Component .....	489
Binding local data.....	489
Binding remote data .....	490
Grouping .....	492
Grouping in Vue Auto complete component.....	493
See Also .....	494
Filtering in Vue Auto complete component.....	494
Change the filter type .....	494
Filter item count.....	496
Limit the minimum filter character.....	497
Case sensitive filtering .....	498
Diacritics Filtering.....	499
See Also .....	500

Localization in Vue Auto complete component.....	500
Loading translations.....	500
See Also .....	502
Style in Vue Auto complete component.....	502
Customizing the appearance of wrapper element .....	502
Customizing the dropdown icon's color .....	502
Customizing the focus color.....	502
Customizing the outline theme's focus color .....	503
Customizing the disabled component's text color .....	503
Customizing the float label element's focusing color .....	503
Customizing the color of the placeholder text .....	503
Customizing the text selection color.....	504
Customizing the background color of focus, hover, and active item's .....	504
Customizing the appearance of pop-up element .....	504
Adding mandatory asterisk to placeholder and float label.....	504
Accessibility in Vue Auto complete component .....	505
WAI-ARIA attributes.....	506
ARIA attributes.....	506
Keyboard interaction .....	507
Ensuring accessibility .....	509
See also .....	509
Two way binding in Vue Auto complete component .....	509
How To .....	510
Autofill in Vue Auto complete component .....	510
Icon support in Vue Auto complete component .....	511
Custom search in Vue Auto complete component.....	513
Avatar .....	514
Getting Started with the Vue Avatar Component in Vue 2 .....	514
Prerequisites .....	514
Setting up the Vue 2 project .....	514
Add Syncfusion Vue packages.....	515
Import Syncfusion CSS styles .....	515
Using Avatar in Vue Application.....	516
Run the project .....	516
See Also .....	517

Getting Started with the Vue Avatar Component in Vue 3 .....	517
Prerequisites .....	517
Set up the Vite project .....	517
Add Syncfusion Vue packages.....	518
Import Syncfusion CSS styles .....	519
Add Syncfusion Vue component.....	519
Run the project .....	520
See also .....	520
Types in Vue Avatar component.....	521
Avatar size .....	521
Avatar types .....	522
How To .....	523
Avatar customization in Vue Avatar component.....	523
Integrate avatar into listview in Vue Avatar component.....	529
Integrate avatar into badge in Vue Avatar component.....	532
Badge .....	536
Getting Started with the Vue Badge Component in Vue 2 .....	536
Prerequisites .....	536
Setting up the Vue 2 project .....	536
Add Syncfusion Vue packages.....	537
Import Syncfusion CSS styles .....	537
Run the project .....	539
See Also .....	539
Getting Started with the Vue Badge Component in Vue 3 .....	539
Prerequisites .....	539
Set up the Vite project .....	539
Add Syncfusion Vue packages.....	541
Import Syncfusion CSS styles .....	541
Add Syncfusion Vue component.....	541
Run the project .....	542
See also .....	542
Types in Vue Badge component .....	542
Badge styles .....	543
Badge types.....	546
How To .....	561

Badge customization in Vue Badge component .....	561
Integrate badge into listview in Vue Badge component.....	567
Dynamic badge content in Vue Badge component .....	571
Barcode .....	575
Getting Started with the Vue Barcode Component in Vue 2.....	575
Prerequisites .....	575
Dependencies.....	575
Setting up the Vue 2 project .....	576
Adding Syncfusion Vue packages.....	576
Add Syncfusion Vue component.....	577
Adding QR Generator control .....	579
Adding Datamatrix Generator control.....	581
Getting Started with Syncfusion Barcode Component in Vue 3 .....	582
Prerequisites .....	582
Creating Vue application using Vue CLI .....	582
Adding Syncfusion Barcode package in the application .....	583
Adding CSS reference for Syncfusion Vue Barcode component.....	583
Adding Syncfusion Vue Barcode component in the application.....	583
Running the application .....	586
Adding Syncfusion Vue QR Generator control in the application.....	587
Adding Syncfusion Vue Datamatrix Generator control in the application .....	588
BarcodeGenerator in Vue Barcode component .....	590
Code39 .....	590
Code39 Extended .....	591
Code 11 .....	592
Codabar .....	593
Code 32 .....	594
Code 93 .....	595
Code 93 Extended .....	596
Code 128 .....	596
Customizing the Barcode color .....	597
Customizing the Barcode dimension .....	598
Customizing the text .....	599
Qrcodegenerator in Vue Barcode component .....	600
QR Code .....	600

Customizing the Barcode color .....	601
Customizing the Barcode dimension .....	601
Customizing the text .....	602
Datamatrixgenerator in Vue Barcode component .....	603
Data Matrix .....	603
Customizing the Barcode color .....	604
Customizing the Barcode dimension .....	605
Customizing the text .....	606
Export in Vue Barcode component.....	607
Export.....	607
Breadcrumb .....	609
Getting Started with the Vue Breadcrumb Component in Vue 2 .....	609
Prerequisites .....	609
Dependencies.....	609
Setting up the Vue 2 project .....	609
Add Syncfusion Vue packages.....	610
Add Syncfusion Vue component.....	610
Run the project .....	611
Add Items to the Breadcrumb Component .....	612
Enable or Disable Navigation .....	612
Getting Started with the Vue Breadcrumb Component in Vue 3 .....	614
Prerequisites .....	614
Set up the Vite project .....	614
Add Syncfusion Vue packages.....	616
Import Syncfusion CSS styles .....	616
Add Syncfusion Vue component.....	616
Run the project .....	619
See also .....	619
Data binding in Vue Breadcrumb component .....	620
Items as tag directive .....	620
Items based on current URL.....	620
Static URL .....	621
Customize text when generated items using Url.....	622
Icons in Vue Breadcrumb component .....	622
Loading Icons in Breadcrumb Item .....	622

Icon Position.....	625
Icon only.....	627
Show icon only for first item.....	629
Navigation in Vue Breadcrumb component .....	630
URL .....	630
Enable navigation for last Breadcrumb item .....	632
Open URL in new page or tab .....	633
Template in Vue Breadcrumb component .....	634
Item Template.....	634
Separator Template .....	635
Customize Specific Item Template.....	637
Overflow in Vue Breadcrumb component.....	638
Collapsed.....	640
Menu.....	641
Wrap.....	642
Scroll.....	642
Hidden.....	643
None.....	644
Accessibility in Vue Breadcrumb component .....	644
WAI-ARIA attributes.....	645
Keyboard interaction .....	645
Ensuring accessibility .....	646
See also .....	646
Bullet Chart .....	646
Getting Started with the Vue Bullet chart Component in Vue 2 .....	646
Prerequisites .....	646
Dependencies.....	646
Setting up the Vue 2 project .....	647
Add Syncfusion Vue packages.....	647
Add Syncfusion Vue component.....	648
Run the project .....	649
Module Injection.....	649
BulletChart With Data .....	649
Add Bullet Chart Title.....	650
Ranges.....	651

Tooltip .....	652
Getting Started with the Vue Bullet Chart Component in Vue 3 .....	653
Prerequisites .....	653
Set up the Vite project .....	653
Add Syncfusion Vue packages.....	654
Add Syncfusion Vue component.....	655
Run the project .....	658
See also .....	658
Bullet chart dimensions in Vue Bullet chart component .....	659
Size for Container .....	659
bulletChart { .....	659
Size for Bullet Chart.....	660
Axis customization in Vue Bullet chart component.....	662
MajorTickLines and MinorTickLines Customization.....	662
Tick Placement .....	663
Label Format .....	664
GroupingSeparator .....	665
Custom Label Format.....	666
Label Placement.....	667
Opposed Position .....	668
Category Label .....	668
Category Label Customization .....	669
Data binding in Vue Bullet chart component .....	670
Ranges in Vue Bullet chart component.....	671
Color Customization.....	672
Value bar in Vue Bullet chart component.....	673
Types of actual bar .....	674
Actual bar customization .....	674
Comparative bar in Vue Bullet chart component .....	676
Types of target bar .....	677
Target bar customization .....	678
Title in Vue Bullet chart component .....	678
Title .....	678
Subtitle .....	679
Title and SubTitle Position .....	680

Title Customization .....	683
SubTitle Customization .....	684
Customization in Vue Bullet chart component .....	685
Orientation.....	685
Right-to-left (RTL).....	686
Animation.....	687
Theme .....	687
Data label in Vue Bullet chart component.....	688
Data Label Customization .....	689
Tool tip in Vue Bullet chart component.....	690
Default Tooltip .....	690
Tooltip Template.....	691
Customize the Appearance of Tooltip .....	691
Tooltip Customization .....	691
Accessibility in Vue Bullet chart component .....	691
WAI-ARIA attributes.....	692
Keyboard interaction .....	693
Ensuring accessibility .....	693
See also .....	693
ButtonGroup .....	693
Getting Started with the Vue Button group Component in Vue 2 .....	693
Prerequisites .....	693
Dependencies.....	693
Setting up the Vue 2 project .....	693
Add Syncfusion Vue packages.....	694
Import Syncfusion CSS styles .....	695
Add Syncfusion Vue component.....	695
Creating Vue Sample.....	695
Run the project .....	697
Orientation.....	697
See Also .....	698
Getting Started with the Vue ButtonGroup Component in Vue 3.....	698
Prerequisites .....	699
Set up the Vite project .....	699
Add Syncfusion Vue packages.....	700



Import Syncfusion CSS styles .....	701
Add Syncfusion Vue component .....	701
Run the project .....	702
See also .....	703
Types and styles in Vue Button group component.....	703
ButtonGroup types .....	703
ButtonGroup styles .....	704
See Also .....	705
Selection and nesting in Vue Button group component.....	705
Selection.....	705
Nesting .....	707
See Also .....	709
Accessibility in Vue Button group component.....	709
Keyboard interaction .....	710
Ensuring accessibility .....	710
See also .....	711
Style and appearance in Vue Button group component .....	711
How To .....	711
Create buttongroup with icons in Vue Button group component.....	711
Create buttongroup with rounded corner in Vue Button group component.....	712
Disable in Vue Button group component .....	713
Enable ripple in Vue Button group component .....	714
Enable rtl in Vue Button group component.....	714
Form submit in Vue Button group component.....	715
Initialize buttongroup using util function in Vue Button group component .....	716
Show buttongroup selected state on initial render in Vue Button group component.....	718
Button .....	719
Getting Started with the Vue Button Component in Vue 2 .....	719
Prerequisites .....	719
Dependencies.....	719
Setting up the Vue 2 project .....	719
Add Syncfusion Vue packages.....	720
Import Syncfusion CSS styles .....	720
Add Syncfusion Vue component.....	720
Run the project .....	722

Change Button type .....	722
See Also .....	723
Getting Started with the Vue Button Component in Vue 3 .....	723
Prerequisites .....	723
Set up the Vite project .....	723
Add Syncfusion Vue packages.....	725
Import Syncfusion CSS styles .....	725
Add Syncfusion Vue component.....	725
Run the project .....	727
See also .....	727
Types and styles in Vue Button component .....	727
Button styles .....	727
Button types.....	728
Icons .....	732
Button size .....	734
See Also .....	735
Style and appearance in Vue Button component.....	735
Accessibility in Vue Button component .....	735
WAI-ARIA attributes.....	736
Keyboard interaction .....	737
Ensuring accessibility .....	737
See also .....	737
How To .....	737
Add link to a button in Vue Button component .....	737
Create a block button in Vue Button component.....	738
Customize button appearance in Vue Button component .....	738
Customize input and anchor elements in Vue Button component .....	739
Repeat button in Vue Button component .....	740
Right to left in Vue Button component.....	741
Set the disabled state in Vue Button component.....	742
Tooltip for button in Vue Button component.....	742
Calendar .....	743
Getting Started with the Vue Calendar Component in Vue 2.....	743
Prerequisites .....	743
Dependencies.....	743

Setting up the Vue 2 project .....	744
Adding Syncfusion packages .....	744
Import Syncfusion CSS styles .....	745
Add Syncfusion Vue component .....	745
Running the Project .....	747
See Also .....	748
Getting Started with the Vue Calendar Component in Vue 3 .....	748
Prerequisites .....	749
Set up the Vite project .....	749
Add Syncfusion Vue packages .....	750
Import Syncfusion CSS styles .....	750
Add Syncfusion Vue component .....	751
Run the project .....	752
Setting the value, min and max dates .....	753
See Also .....	754
Date range in Vue Calendar component .....	755
Globalization in Vue Calendar component .....	756
Right-to-Left .....	758
Customization in Vue Calendar component .....	759
Disable weekends .....	759
Day cell format .....	760
Highlight Weekends .....	763
See Also .....	764
Multi select in Vue Calendar component .....	764
Calendar views in Vue Calendar component .....	765
View restriction .....	766
Accessibility in Vue Calendar component .....	767
WAI-ARIA attributes .....	768
Keyboard interaction .....	768
Ensuring accessibility .....	770
See also .....	770
Islamic calendar in Vue Calendar component .....	770
Two way binding in Vue Calendar component .....	776
Style appearance in Vue Calendar component .....	777
Customizing the background color for the Calendar .....	777

Customizing the Calendar date elements on hovering.....	778
Customizing the border of date cell grid .....	778
Customizing the Calendar title.....	778
Customizing the previous and next icon.....	778
Customizing the footer button .....	779
Customizing the selected date cell grid .....	779
Customizing the content header in Calendar .....	779
How To .....	780
Set clear button in calendar in Vue Calendar component.....	780
Customize the calendar day header in Vue Calendar component .....	781
Show dates of other months in Vue Calendar component .....	782
Select a sequence of dates in calendar in Vue Calendar component.....	783
Skip a month in calendar in Vue Calendar component .....	785
Render the calendar with week numbers in Vue Calendar component.....	786
Change the first day of week in Vue Calendar component .....	787
Card.....	788
Getting Started with the Vue Card Component in Vue 2 .....	788
Prerequisites .....	788
Dependencies.....	788
Setting up the Vue 2 project .....	788
Adding Syncfusion packages .....	789
Import Syncfusion CSS styles .....	789
Creating Vue Sample.....	789
Adding a header and content .....	790
Running the Project .....	790
See Also .....	791
Getting Started with the Vue Card Component in Vue 3 .....	791
Prerequisites .....	791
Set up the Vite project .....	791
Add Syncfusion Vue packages.....	793
Import Syncfusion CSS styles .....	793
Adding a header and content .....	793
Adding Syncfusion Vue Card component in the application .....	794
Run the project .....	795
Header content in Vue Card component.....	795

Header.....	795
Content .....	797
Card image in Vue Card component .....	798
Images.....	798
Divider .....	799
See Also .....	800
Action buttons in Vue Card component .....	800
Vertical .....	801
See Also .....	802
Horizontal in Vue Card component .....	802
Stacked cards .....	803
Style in Vue Card component .....	804
Customizing the card .....	804
Customizing the Header element .....	804
Customizing the card content.....	805
Divider used to separate the elements inside the card.....	805
Including image within card element .....	805
Including a title or caption for the image .....	805
To include heading image within the header .....	806
Customizing the Header main title .....	806
Customizing the Header subtitle.....	806
Including action buttons or anchor tags .....	806
To align card elements horizontally .....	806
To align elements vertically within the horizontal layout.....	807
How To .....	807
Customize the card image title position in Vue Card component .....	807
Integrate other component inside the card in Vue Card component .....	809
Carousel .....	810
Getting Started with the Vue Carousel Component in Vue 2 .....	810
Prerequisites .....	810
Dependencies.....	810
Setting up the Vue 2 project .....	810
Add Syncfusion Vue packages.....	811
Add Syncfusion Vue component.....	811
Run the project .....	814

Getting Started with Syncfusion Vue Carousel Component in Vue 3 .....	814
Prerequisites .....	814
Setup the Vite project .....	814
Adding Syncfusion Vue packages .....	816
Import Syncfusion CSS styles .....	816
Adding Syncfusion Vue Carousel component in the application .....	817
Run the project .....	821
See also .....	821
Populating items in Vue Carousel component .....	822
Populating items using carousel item .....	822
Populating items using data source .....	823
Selection .....	825
Partial visible slides .....	829
See Also .....	832
Navigators and indicators in Vue Carousel component .....	832
Navigators .....	833
Indicators .....	838
Play button .....	851
Animations and transitions in Vue Carousel component .....	855
Animations .....	855
Intervals between slides .....	858
Auto play slides .....	860
Pause on hover .....	862
Looping slides .....	864
Slide changing events .....	865
Disable touch swiping .....	867
Swipe Modes .....	869
Accessibility in Vue Carousel component .....	871
ARIA attributes .....	872
Keyboard interaction .....	873
Ensuring accessibility .....	873
See also .....	873
Styles and appearance in Vue Carousel component .....	873
CSS Structure in Vue Carousel Control .....	873
Customizing the indicators .....	874

Customizing the navigators.....	875
Customizing partial slides size .....	878
3D Chart .....	879
Getting started with the Vue 3D chart component in Vue 2 .....	879
Prerequisites .....	879
Dependencies.....	879
Setting up the Vue 2 project .....	880
Add Syncfusion Vue packages.....	880
Add Syncfusion Vue component.....	881
Run the project .....	882
Module injection.....	882
Populate 3D chart with data .....	883
Add 3D chart title .....	884
Enable legend.....	885
Add data label.....	886
Enable tooltip.....	888
Getting started with the Vue 3D chart component in Vue 3 .....	889
Prerequisites .....	889
Set up the Vite project .....	889
Add Syncfusion Vue packages.....	891
Add Syncfusion Vue component.....	891
Run the project .....	894
See also .....	894
Chart.....	895
Getting Started with the Vue Chart Component in Vue 2 .....	895
Prerequisites .....	895
Dependencies.....	895
Setting up the Vue 2 project .....	895
Add Syncfusion Vue packages.....	896
Add Syncfusion Vue component.....	896
Run the project .....	897
Module Injection.....	898
Populate Chart with Data.....	898
Add Chart Title .....	900
Enable Legend.....	901

Add Data Label .....	902
Enable Tooltip .....	903
Getting Started with the Vue Chart Component in Vue 3 .....	905
Prerequisites .....	905
Set up the Vite project .....	905
Add Syncfusion Vue packages.....	906
Add Syncfusion Vue component.....	907
Run the project .....	909
See also .....	910
Working with data in Vue Chart component.....	910
Local Data.....	910
Common Datasource .....	911
Remote Data .....	912
Empty points .....	913
Chart dimensions in Vue Chart component.....	915
Size for Container.....	915
Size for Chart.....	916
Category axis in Vue Chart component .....	918
Labels Placement .....	919
Range .....	921
Indexed category axis.....	922
Numeric axis in Vue Chart component .....	923
Range .....	923
Range Padding.....	924
Label Format .....	930
GroupingSeperator .....	931
Custom Label Format.....	932
Date time axis in Vue Chart component.....	933
DateTime Axis .....	933
DateTimeCategory Axis.....	934
Label Format .....	941
Logarithmic axis in Vue Chart component.....	942
Range .....	944
Logarithmic Base.....	945
Logarithmic Interval .....	946



Axis labels in Vue Chart component .....	947
Smart Axis Labels .....	947
Axis Labels Positioning .....	950
Multilevel Labels .....	951
Edge Label Placement .....	957
Labels Customization .....	958
Customizing Specific Point .....	959
Trim using maximum label width.....	960
Line break support .....	961
Axis customization in Vue Chart component.....	962
Axis Crossing .....	962
Title .....	964
Title Rotation.....	965
Tick Lines Customization .....	966
Grid Lines Customization .....	967
Multiple Axis .....	969
Inversed Axis .....	970
Opposed Position .....	971
Strip line in Vue Chart component.....	972
Horizontal Strip lines.....	972
Vertical Striplines .....	973
Customize the strip line .....	974
Customize the stripline text.....	975
Dash Array.....	976
Recurrence Stripline.....	977
Size Type .....	978
Segment Stripline.....	980
Multiple panes in Vue Chart component.....	981
Rows.....	981
Columns .....	984
Chart Types .....	987
Line Chart in Vue Chart component.....	987
Step line in Vue Chart component.....	990
Stacked Line in Chart Vue Chart Component.....	992
100% Stacked Line in Chart Vue Chart Component.....	996

Spline in Vue Chart Component.....	999
Area in Vue Chart Component .....	1001
Range Area in Vue Chart Component .....	1005
Range step area in Vue Chart component .....	1008
Spline Range Area in Vue Chart Component .....	1011
Stacked Area in Vue Chart Component .....	1014
100% Stacked Area in Vue Chart Component.....	1017
Stacked step area in Vue Chart component .....	1020
Step area in Vue Chart component.....	1023
Spline Area in Vue Chart Component .....	1025
Column Chart in Vue Chart Component .....	1028
Range Column in Vue Chart Component .....	1034
Stacked Column in Vue Chart Component .....	1037
100% Stacked Column in Vue Chart Component.....	1041
Bar Chart in Vue Chart Component .....	1044
Stacked Bar in Vue Chart Component.....	1051
100% Stacked Bar in Vue Chart Component.....	1055
Scatter in Vue Chart Component .....	1058
Bubble in Vue Chart Component .....	1061
Polar in Vue Chart Component .....	1064
Radar in Vue Chart Component .....	1074
Hilo in Vue Chart Component .....	1077
High Low Open Close in Vue Chart Component .....	1080
Candle in Vue Chart Component .....	1082
Box and Whisker in Vue Chart Component .....	1085
Watterfall in Vue Chart Component .....	1089
Histogram in Vue Chart Component.....	1091
Error Bar in Vue Chart Component.....	1093
Vertical in Vue Chart Component .....	1100
Pareto in Vue Chart Component.....	1101
Chart series in Vue Chart component.....	1104
Multiple Series .....	1104
Combination Series .....	1105
Enable Complex Property in Series .....	1106
Technical indicators in Vue Chart component.....	1107

Accumulation Distribution .....	1107
Average True Range (ATR) .....	1112
Bollinger Band .....	1116
Exponential Moving Average (EMA) .....	1123
Momentum .....	1126
Moving Average Convergence Divergence (MACD) .....	1134
Relative Strength Index (RSI).....	1142
Simple Moving Average (SMA) .....	1146
Stochastic .....	1150
Trend lines in Vue Chart component .....	1158
Linear.....	1158
Exponential .....	1159
Logarithmic .....	1160
Polynomial .....	1162
Power.....	1163
Moving Average .....	1164
Forecasting.....	1166
Show or hide a trendline.....	1169
Data markers in Vue Chart component .....	1170
Marker.....	1170
Shape.....	1171
Images .....	1172
Customization .....	1173
Customizing specific point .....	1174
Fill marker with series color .....	1175
Data labels in Vue Chart component .....	1176
Position .....	1177
Data Label Template .....	1179
Text Mapping .....	1180
Format.....	1181
Margin.....	1182
DataLabel Rotation .....	1183
Customization .....	1184
Customizing Specific Point .....	1185
Show percentage based on each series points.....	1186

Chart annotations in Vue Chart component.....	1189
Region .....	1190
Co-ordinate Units .....	1192
Alignment.....	1193
Adding y-axis sub title through on annotation .....	1194
Legend in Vue Chart component .....	1196
Position and Alignment.....	1196
Legend Reverse .....	1198
Customization .....	1201
Set the label color based on series color .....	1208
Enable Animation .....	1209
Collapsing Legend Item .....	1210
Legend Title .....	1211
Arrow Page Navigation .....	1212
Legend Item Padding .....	1214
Tooltip in Vue Chart component.....	1215
Default tooltip.....	1215
Fixed tooltip .....	1216
Format the tooltip.....	1217
Individual series format .....	1218
Tooltip template .....	1220
Customize the appearance of tooltip .....	1221
Tooltip mapping name .....	1223
Zooming in Vue Chart component.....	1224
Enable zooming.....	1224
Modes .....	1225
Toolbar .....	1227
Enable pan.....	1228
Eanble scrollbar.....	1229
Auto interval on zooming.....	1230
Data editing in Vue Chart component .....	1232
Enable Data Editing .....	1232
Cross hair and track ball in Vue Chart component .....	1233
Tooltip for axis .....	1235
Customization .....	1236

Trackball.....	1237
Synchronized Charts in Vue Chart component .....	1239
Tooltip synchronization.....	1239
Crosshair synchronization.....	1242
Zooming synchronization.....	1245
Selection synchronization .....	1247
Selection in Vue Chart component .....	1250
Point .....	1251
Series.....	1252
Cluster .....	1253
Rectangular selection.....	1254
Lasso selection .....	1255
Multi-region selection.....	1257
Selection type.....	1258
Selection on load.....	1259
Selection through on legend.....	1261
Customization for selection .....	1262
Chart print in Vue Chart component .....	1263
Print.....	1263
Export.....	1264
Multiple chart export.....	1270
Exporting chart using base64 string.....	1271
Chart appearance in Vue Chart component .....	1273
Custom color palette.....	1273
Data point customization.....	1274
Point level customization.....	1277
Chart area customization.....	1279
Animation.....	1282
Chart title .....	1286
Chart subTitle.....	1291
Render methods in Vue Chart component .....	1293
SVG .....	1293
Canvas .....	1294
Accessibility in Vue Chart component .....	1294
WAI-ARIA attributes.....	1295

Keyboard interaction .....	1295
Ensuring accessibility .....	1296
See also .....	1296
Internationalization in Vue Chart component .....	1296
Localization in Vue Chart component.....	1298
How To .....	1299
Hide tool tip in Vue Chart component.....	1299
Dotted line in Vue Chart component.....	1301
Dynamic chart in Vue Chart component.....	1302
CheckBox.....	1304
Getting Started with the Vue Checkbox Component in Vue 2 .....	1304
Prerequisites .....	1304
Dependencies.....	1304
Setting up the Vue 2 project .....	1304
Add Syncfusion Vue packages.....	1305
Import Syncfusion CSS styles .....	1305
Add Syncfusion Vue component.....	1305
Run the project .....	1307
Change the Checkbox state.....	1307
Getting Started with the Vue CheckBox Component in Vue 3 .....	1309
Prerequisites .....	1309
Set up the Vite project .....	1309
Add Syncfusion Vue packages.....	1310
Import Syncfusion CSS styles .....	1311
Add Syncfusion Vue component.....	1311
Run the project .....	1312
See also .....	1313
Label and size in Vue Check box component.....	1313
Label.....	1313
Size .....	1313
See Also .....	1314
Accessibility in Vue Check box component.....	1314
WAI-ARIA attributes.....	1315
Keyboard interaction .....	1315
Ensuring accessibility .....	1315

See also .....	1316
Two way binding in Vue Check box component .....	1316
Style and appearance in Vue Check box component .....	1317
How To .....	1317
Customized checkbox in Vue Check box component .....	1317
Name and value in form submit in Vue Check box component .....	1322
Right to left in Vue Check box component .....	1323
Chips.....	1323
Getting Started with the Vue Chips Component in Vue 2 .....	1323
Prerequisites .....	1323
Dependencies.....	1323
Setting up the Vue 2 project .....	1324
Add Syncfusion Vue packages.....	1324
Import Syncfusion CSS styles .....	1325
Add Syncfusion Vue component.....	1325
Run the project .....	1326
Getting Started with the Vue Chips Component in Vue 3 .....	1327
Prerequisites .....	1327
Add Syncfusion Vue packages.....	1328
Import Syncfusion CSS styles .....	1329
Add Syncfusion Vue component.....	1329
Run the project .....	1331
See also .....	1331
Types in Vue Chips component.....	1331
Input Chip.....	1332
Choice Chip .....	1332
Filter Chip .....	1332
Action Chip.....	1333
Customization in Vue Chips component.....	1334
Styles .....	1334
Leading Icon .....	1335
Avatar.....	1335
Avatar Content.....	1336
Trailing Icon.....	1337
Outline Chip .....	1337

Style in Vue Chips component .....	1338
Customizing the chip text .....	1338
Customizing the chip icon .....	1338
Customizing the chip delete button.....	1338
Customizing the chip outline .....	1339
Customizing the chip on selection .....	1339
Customizing the chip avatar text .....	1339
Accessibility in Vue Chips component .....	1339
WAI-ARIA attributes.....	1340
Keyboard interaction .....	1341
Ensuring accessibility .....	1341
See also .....	1341
Circular gauge .....	1341
Getting Started with the Vue Circular Gauge Component in Vue 2 .....	1341
Prerequisites .....	1341
Dependencies.....	1341
Setting up the Vue 2 project .....	1342
Add Syncfusion Vue packages.....	1343
Adding Syncfusion Vue Circular Gauge component .....	1343
Run the project .....	1344
Set Pointer Value .....	1344
Getting Started with the Vue Circular Gauge Component in Vue 3 .....	1345
Prerequisites .....	1345
Set up the Vite project .....	1345
Add Syncfusion Vue packages.....	1347
Add Syncfusion Vue component.....	1347
Run the project .....	1350
See also .....	1351
Gauge dimensions in Vue Circular gauge component .....	1351
Size for Container.....	1351
Size for Circular Gauge .....	1351
Gauge axes in Vue Circular gauge component .....	1352
Axis Customization.....	1352
Angles and Direction .....	1353
Axis Radius .....	1355



Ticks.....	1356
Labels .....	1358
Minimum and Maximum .....	1366
Multiple Axes .....	1366
Gauge ranges in Vue Circular gauge component.....	1367
Start and End.....	1367
Customization .....	1368
Rounded Corner Ranges .....	1369
Radius.....	1370
Dragging Range .....	1372
Multiple Ranges .....	1373
Gradient Color.....	1374
See also .....	1380
Gauge pointers in Vue Circular gauge component.....	1380
Needle Pointers.....	1381
RangeBar Pointer .....	1385
Marker Pointer .....	1387
Dragging pointer .....	1389
Multiple Pointers .....	1390
Animation.....	1391
Gradient Color.....	1392
Gauge annotations in Vue Circular gauge component.....	1395
Content .....	1395
Position .....	1396
Sub Gauge .....	1398
See also .....	1400
Animation in Vue Circular Gauge component .....	1400
Gauge legend in Vue Circular gauge component.....	1402
Legend customization .....	1402
Toggle option in legend .....	1405
Paging support in legend .....	1406
Legend text customization.....	1408
Gauge user interaction in Vue Circular gauge component.....	1409
Tooltip for pointer.....	1409
Tooltip for ranges.....	1409

Tooltip for annotation.....	1410
Pointer Drag .....	1413
changes .....	1414
Print and export in Vue Circular Gauge component.....	1417
Print.....	1417
Export.....	1418
Gauge appearance in Vue Circular gauge component .....	1422
Gauge Title .....	1422
Gauge Position .....	1422
Area Customization.....	1424
Radius calculation based on angles .....	1427
Accessibility in Vue Circular gauge component .....	1427
WAI-ARIA attributes.....	1427
Screen reading in Circular Gauge.....	1428
Ensuring accessibility .....	1428
See also .....	1428
Internationalization in Vue Circular Gauge component .....	1428
Globalization .....	1428
Right-to-left.....	1429
ColorPicker .....	1431
Getting Started with the Vue Color picker Component in Vue 2.....	1431
Prerequisites .....	1431
Dependencies.....	1431
Setting up the Vue 2 project .....	1432
Add Syncfusion Vue packages.....	1433
Import Syncfusion CSS styles .....	1433
Add Syncfusion Vue component.....	1433
Inline type .....	1435
See Also .....	1437
Getting Started with the Vue ColorPicker Component in Vue 3.....	1437
Prerequisites .....	1437
Set up the Vite project .....	1437
Add Syncfusion Vue packages.....	1438
Import Syncfusion CSS styles .....	1439
Add Syncfusion Vue component.....	1439

Run the project .....	1440
See also .....	1441
Mode and value in Vue Color picker component .....	1442
Rendering palette at initial load .....	1442
Color value .....	1442
See Also .....	1443
Localization in Vue Color picker component .....	1443
Localization .....	1443
Right to Left - RTL .....	1444
See Also .....	1445
Accessibility in Vue Color picker component.....	1445
WAI-ARIA attributes.....	1446
Keyboard interaction .....	1447
Ensuring accessibility .....	1447
See also .....	1447
Two way binding in Vue Color picker component .....	1447
Style and appearance in Vue Color picker component.....	1448
How To .....	1448
Hide control buttons in Vue Color picker component.....	1448
Render palette alone in Vue Color picker component .....	1449
Colorpicker in dropdownbutton in Vue Color picker component .....	1450
Customize colorpicker in Vue Color picker component.....	1451
Handle no color support in Vue Color picker component .....	1460
Disabled in Vue Color picker component.....	1465
ComboBox.....	1466
Getting Started with the Vue Combo box Component in Vue 2.....	1466
Prerequisites .....	1466
Setting up the Vue 2 project .....	1466
Add Syncfusion Vue packages.....	1467
Import Syncfusion CSS styles .....	1467
Add Syncfusion Vue component.....	1467
Binding data source .....	1468
Run the application .....	1470
Custom values.....	1470
Configure the popup list .....	1471

See Also .....	1473
Getting Started with the Vue ComboBox Component in Vue 3 .....	1473
Prerequisites .....	1473
Set up the Vite project .....	1473
Add Syncfusion Vue packages.....	1474
Import Syncfusion CSS styles .....	1475
Add Syncfusion Vue component.....	1475
Run the project .....	1477
Custom values.....	1477
Configure the popup list .....	1478
See Also .....	1479
Data binding in Vue Combo box component.....	1479
Binding local data.....	1480
Binding remote data .....	1482
See Also .....	1483
Templates in Vue Combo box component .....	1483
Item template .....	1483
Group template.....	1485
Header template .....	1486
Footer template .....	1487
No records template .....	1489
Action failure template .....	1489
See Also .....	1491
Virtualization in ComboBox Component .....	1491
Binding local data.....	1491
Binding remote data .....	1492
Grouping .....	1494
Filtering with Virtualization.....	1495
Grouping in Vue Combo box component .....	1497
Customization .....	1498
See Also .....	1498
Filtering in Vue Combo box component .....	1498
Limit the minimum filter character.....	1499
Change the filter type .....	1500
Case sensitive filtering .....	1502

Diacritics Filtering.....	1503
See Also .....	1504
Localization in Vue Combo box component .....	1504
Loading translations.....	1504
See Also .....	1505
Style in Vue Combo box component.....	1505
Customizing the appearance of wrapper element .....	1505
Customizing the dropdown icon's color .....	1506
Customizing the focus color .....	1506
Customizing the outline theme's focus color .....	1506
Customizing the disabled component's text color .....	1506
Customizing the float label element's focusing color .....	1507
Customizing the color of the placeholder text .....	1507
Customizing the text selection color.....	1507
Customizing the background color of focus, hover, and active item's .....	1507
Customizing the appearance of pop-up element .....	1508
Adding mandatory asterisk to placeholder and float label.....	1508
Accessibility in Vue Combo box component.....	1509
WAI-ARIA attributes.....	1510
Keyboard interaction .....	1510
Ensuring accessibility .....	1512
See also .....	1512
Two way binding in Vue Combo box component .....	1512
How To .....	1513
Autofill in Vue Combo box component.....	1513
Cascading in Vue Combo box component .....	1514
Icons support in Vue Combo box component .....	1516
Context Menu .....	1517
Getting Started with the Vue Context menu Component in Vue 2 .....	1517
Prerequisites .....	1517
Dependencies.....	1517
Setting up the Vue 2 project .....	1518
Add Syncfusion Vue packages.....	1518
Import Syncfusion CSS styles .....	1519
Add Syncfusion Vue component.....	1519

Run the project .....	1522
Rendering items with Separator .....	1522
See Also .....	1524
Getting Started with the Vue ContextMenu Component in Vue 3.....	1524
Prerequisites .....	1525
Set up the Vite project .....	1525
Add Syncfusion Vue packages.....	1526
Import Syncfusion CSS styles .....	1527
Add Syncfusion Vue component.....	1527
Run the project .....	1531
See also .....	1531
Icons and navigation in Vue Context menu component.....	1531
Icons .....	1531
Navigation .....	1534
See Also .....	1536
Template and multilevel nesting in Vue Context menu component.....	1536
Template .....	1536
Multilevel nesting .....	1537
Accessibility in Vue Context menu component .....	1538
WAI-ARIA attributes.....	1539
Keyboard interaction .....	1540
Ensuring accessibility .....	1540
See also .....	1540
Style and appearance in Vue Context menu component .....	1540
How To .....	1540
Open and close contextmenu in Vue Context menu component .....	1540
Change menu items dynamically in Vue Context menu component.....	1542
Template in Vue Context menu component.....	1543
Underline a character in the item text in Vue Context menu component .....	1548
Open a dialog on contextmenu item click in Vue Context menu component.....	1549
Change animation settings in Vue Context menu component.....	1551
Dashboard layout.....	1553
Getting Started with the Vue Dashboard layout Component in Vue 2 .....	1553
Prerequisites .....	1553
Dependencies.....	1553

Setting up the Vue 2 project .....	1553
Add Syncfusion Vue packages.....	1554
Import Syncfusion CSS styles .....	1554
Add Syncfusion Vue component.....	1555
Run the project .....	1559
Getting Started with the Vue Dashboard Layout Component in Vue 3.....	1560
Prerequisites .....	1560
Set up the Vite project .....	1560
Add Syncfusion Vue packages.....	1562
Import Syncfusion CSS styles .....	1562
Add Syncfusion Vue component.....	1563
Run the project .....	1566
See also .....	1566
Setting size of cells in Vue Dashboard layout component.....	1566
Modifying cell size.....	1567
Setting cell spacing.....	1568
Graphical representation of layout.....	1569
Rendering component in right-to-left direction .....	1570
Panels.....	1572
Position sizing of panels in Vue Dashboard layout component.....	1572
Setting header of panels in Vue Dashboard layout component.....	1575
Add remove panels in Vue Dashboard layout component.....	1581
Interaction With Panels .....	1585
Dragging moving of panels in Vue Dashboard layout component .....	1585
Moving panels in Vue Dashboard layout component.....	1592
Resizing of panels in Vue Dashboard layout component .....	1594
Floating of panels in Vue Dashboard layout component .....	1597
Responsive adaptive in Vue Dashboard layout component.....	1599
State maintenance in Vue Dashboard layout component.....	1601
Style in Vue Dashboard layout component .....	1603
Customizing the dashboard layout panel header .....	1603
Customizing the dashboard layout panel content.....	1604
Customizing the dashboard layout panel resize icon .....	1604
Customizing the dashboard layout panel background .....	1604
Accessibility in Vue Dashboard Layout component.....	1604

WAI-ARIA attributes.....	1605
Keyboard interaction .....	1606
Ensuring accessibility .....	1606
See also .....	1606
DatePicker .....	1606
Getting Started with the Vue Datepicker Component in Vue 2 .....	1606
Prerequisites .....	1606
Dependencies.....	1606
Setting up the Vue 2 project .....	1607
Add Syncfusion Vue packages.....	1607
Import Syncfusion CSS styles .....	1608
Add Syncfusion Vue component.....	1608
Run the project .....	1610
Setting the value, min and max dates.....	1611
See Also .....	1612
Getting Started with the Vue DatePicker Component in Vue 3 .....	1612
Prerequisites .....	1612
Set up the Vite project .....	1612
Add Syncfusion Vue packages.....	1614
Import Syncfusion CSS styles .....	1614
Add Syncfusion Vue component.....	1615
Run the project .....	1616
Setting the value, min and max dates.....	1617
See Also .....	1618
Date range in Vue Datepicker component .....	1618
Date format in Vue Datepicker component.....	1619
Date masking in Vue Datepicker component .....	1620
Configure Mask Placeholder .....	1622
Globalization in Vue Datepicker component.....	1624
Right-To-Left .....	1626
Strict mode in Vue Datepicker component.....	1627
Enable Strict Mode.....	1627
Disable Strict Mode.....	1628
Customization in Vue Datepicker component .....	1629
Adding mandatory asterisk to placeholder and float label.....	1631



See Also .....	1631
View in Vue Datepicker component .....	1631
Start view .....	1632
Depth view .....	1632
Accessibility in Vue Datepicker component.....	1633
WAI-ARIA attributes.....	1634
Keyboard Interaction .....	1634
Ensuring accessibility .....	1636
See also .....	1636
Two way binding in Vue Datepicker component.....	1636
Style appearance in Vue Datepicker component .....	1637
Customizing the appearance of DatePicker wrapper element.....	1637
Customizing the DatePicker icon element.....	1638
Customizing the Calendar popup of the DatePicker.....	1638
Full screen mode support in mobiles and tablets.....	1638
How To .....	1641
Disabled the datepicker component in Vue Datepicker component .....	1641
Set the placeholder in Vue Datepicker component.....	1641
Set the readonly in Vue Datepicker component.....	1642
Open datepicker popup on input click in Vue Datepicker component.....	1643
Prevent the popup close in Vue Datepicker component.....	1643
Client side validation in Vue Datepicker component.....	1644
Customize the datepicker day header in Vue Datepicker component .....	1646
DateRangePicker .....	1647
Getting Started with the Vue Daterangepicker Component in Vue 2 .....	1647
Prerequisites .....	1647
Dependencies.....	1647
Setting up the Vue 2 project .....	1648
Add Syncfusion Vue packages.....	1649
Import Syncfusion CSS styles .....	1649
Add Syncfusion Vue component.....	1650
Run the project .....	1652
See Also .....	1654
Getting Started with the Vue DateRangePicker Component in Vue 3 .....	1654
Prerequisites .....	1654

Set up the Vite project .....	1654
Add Syncfusion Vue packages.....	1655
Import Syncfusion CSS styles .....	1656
Add Syncfusion Vue component.....	1656
Run the project .....	1658
Setting the start date and end date .....	1659
See Also .....	1660
Range restriction in Vue Daterangepicker component .....	1660
Restrict the range within a range.....	1660
Range span .....	1662
Strict mode.....	1663
Globalization in Vue Daterangepicker component.....	1663
Right-To-Left .....	1666
Customize the date format .....	1667
Customization in Vue Daterangepicker component.....	1668
Day cell format.....	1668
Preset Ranges.....	1669
First day of week .....	1670
See Also .....	1671
Accessibility in Vue Daterangepicker component .....	1671
WAI-ARIA attributes.....	1672
Keyboard Interaction .....	1673
Ensuring accessibility .....	1674
See also .....	1674
Two way binding in Vue Daterangepicker component.....	1674
Style appearance in Vue Daterangepicker component .....	1675
Customizing the appearance of DateRangePicker wrapper element.....	1675
Customizing the DateRangePicker icon element.....	1676
Customizing the DateRangePicker popup calendar header .....	1676
Customizing the DateRangePicker popup calendar header title .....	1676
Customizing the DateRangePicker popup calendar content .....	1677
Customizing the DateRangePicker popup calendar content title.....	1677
Customizing the DateRangePicker popup calendar previous and next icon .....	1677
Customizing the DateRangePicker popup calendar date cell grid on hovering.....	1677
Customizing the DateRangePicker popup calendar primary button in footer .....	1678

Customizing the DateRangePicker popup calendar cancel button.....	1678
Customizing the footer element in the DateRangePicker popup calendar .....	1678
Customizing the selected date cell grid in the DateRangePicker popup calendar .....	1678
Full screen mode support in mobiles and tablets.....	1679
How To .....	1681
Disable the daterangepicker component in Vue Daterangepicker component .....	1681
Set the placeholder in Vue Daterangepicker component.....	1681
Customization using cssclass in Vue Daterangepicker component .....	1682
Customize the daterangepicker day header in Vue Daterangepicker component .....	1684
DateTimePicker .....	1686
Getting Started.....	1686
Prerequisites .....	1686
Dependencies.....	1686
Get Started with Vue CLI.....	1687
Adding Syncfusion packages .....	1687
Registering Vue Component .....	1687
Creating Vue Sample.....	1688
Adding CSS Reference .....	1689
Running the Application.....	1689
Setting the value,min and max .....	1690
See Also .....	1691
Getting Started with the Vue DateTimePicker Component in Vue 3 .....	1691
Prerequisites .....	1691
Set up the Vite project .....	1691
Add Syncfusion Vue packages.....	1692
Import Syncfusion CSS styles .....	1693
Add Syncfusion Vue component.....	1693
Run the project .....	1695
Setting the value,min and max .....	1695
See Also .....	1697
Globalization in Vue Datetimepicker component.....	1697
Right-To-Left .....	1700
Strict mode in Vue Datetimepicker component .....	1701
Enable Strict Mode.....	1701
Disable Strict Mode.....	1702

Date time range in Vue Datetimepicker component.....	1703
Date time masking in Vue Datetimepicker component.....	1704
Configure Mask Placeholder .....	1706
Customization in Vue Datetimepicker component.....	1707
Day and Time Cell format.....	1707
Adding mandatory asterisk to placeholder and float label.....	1708
See Also .....	1709
Accessibility in Vue Datetimepicker component .....	1709
WAI-ARIA attributes.....	1710
Keyboard Interaction .....	1710
Ensuring accessibility .....	1712
See also .....	1713
Two way binding in Vue Datetimepicker component.....	1713
Style appearance in Vue Datetimepicker component .....	1714
Customizing the appearance of DateTimePicker wrapper element.....	1714
Customizing the DateTimePicker icons element .....	1714
Customizing the time picker popup in the DateTimePicker .....	1714
Customizing the Calendar popup of the DateTimePicker.....	1714
Full screen mode support in mobiles and tablets.....	1715
How To .....	1717
Disable the datetimepicker component in Vue Datetimepicker component.....	1717
Set the placeholder in Vue Datetimepicker component .....	1717
Customize the datetimepicker day header in Vue Datetimepicker component .....	1718
Diagram.....	1720
Getting Started with the Vue Diagram Component in Vue 2 .....	1720
Prerequisites .....	1720
Dependencies.....	1720
Setting up the Vue 2 project .....	1720
Add Syncfusion Vue packages.....	1721
Import Syncfusion CSS styles .....	1721
Add Syncfusion Vue component.....	1722
Run the project .....	1722
Module Injection.....	1723
Flow Diagram .....	1725
Automatic Organization Chart .....	1729

Getting Started with the Vue Diagram Component in Vue 3 .....	1735
Prerequisites .....	1735
Set up the Vite project .....	1735
Add Syncfusion Vue packages.....	1736
Import Syncfusion CSS styles .....	1737
Add Syncfusion Vue component.....	1737
Run the project .....	1744
See also .....	1745
Nodes in Vue Diagram component.....	1745
Create node.....	1746
Add node through nodes collection.....	1746
Add/Remove node at runtime .....	1746
Add node from palette.....	1748
Create node through data source.....	1748
Draw nodes .....	1748
Position .....	1748
Flip.....	1750
Appearance .....	1751
Gradient .....	1752
Linear gradient .....	1752
Radial gradient .....	1754
Shadow.....	1755
Customizing shadow .....	1756
Icon.....	1757
Customizing expand icon .....	1758
Customizing collapse icon .....	1759
Interaction.....	1759
Constraints .....	1759
Custom properties .....	1759
Stack order .....	1759
Data flow .....	1759
See Also.....	1762
Shapes in Vue Diagram component.....	1763
Text .....	1763
Image.....	1764

Image alignment .....	1766
HTML.....	1768
HTML Node With Template .....	1769
Native .....	1771
SVG content alignment .....	1774
Basic shapes .....	1775
Path .....	1776
Flow Shapes .....	1777
Bpmn shapes in Vue Diagram component.....	1778
Event .....	1780
Gateway .....	1783
Activity .....	1784
Tasks.....	1786
Subprocess .....	1788
Event Subprocess .....	1789
Transaction Subprocess .....	1791
Process .....	1792
Loop.....	1792
Compensation .....	1794
Call.....	1796
Ad-Hoc.....	1797
Boundary .....	1798
Data .....	1799
Datasource .....	1801
Artifact .....	1802
Text Annotation .....	1802
Group .....	1803
BPMN Flows .....	1804
Association .....	1804
Sequence.....	1806
Message .....	1807
UML diagram in Vue Diagram component .....	1809
UML Class Diagram .....	1809
Uml Class Diagram Shapes .....	1809
UML Class Relationships .....	1813

How to add UML child at runtime.....	1819
Adding UML Nodes in Symbol palette .....	1824
Editing .....	1827
UML Activity diagram.....	1827
UML Activity diagram Shapes .....	1827
Connectors in Vue Diagram component.....	1831
Create Connector .....	1831
Add connectors through connectors collection.....	1832
Add connector at run time** .....	1833
Connectors from palette.....	1834
Draw connectors .....	1834
Update Connector at runtime.....	1834
Connect nodes .....	1835
Connections with ports .....	1838
Segments.....	1842
Straight.....	1843
Orthogonal .....	1845
Avoid overlapping .....	1848
How to customize Orthogonal Segment Thumb Shape.....	1850
Bezier .....	1852
Avoid overlapping with bezier .....	1856
Decorator .....	1863
Padding .....	1865
Hit padding.....	1866
Flip.....	1867
Bridging .....	1869
Corner radius.....	1871
Appearance .....	1872
Segment Appearance.....	1873
Decorator Appearance.....	1875
Interaction.....	1876
Automatic line routing .....	1876
Constraints .....	1879
Custom Properties .....	1880
Stack Order.....	1881

Enable Connector Splitting.....	1882
See Also .....	1883
Group in Vue Diagram component .....	1884
Create group .....	1884
Add group when initializing diagram .....	1884
Add group at runtime .....	1887
Add children To group at runtime .....	1888
Remove children from group at runtime .....	1888
Container.....	1890
Difference between a basic group and containers .....	1892
Interaction.....	1892
See Also .....	1892
Swim lane in Vue Diagram component.....	1892
Create a swimlane.....	1892
Lanes .....	1898
Phase .....	1911
Add swimlane to palette .....	1917
Limitations.....	1920
Labels in Vue Diagram component .....	1920
Create annotation .....	1921
Add annotations at runtime.....	1922
Remove annotation .....	1923
Update annotation at runtime.....	1924
Alignment.....	1925
Offset.....	1925
Horizontal and vertical alignment.....	1926
Annotation alignment with respect to segments .....	1929
Margin.....	1931
Text align .....	1932
Hyperlink .....	1933
Template Support for Annotation .....	1934
Wrapping.....	1935
Text overflow .....	1937
Appearance .....	1938
Interaction.....	1940



Edit .....	1941
Read-only annotations.....	1941
Drag Limit .....	1942
Multiple annotations .....	1943
Constraints .....	1945
Ports in Vue Diagram component.....	1945
Create Port.....	1946
Add ports when initializing nodes.....	1946
Add ports at runtime.....	1947
Remove ports at runtime.....	1948
Update Port at runtime.....	1950
Appearance .....	1951
Offset.....	1953
Constraints .....	1953
Constraints in Vue Diagram component.....	1953
Diagram constraints .....	1953
Node constraints .....	1954
Connector constraints.....	1955
Port constraints.....	1956
Annotation constraints .....	1957
Selector constraints .....	1958
Snap constraints.....	1958
Boundary constraints.....	1959
Inherit behaviors.....	1960
Bitwise operations .....	1961
Add operation .....	1961
Remove Operation.....	1961
Check Operation .....	1961
Interaction in Vue Diagram component .....	1961
Selection.....	1961
Single selection .....	1961
Selecting a group.....	1962
Multiple selection .....	1962
Select/Unselect elements using program .....	1963
Select entire elements in diagram programmatically.....	1963

Drag.....	1963
Resize .....	1964
Customize the resize-thumb .....	1964
Rotate.....	1966
Connection editing.....	1967
End point handles .....	1967
Straight segment editing.....	1967
Orthogonal thumbs.....	1968
Bezier thumbs .....	1969
Drag and drop nodes over other elements.....	1969
User handles .....	1969
Alignment.....	1970
Offset.....	1970
Side.....	1970
Horizontal and vertical alignments .....	1970
Margin .....	1970
Notification for the mouse button clicked.....	1970
Appearance .....	1971
Zoom pan .....	1973
Zoom pan status.....	1974
Keyboard .....	1976
See Also .....	1976
Tools in Vue Diagram component.....	1976
Drawing tools .....	1976
Shapes .....	1977
Path .....	1978
Connectors .....	1979
Text .....	1980
Polygon shape .....	1981
Polyline Connector .....	1982
Tool selection .....	1983
Events.....	1985
Freehand Drawing.....	1986
Grid lines in Vue Diagram component.....	1987
Customize the gridlines visibility.....	1987

Appearance .....	1988
Line Intervals .....	1989
Snapping .....	1990
Snap To Lines .....	1990
Customization of Snap Intervals .....	1991
Snap To Objects .....	1992
Page settings in Vue Diagram component .....	1993
Page size and appearance .....	1993
Set background image .....	1995
Multiple page and page breaks .....	1996
Boundary constraints .....	1998
Scroll settings in Vue Diagram component .....	1999
Get current scroll status .....	2000
Define scroll status .....	2000
Update scroll status .....	2000
AutoScroll .....	2001
Autoscroll border .....	2003
Scroll limit .....	2004
Scroll padding .....	2005
Scrollable Area .....	2006
UpdateViewport .....	2007
Data binding in Vue Diagram component .....	2008
Local data .....	2008
Remote data .....	2011
CRUD .....	2013
Read DataSource .....	2013
How to perform Editing at runtime .....	2014
InsertData .....	2014
UpdateData .....	2015
DeleteData .....	2016
See Also .....	2017
Automatic layout in Vue Diagram component .....	2018
Layout modes .....	2018
Hierarchical layout .....	2018
Radial tree layout .....	2020

Organizational Chart .....	2023
Symmetric layout .....	2037
Mind Map layout.....	2037
Tree Orientation in layout.....	2038
Complex hierarchical tree .....	2040
Customize layout.....	2047
Accessibility in Vue Diagram component.....	2061
WAI-ARIA attributes.....	2062
Aria-label .....	2062
Keyboard interaction .....	2063
Ensuring accessibility .....	2063
See also .....	2063
Commands in Vue Diagram component.....	2063
Align .....	2064
Distribute .....	2066
Sizing .....	2069
Clipboard.....	2071
Grouping .....	2073
Z-Order command.....	2075
Zoom .....	2080
Nudge command.....	2081
Nudge by using arrow keys .....	2082
BringIntoView .....	2083
BringToCenter .....	2084
FitToPage command .....	2085
Command manager.....	2086
Custom command .....	2086
Modify the existing command .....	2088
See Also .....	2090
Undo redo in Vue Diagram component.....	2090
Undo and redo .....	2090
Undo/redo through shortcut keys .....	2091
Undo/redo through public APIs .....	2091
canLog .....	2094
History change event .....	2096

Stack limit.....	2096
Retain selection.....	2097
Virtualization in Vue Diagram component.....	2098
Virtualization in Diagram .....	2098
Serialization in Vue Diagram component .....	2099
Save .....	2099
Load.....	2100
Prevent default values .....	2101
Export in Vue Diagram component.....	2101
Exporting options .....	2102
File Name .....	2102
Format.....	2102
Margin.....	2103
Mode.....	2103
Region .....	2104
PageSettings.....	2105
Content .....	2106
Custom bounds .....	2106
Export diagram with stretch option.....	2108
Print.....	2109
Limitations.....	2110
Tooltip in Vue Diagram component.....	2110
Default tooltip.....	2110
Common tooltip for all nodes and connectors .....	2110
Tooltip for a specific node/connector.....	2112
Tooltip for Ports .....	2113
Tooltip template content.....	2115
Tooltip alignments .....	2116
Tooltip animation.....	2118
User handle in Vue Diagram component.....	2120
Alignment.....	2120
Fixed user handles .....	2122
Initialization an fixed user handles .....	2122
Customization .....	2123
Customizing the node fixed user handle .....	2125

Customizing the connector fixed user handle .....	2128
Style in Vue Diagram component .....	2132
Customizing the connector end point handle.....	2132
Customizing the connector end point handle when connected.....	2133
Customizing the connector end point handle when disabled .....	2133
Customizing the bezier connector handle .....	2133
Customizing the bezier connector line .....	2133
Customizing the resize handle .....	2133
Customizing the selector pivot line.....	2134
Customizing the selector border.....	2134
Customizing the rotate handle .....	2134
Customizing the symbolpalette while hovering .....	2134
Customizing the symbolpalette when selected .....	2135
Customizing the ruler.....	2135
Customizing the ruler overlap.....	2135
Customizing the text edit.....	2135
Customizing the text edit on selection .....	2136
Ruler in Vue Diagram component.....	2136
Adding Rulers to the Diagram .....	2136
Customizing the Ruler .....	2137
Layers in Vue Diagram component.....	2138
Visible .....	2138
Lock .....	2140
Objects .....	2142
AddInfo.....	2144
Context menu in Vue Diagram component .....	2151
Customize context menu .....	2151
Template Support for Context menu.....	2156
Context menu events.....	2158
Symbol palette in Vue Diagram component.....	2160
Create symbol palette.....	2161
Add palettes to SymbolPalette .....	2162
Customize the Palette Header .....	2171
Restrict expansion of the palette panel.....	2173
Stretch the symbols into the palette .....	2175

Add/Remove symbols to palette at runtime .....	2177
Customize the size of symbols .....	2177
Symbol Preview.....	2179
Default Settings.....	2181
Adding symbol description for symbols in the palette .....	2183
Appearance of symbol description .....	2185
Tooltip for symbols in symbol palette .....	2187
Palette Interaction .....	2191
DragEnter .....	2192
DragLeave .....	2192
DragOver .....	2192
See Also .....	2192
Overview in Vue Diagram component.....	2192
Create overview .....	2192
How to load EJ1 diagram in EJ2 diagram .....	2196
Dialog .....	2197
Getting Started.....	2197
Prerequisites .....	2197
Dependencies.....	2197
Get Started with Vue CLI.....	2197
Adding Syncfusion packages .....	2198
Registering Vue Component .....	2198
Creating Vue sample .....	2199
Adding CSS reference.....	2199
Running the application.....	2200
Modal dialog .....	2201
Enable header .....	2202
Enable footer.....	2203
Draggable .....	2205
Positioning.....	2205
See Also .....	2208
Getting Started with the Vue Dialog Component in Vue 3.....	2208
Prerequisites .....	2208
Set up the Vite project .....	2208
Add Syncfusion Vue packages.....	2210

Import Syncfusion CSS styles .....	2210
Add Syncfusion Vue component .....	2211
Run the project .....	2212
Template in Vue Dialog component .....	2213
Header.....	2213
Footer.....	2213
Content .....	2213
See Also .....	2218
Animation in Vue Dialog component.....	2218
Resize in Vue Dialog component .....	2220
Dialog utility in Vue Dialog component .....	2221
Alert dialog.....	2222
Confirm dialog.....	2224
Close utility dialog.....	2225
Style in Vue Dialog component.....	2227
Customizing the dialog header .....	2227
Customizing the dialog content .....	2227
Customizing modal dialog overlay .....	2227
Customizing the dialog resize icon.....	2227
Customizing the dialog close button.....	2228
Customizing the dialog footer button.....	2228
Localization in Vue Dialog component .....	2228
Loading translations.....	2229
Accessibility in Vue Dialog component.....	2230
WAI-ARIA attributes.....	2231
Keyboard interaction .....	2231
See Also .....	2233
Ensuring accessibility .....	2233
See also .....	2234
How To .....	2234
Create nested dialog in Vue Dialog component .....	2234
Position the dialog on center of the page on scrolling in Vue Dialog component .....	2235
Load dialog content using ajax in Vue Dialog component.....	2237
Render a dialog without header in Vue Dialog component.....	2238
Show dialog with full screen in Vue Dialog component .....	2239



Display a dialog with custom position in Vue Dialog component.....	2240
Prevent closing of modal dialog in Vue Dialog component .....	2242
Prevent the focus on the first element in Vue Dialog component .....	2244
Prevent opening of the dialog in Vue Dialog component.....	2246
Read all the values from dialog on button click in Vue Dialog component .....	2248
Customize the dialog appearance in Vue Dialog component.....	2251
Close dialog while click on outside of dialog in Vue Dialog component.....	2253
Add an icons to dialog buttons in Vue Dialog component .....	2255
Add a minimize maximize buttons in Vue Dialog component .....	2257
Setting max height to the dialog in Vue Dialog component .....	2263
DocumentEditor .....	2264
Overview .....	2264
Key Features.....	2264
Supported Web platforms .....	2265
Getting Started with the Vue DcoumentEditor Component in Vue 2 .....	2266
Prerequisites .....	2266
Dependencies.....	2266
Setting up the Vue 2 project .....	2267
Add Syncfusion Vue packages.....	2267
Import Syncfusion CSS styles .....	2268
Add Syncfusion Vue component.....	2268
Frequently Asked Questions .....	2273
Getting Started with Syncfusion Document Editor Component in Vue 3.....	2273
Prerequisites .....	2273
Creating Vue application using Vue CLI .....	2273
Adding Syncfusion DocumentEditor package in the application .....	2274
Adding CSS reference.....	2274
Adding Component .....	2275
Feature module in Vue Document editor component .....	2281
Import in Vue Document editor component .....	2283
Import document from local machine .....	2284
Convert word documents into SFDT .....	2285
Compatibility with Microsoft Word .....	2289
See Also .....	2290
Server side export in Vue Document editor component.....	2290

SFDT to DOCX export .....	2290
Export in Vue Document editor component.....	2292
SFDT export.....	2292
Word export.....	2293
Text export.....	2294
Export as blob .....	2295
See Also .....	2298
Web services in Vue Document editor component.....	2298
Which operations require server-side interaction.....	2299
Required Web API structure .....	2300
Customize the expected method name.....	2300
Modify the XMLHttpRequest before request send .....	2301
Server Deployment .....	2303
Word processor server docker image overview in Vue Document editor component.....	2303
Provide your license key for activation.....	2303
Provide your license key for activation.....	2304
Provide your license key for activation.....	2305
Provide your license key for activation.....	2306
How to deploy word processor server docker container in azure app service in Vue Document editor component .....	2308
How to deploy word processor server docker container in azure kubernetes service in Vue Document editor component .....	2309
How to publish documenteditor web api application in azure app service from visual studio in Vue Document editor component .....	2312
How to deploy documenteditor java web api in azure in Vue Document editor component .....	2314
Accessibility in Vue Document editor component.....	2316
Keyboard interaction .....	2317
Ensuring accessibility .....	2317
See also .....	2317
Collaborative Editing (preview).....	2317
Prerequisites .....	2318
How to enable collaborative editing in client side.....	2318
How to enable collaborative editing in ASP.NET Core.....	2322
Image in Vue Document editor component .....	2328
Image resizing .....	2330
Changing size.....	2330

Text wrapping style .....	2330
Positioning the image .....	2330
See Also .....	2330
Shapes in Vue Document editor component.....	2331
Supported shapes .....	2331
Text box Shape .....	2331
Shape Resizer .....	2331
Text wrapping style .....	2332
Positioning the shape.....	2332
Text wrapping style in Vue Document editor component.....	2332
In-Line with Text.....	2332
In Front of Text.....	2332
Top and Bottom .....	2332
Behind .....	2333
Square .....	2333
Bookmark in Vue Document editor component.....	2333
Add bookmark.....	2334
Select Bookmark .....	2334
Delete Bookmark .....	2334
Get Bookmark from document .....	2334
Get Bookmark from selection .....	2334
Replace bookmark content .....	2334
Show or Hide bookmark.....	2335
Bookmark Dialog.....	2335
See Also .....	2336
Link in Vue Document editor component.....	2336
Navigate a hyperlink .....	2336
Copy link.....	2338
Add hyperlink .....	2338
Customize screen tip.....	2340
Remove hyperlink .....	2340
Hyperlink dialog .....	2340
See Also .....	2341
Create a table.....	2342
Insert rows .....	2342

Delete table.....	2343
Delete row.....	2343
Delete column.....	2343
Merge cells.....	2344
Positioning the table .....	2344
How to work with tables.....	2344
See Also .....	2347
Table of contents in Vue Document editor component .....	2347
Inserting table of contents.....	2347
Update or edit table of contents .....	2350
See Also .....	2351
Header footer in Vue Document editor component .....	2351
Go to header footer region.....	2351
Link to previous.....	2352
Header and footer distance .....	2352
Close header footer region .....	2353
See Also .....	2353
Text format in Vue Document editor component .....	2353
Bold .....	2353
Italic.....	2353
Underline property .....	2354
Strikethrough property .....	2354
Superscript property .....	2354
Subscript property .....	2355
Size .....	2355
Color.....	2356
Font .....	2356
Highlight color.....	2356
Toolbar with options for text formatting.....	2356
See Also .....	2360
Paragraph format in Vue Document editor component .....	2360
Indentation.....	2360
Special indentation .....	2360
Increase indent .....	2360
Decrease indent .....	2361

Text alignment .....	2361
Line spacing and its type .....	2361
Paragraph spacing.....	2361
Pagination properties.....	2362
Paragraph Border .....	2362
Show or Hide Paragraph marks.....	2363
Toolbar with paragraph formatting options .....	2363
See Also .....	2367
Styles in Vue Document editor component.....	2367
Styles definition overview .....	2367
Default style .....	2367
Style hierarchy .....	2367
Defining new styles .....	2368
Defining a character style .....	2368
Defining a paragraph style .....	2369
Defining a linked style .....	2369
Applying a style .....	2370
Get Styles .....	2371
Modify an existing style .....	2371
List format in Vue Document editor component.....	2372
Create bullet list .....	2372
Create numbered list .....	2372
Clear list.....	2372
Working with lists .....	2373
Editing numbered list.....	2374
See Also .....	2375
Table format in Vue Document editor component .....	2375
Cell margins.....	2375
Background color .....	2376
Cell spacing.....	2376
Cell vertical alignment.....	2376
Table alignment .....	2376
Cell width .....	2377
Table width .....	2377
Apply borders.....	2377

Working with row formatting .....	2378
Row height .....	2378
Header row .....	2378
Allow row break across pages.....	2378
See Also .....	2379
Section format in Vue Document editor component .....	2379
Page size.....	2379
Page margins.....	2379
Header distance .....	2380
Footer distance .....	2380
Columns .....	2380
Breaks.....	2380
See Also .....	2381
Comments in Vue Document editor component.....	2381
Add a new comment.....	2381
Comment navigation.....	2381
Delete comment .....	2381
Delete all comment.....	2381
Protect the document in comments only mode.....	2381
Track changes in Vue Document editor component .....	2383
Enable track changes in Document Editor .....	2383
Get all tracked revisions.....	2383
Accept or Reject all changes programmatically .....	2384
Accept or reject a specific revision .....	2384
Navigate between the tracked changes .....	2385
Filtering changes based on user.....	2385
Protect the document in track changes only mode.....	2386
Event .....	2387
Fields in Vue Document editor component.....	2389
Adding Fields .....	2389
Update fields.....	2389
Get field info .....	2389
Set field info .....	2390
See Also .....	2390
Form fields in Vue Document editor component .....	2390

Insert form field .....	2390
Get form field names .....	2391
Get form field properties .....	2391
Set form field properties.....	2391
Export form field data .....	2392
Import form field data .....	2392
Reset form fields .....	2392
Protect the document in form filling mode .....	2392
Clipboard in Vue Document editor component.....	2394
Copy .....	2394
Cut .....	2394
Paste.....	2394
Local paste .....	2394
Paste with formatting .....	2395
See Also .....	2396
History in Vue Document editor component.....	2396
Enable or disable history.....	2396
Undo and redo .....	2397
Stack size .....	2397
See Also .....	2398
Find and replace in Vue Document editor component .....	2398
Options pane.....	2398
Search.....	2400
Search results .....	2401
SearchResultsChange event.....	2402
Customize find and replace.....	2402
Keyboard shortcut in Vue Document editor component .....	2404
Text formatting .....	2404
Paragraph formatting.....	2404
Clipboard.....	2405
Keyboard shortcut to navigate around the document .....	2405
Keyboard shortcut to extend selection.....	2406
Find and Replace .....	2406
Create, Save and Print document .....	2406
Edit Operation.....	2406

Insert special characters .....	2407
Dialog .....	2407
See Also .....	2407
Scrolling zooming in Vue Document editor component.....	2407
Zooming .....	2410
Page Fit Type .....	2411
Zoom option using UI.....	2412
Print in Vue Document editor component .....	2417
Improve print quality .....	2419
Print using window object .....	2420
Page setup.....	2420
Dialog in Vue Document editor component .....	2422
Font Dialog .....	2422
Paragraph dialog .....	2423
Table dialog.....	2424
Bookmark dialog .....	2425
Hyperlink dialog .....	2426
Table of contents dialog.....	2426
Styles Dialog .....	2428
Style dialog.....	2429
List dialog .....	2430
Borders and shading dialog.....	2431
Table options dialog.....	2432
Table properties dialog .....	2434
Page setup dialog .....	2435
See Also .....	2436
Right to left in Vue Document editor component .....	2436
Chart in Vue Document editor component .....	2443
Supported Chart Types .....	2462
Restrict editing in Vue Document editor component.....	2462
Set current user .....	2462
Highlighting the text area .....	2462
Restrict Editing Pane .....	2463
See Also .....	2468
Spell check in Vue Document editor component .....	2468



Features .....	2469
Enable SpellCheck .....	2469
Disable SpellCheck .....	2470
Spell check settings .....	2470
Context menu.....	2472
Global local in Vue Document editor component .....	2473
Localization .....	2473
Document Editor .....	2474
Color Picker .....	2478
Notes in Vue Document editor component.....	2479
Insert footnotes .....	2479
Insert endnotes.....	2480
Update or edit footnotes and endnotes .....	2482
How To .....	2482
Override the keyboard shortcuts in Vue Document editor component.....	2482
Customize context menu in Vue Document editor component.....	2484
Customize tool bar in Vue Document editor component.....	2492
Change document view in Vue Document editor component .....	2494
Open default document in Vue Document editor component .....	2494
Read only by default in Vue Document editor component .....	2499
Open document by address in Vue Document editor component.....	2504
Deploy document editor component for mobile in Vue Document editor component .....	2506
Disable optimized text measuring in Vue Document editor component .....	2508
Get the selected content in Vue Document editor component .....	2510
Set default format in document editor in Vue Document editor component .....	2513
Show hide spinner in Vue Document editor component .....	2518
Resize document editor in Vue Document editor component.....	2521
Export document as pdf in Vue Document editor component .....	2523
Customize font family drop down in Vue Document editor component .....	2528
Auto save document in document editor in Vue Document editor component.....	2530
Retrieve the bookmark content as text in Vue Document editor component .....	2534
Get current word in Vue Document editor component .....	2540
Insert page number and navigate to page in Vue Document editor component.....	2543
Move selection to specific position in Vue Document editor component .....	2548
Disable header and footer edit in document editor in Vue Document editor component.....	2550

Insert text in current position in Vue Document editor component.....	2555
Change the cursor color in document editor in Vue Document editor component .....	2560
Hide tool bar and properties pane in Vue Document editor component .....	2561
Insert text or image in table programmatically in Vue Document editor component .....	2563
Change the default search highlight color in Vue Document editor component.....	2566
How to optimize the SFDT file .....	2568
How to disable auto focus in Syncfusion Vue Document Editor component.....	2569
How to disable drag and drop in document editor in vue Document editor component.....	2569
Enable ruler .....	2570
DropDownButton.....	2573
Getting Started with the Vue Dropdown Button Component in Vue 2 .....	2573
Prerequisites .....	2573
Dependencies.....	2573
Setting up the Vue 2 project .....	2573
Add Syncfusion Vue packages.....	2574
Import Syncfusion CSS styles .....	2575
Add Syncfusion Vue component.....	2575
Run the project .....	2577
See Also .....	2578
Getting Started with the Vue DropDownButton Component in Vue 3 .....	2578
Prerequisites .....	2578
Set up the Vite project .....	2578
Add Syncfusion Vue packages.....	2580
Import Syncfusion CSS styles .....	2580
Add Syncfusion Vue component.....	2580
Run the project .....	2583
See also .....	2583
Icons in Vue Drop down button component .....	2583
DropDownButton icons.....	2583
Vertical button .....	2587
See Also .....	2589
Popup items in Vue Drop down button component .....	2589
Icons .....	2589
Navigations .....	2591
Template .....	2593

Separator.....	2598
See Also .....	2600
Accessibility in Vue Drop down button component .....	2600
WAI-ARIA attributes.....	2601
Keyboard interaction .....	2602
Ensuring accessibility .....	2602
See also .....	2602
Style and appearance in Vue Drop down button component .....	2602
How To .....	2602
Change caret icon in Vue Drop down button component.....	2602
Create dropdownbutton with rounded corner in Vue Drop down button component .....	2604
Create right to left dropdownbutton in Vue Drop down button component .....	2605
Customize icon and width in Vue Drop down button component .....	2606
Disable a dropdownbutton in Vue Drop down button component .....	2607
Group popup items with listview component in Vue Drop down button component.....	2609
Hide dropdown arrow in Vue Drop down button component .....	2610
Open a dialog on popup item click in Vue Drop down button component.....	2611
Position popup open in Vue Drop down button component .....	2613
Underline a character in the item text in Vue Drop down button component .....	2614
DropDownList .....	2615
Getting Started with the Vue Drop down list Component in Vue 2 .....	2615
Prerequisites .....	2615
Setting up the Vue 2 project .....	2615
Add Syncfusion Vue packages.....	2616
Import Syncfusion CSS styles .....	2616
Add Syncfusion Vue component.....	2617
Binding data source .....	2617
Run the project .....	2619
Configure the Popup List.....	2620
See Also .....	2621
Getting Started with the Vue DropDownList Component in Vue 3 .....	2621
Prerequisites .....	2621
Set up the Vite project .....	2621
Import Syncfusion CSS styles .....	2623
Add Syncfusion Vue component.....	2623

Run the project .....	2625
Configure the Popup List.....	2625
See Also .....	2627
Data binding in Vue Drop down list component.....	2627
Binding local data.....	2627
Binding remote data .....	2630
See Also .....	2631
Templates in Vue Drop down list component .....	2631
Item template .....	2631
Value template.....	2632
Group template.....	2633
Header template .....	2635
Footer template .....	2637
No records template .....	2638
Action failure template .....	2639
See Also .....	2640
Virtualization in DropDown List .....	2640
Binding local data.....	2640
Binding remote data .....	2642
Grouping .....	2643
Filtering with Virtualization.....	2644
Grouping in Vue Drop down list component .....	2646
Customization .....	2647
See Also .....	2647
Filtering in Vue Drop down list component.....	2647
Limit the minimum filter character.....	2648
Change the filter type .....	2650
Case sensitive filtering .....	2651
Diacritics Filtering.....	2652
See Also .....	2653
Localization in Vue Drop down list component .....	2653
Loading translations.....	2653
See Also .....	2655
Style in Vue Drop down list component .....	2655
Customizing the appearance of wrapper element .....	2655

Customizing the dropdown icon's color .....	2655
Customizing the focus color .....	2655
Customizing the outline theme's focus color .....	2655
Customizing the disabled component's text color .....	2656
Customizing the float label element's focusing color .....	2656
Customizing the color of the placeholder text .....	2656
Customizing the background color of focus, hover, and active item's .....	2657
Customizing the appearance of pop-up element .....	2657
Adding mandatory asterisk to placeholder and float label.....	2657
Accessibility in Vue Drop down list component .....	2658
WAI-ARIA attributes.....	2659
Keyboard interaction .....	2659
Ensuring accessibility .....	2661
See also .....	2661
Two way binding in Vue Drop down list component.....	2661
How To .....	2662
Add item in Vue Drop down list component .....	2662
Cascading in Vue Drop down list component.....	2664
Clear item in Vue Drop down list component.....	2665
Close popup in Vue Drop down list component .....	2666
Group header in Vue Drop down list component.....	2667
Highlight filtering in Vue Drop down list component .....	2668
Icons support in Vue Drop down list component .....	2670
Incremental search in Vue Drop down list component .....	2671
Modify data in Vue Drop down list component .....	2671
Multiple cascading in Vue Drop down list component.....	2672
Remote data bind in Vue Drop down list component .....	2674
Remove item in Vue Drop down list component.....	2675
Search on filtering in Vue Drop down list component.....	2676
Tooltip in Vue Drop down list component.....	2677
Value change in Vue Drop down list component .....	2679
Value support in Vue Drop down list component .....	2680
Dropdown Tree .....	2680
Getting Started with the Vue Drop down tree Component in Vue 2 .....	2680
Prerequisites .....	2680

Setting up the Vue 2 project .....	2680
Add Syncfusion Vue packages.....	2681
Import Syncfusion CSS styles .....	2681
Add Syncfusion Vue component.....	2682
Binding data source .....	2682
Run the project .....	2686
Getting Started with the Vue Dropdown Tree Component in Vue 3.....	2686
Prerequisites .....	2687
Set up the Vite project .....	2687
Add Syncfusion Vue packages.....	2688
Import Syncfusion CSS styles .....	2688
Add Syncfusion Vue component.....	2689
Run the project .....	2691
See also .....	2692
Data binding in Vue Drop down tree component.....	2692
Local data .....	2692
Remote data.....	2696
Prevent Node selection.....	2697
Templates in Vue Drop down tree component .....	2698
Item template .....	2698
Header template .....	2700
Footer template .....	2701
No records template .....	2703
Action failure template .....	2704
Custom template to show selected items in input.....	2705
Checkbox in Vue Drop down tree component.....	2707
Auto Check .....	2708
Select All.....	2710
Localization in Vue Drop down tree component .....	2711
Accessibility in Vue Dropdown Tree component.....	2711
WAI-ARIA attributes.....	2712
Keyboard interaction .....	2713
Ensuring accessibility .....	2714
See also .....	2714
FileManager .....	2714

Getting Started with the Vue File manager Component in Vue 2 .....	2714
Prerequisites .....	2714
Dependencies.....	2714
Setting up the Vue 2 project .....	2715
Add Syncfusion Vue packages.....	2716
Import Syncfusion CSS styles .....	2716
Add Syncfusion Vue component.....	2716
Run the project .....	2718
File Download support.....	2718
File Upload support.....	2719
Image Preview support .....	2719
Injecting feature modules .....	2720
Switching initial view of the File Manager .....	2721
Maintaining component state on page reload .....	2722
Rendering component in right-to-left direction .....	2724
Specifying the current path of the File Manager .....	2725
Getting Started with the Vue File Manager Component in Vue 3 .....	2726
Prerequisites .....	2726
Set up the Vite project .....	2726
Add Syncfusion Vue packages.....	2728
Import Syncfusion CSS styles .....	2728
Add Syncfusion Vue component.....	2729
Run the project .....	2731
See also .....	2731
User interface in Vue File manager component .....	2732
Toolbar .....	2733
Files and folders navigation .....	2733
View .....	2734
Context menu.....	2735
File operations in Vue File manager component.....	2736
Folder Upload support .....	2737
File operation request and response Parameters .....	2742
File request and response contents.....	2758
Action Buttons .....	2759
Views in Vue File manager component .....	2760

LargeIcons View .....	2760
Details View.....	2761
Customization in Vue File manager component.....	2762
Context menu customization.....	2763
Details view customization .....	2764
Navigation pane customization .....	2765
Show/Hide file extension.....	2766
Show/Hide hidden items.....	2767
Show/Hide thumbnail images in large icons view .....	2768
Toolbar customization .....	2769
Upload customization .....	2771
Tooltip customization .....	2772
Multiple selection in Vue File manager component.....	2773
Drag and drop in Vue File manager component.....	2774
File system provider in Vue File manager component .....	2776
ASP.NET Core file system provider .....	2776
ASP.NET MVC 5 file system provider .....	2778
ASP.NET Core Azure cloud file system Provider .....	2780
ASP.NET MVC Azure cloud file system Provider .....	2782
ASP.NET Core Amazon S3 cloud file provider .....	2784
ASP.NET MVC Amazon S3 cloud file provider.....	2786
File Transfer Protocol file system provider .....	2787
SQL database file system provider.....	2789
Google Drive file system provider.....	2792
NodeJS file system provider.....	2794
Firebase Realtime Database file system provider.....	2796
IBM Cloud Object Storage file provider .....	2803
Localization in Vue File manager component.....	2805
Accessibility in Vue File Manager component .....	2811
WAI-ARIA attributes.....	2812
Keyboard interaction .....	2813
Ensuring accessibility .....	2814
See also .....	2814
Access control in Vue File manager component.....	2815
Access Rules .....	2815



Permissions .....	2816
How To .....	2818
Adding custom item to context menu in Vue File manager component.....	2818
Adding custom item to toolbar in Vue File manager component .....	2819
Enable disable toolbar item in Vue File manager component.....	2821
Customize custom thumbnail in Vue File manager component.....	2823
Nested items in Vue File manager component.....	2840
Create the custom file provider using NodeJS.....	2844
FloatingActionButton .....	2863
Getting Started with the Vue Floating action button Component in Vue 2 .....	2863
Prerequisites .....	2863
Dependencies.....	2864
Setting up the Vue 2 project .....	2864
Add Syncfusion Vue packages.....	2865
Import Syncfusion CSS styles .....	2865
Add Syncfusion Vue component .....	2865
Run the project .....	2867
Click event .....	2867
Getting Started with Syncfusion Floating Action Button Component in Vue 3 .....	2868
Prerequisites .....	2868
Set up the Vite project .....	2868
Add Syncfusion Vue packages.....	2870
Import Syncfusion CSS styles .....	2870
Adding Syncfusion Vue Floating Action Button Component in the Application.....	2871
Run the project .....	2872
See also .....	2873
Icons in Vue Floating action button component .....	2873
FAB with icon .....	2873
FAB with icon and text .....	2874
Styles in Vue Floating action button component.....	2878
FAB styles .....	2878
Styles customization .....	2879
Show text on hover .....	2879
Positions in Vue Floating action button component .....	2880
Custom position .....	2884

Events in Vue Floating action button component .....	2885
created .....	2885
onclick .....	2886
Form Validator .....	2887
Validation rules in Vue Form validator component.....	2887
Default Rules .....	2887
Error messages in Vue Form validator component .....	2888
Customizing Error Messages.....	2889
Validation events in Vue Form validator component.....	2890
Validation Begin .....	2890
Validation Complete .....	2891
Gantt .....	2893
Getting Started with the Vue Gantt Component in Vue 2.....	2893
Prerequisites .....	2893
Dependencies.....	2893
Setting up the Vue 2 project .....	2894
Add Syncfusion Vue packages.....	2895
Import Syncfusion CSS styles .....	2895
Add Syncfusion Vue component.....	2895
Module injection.....	2896
Binding Gantt with data .....	2896
Mapping task fields .....	2897
Defining timeline.....	2899
Enabling toolbar .....	2901
Enabling editing.....	2903
Enabling predecessors or task relationships.....	2910
Assigning resources.....	2912
Enable filtering .....	2914
Enable sorting .....	2916
Defining eventmarkers.....	2918
Run the application .....	2919
Getting Started with the Vue Gantt Component in Vue 3.....	2921
Prerequisites .....	2921
Set up the Vite project .....	2921
Add Syncfusion Vue packages.....	2923

Import Syncfusion CSS styles .....	2923
Add Syncfusion Vue component .....	2924
Run the project .....	2930
Binding Gantt with data .....	2932
Mapping task fields .....	2934
Defining timeline.....	2937
Enable Toolbar .....	2939
Enabling editing.....	2942
Enabling predecessors or task relationships.....	2953
Assigning resources.....	2955
Enable filtering .....	2959
Enable Sorting .....	2962
Defining eventmarkers.....	2965
Running the application .....	2967
Module in Vue Gantt component .....	2967
Data Binding.....	2968
Data binding in Vue Gantt component .....	2968
Immutable in Vue Gantt component .....	2997
Loading animation in Vue Gantt component.....	2998
Columns .....	3000
Columns in Vue Gantt component.....	3000
Column reordering in Vue Gantt component .....	3009
Column resizing in Vue Gantt component .....	3012
Column template in Vue Gantt component .....	3015
Column menu in Vue Gantt component .....	3018
Responsive columns in Vue Gantt component .....	3022
Check box columns in Vue Gantt component.....	3023
Column spanning in Vue Gantt component.....	3024
Timeline.....	3026
Time line in Vue Gantt component .....	3026
Top tier and bottom tier in Vue Gantt component .....	3036
Zooming in Vue Gantt component.....	3042
Event markers in Vue Gantt component .....	3046
Holidays in Vue Gantt component .....	3047
Splitter in Vue Gantt component .....	3048

Splitter.....	3048
Change splitter position dynamically.....	3049
Scheduling Tasks .....	3051
Task scheduling in Vue Gantt component .....	3051
Task dependency in Vue Gantt component.....	3065
Taskbar.....	3076
Taskbar in Vue Gantt component .....	3076
Baseline in Vue Gantt component .....	3089
Task labels in Vue Gantt component .....	3091
Data markers in Vue Gantt component .....	3091
Critical path in Vue Gantt component .....	3093
Customize taskbar in critical path.....	3094
Toolbar in Vue Gantt component .....	3095
Built-in toolbar items .....	3096
Custom toolbar items .....	3097
Built-in and custom items in toolbar .....	3098
Enable/disable toolbar items.....	3099
Add input elements in toolbar .....	3101
Managing Tasks.....	3102
Managing tasks in Vue Gantt component .....	3102
Adding new tasks in Vue Gantt component .....	3108
Editing tasks in Vue Gantt component .....	3113
Deleting tasks in Vue Gantt component .....	3125
Task bar editing in Vue Gantt component .....	3128
In dent and out dent in Vue Gantt component .....	3132
Splitting and merging tasks in Vue Gantt component .....	3134
Maintaining data in server in Vue Gantt component .....	3136
Validation in Vue Gantt control .....	3142
Scrolling in Vue Gantt component.....	3148
Set width and height.....	3148
Responsive with the parent container.....	3149
Scroll To Date method .....	3150
Set the vertical scroll position.....	3151
Virtual scroll in Vue Gantt component .....	3152
Row virtualization .....	3152

Timeline virtualization .....	3154
Limitations for virtual scroll .....	3156
Resources in Project View .....	3156
Resources in Vue Gantt component .....	3156
Work in Vue Gantt component.....	3161
Resource View.....	3166
Resource view in Vue Gantt component .....	3166
Resource Multi Taskbar in Vue Gantt component.....	3175
Filtering .....	3182
Filtering in Vue Gantt component .....	3182
Filter menu in Vue Gantt component .....	3193
Excel like filter in Vue Gantt component .....	3194
Searching in Vue Gantt component .....	3196
Sorting in Vue Gantt component .....	3205
Sorting column on Gantt initialization .....	3206
Sorting column dynamically .....	3208
Clear all the sorting dynamically .....	3210
Sorting events .....	3211
Touch interaction .....	3213
Selection.....	3214
Selection in Vue Gantt component.....	3214
Row selection in Vue Gantt component .....	3223
Cell selection in Vue Gantt component .....	3231
Rows.....	3236
Rows in Vue Gantt component .....	3236
Drag and Drop in Vue Gantt component .....	3249
Export.....	3256
Excel Export.....	3256
Pdf Export.....	3269
Context menu in Vue Gantt component.....	3297
Custom context menu items.....	3298
Touch interaction .....	3300
State persistence in Vue Gantt component.....	3300
Get or set localStorage value .....	3300
Prevent columns from persisting .....	3301

Persist the header template and header Text .....	3302
Accessibility in Vue Gantt component .....	3304
WAI-ARIA attributes .....	3305
Keyboard navigation .....	3306
Ensuring accessibility .....	3307
See also .....	3307
Global local in Vue Gantt component.....	3307
Localization .....	3307
Internationalization.....	3311
Right to left (RTL) .....	3312
See Also .....	3315
Style and appearance in Vue Gantt component.....	3315
Grid lines .....	3316
Timezone in Vue Gantt component.....	3317
Understanding date manipulation in JavaScript.....	3317
Display same time everywhere with no time difference.....	3318
CRUD operations with timezone.....	3319
Timezone methods .....	3322
How To .....	3324
Open add edit dialog in Vue Gantt component .....	3324
Change schedule dates in Vue Gantt component .....	3325
Coppypasterecords in Vue Gantt component .....	3326
Maintain record index in Vue Gantt component.....	3328
Custom field in Vue Gantt component .....	3331
Maintain zoom to fit in Vue Gantt component.....	3334
Drag and drop from another in Vue Gantt component.....	3336
New row position in Vue Gantt component .....	3339
Grid.....	3342
Getting Started with the Vue Grid Component in Vue 2 .....	3342
Prerequisites .....	3342
Setting up the Vue 2 project .....	3342
Add Syncfusion Vue packages.....	3343
Import Syncfusion CSS styles .....	3343
Add Syncfusion Vue component.....	3344
Register the Syncfusion Vue component.....	3344

Vue 3 Application .....	3344
Defining Row Data .....	3345
Defining Columns .....	3346
Module injection .....	3347
Enable Paging .....	3347
Enable Sorting .....	3348
Enable Filtering .....	3349
Enable Grouping.....	3351
Enable Aggregate .....	3352
Run the project .....	3354
See Also .....	3356
Getting Started with the Vue Grid Component in Vue 3 .....	3356
Prerequisites .....	3356
Set up the Vite project .....	3356
Add Syncfusion Vue packages.....	3357
Import Syncfusion CSS styles .....	3358
Add Syncfusion Vue component.....	3358
Run the project .....	3362
See also .....	3362
Module in Vue Grid component .....	3362
Data Binding.....	3363
Data binding in Vue Grid component .....	3363
Local data in Vue Grid component.....	3376
Remote data in Vue Grid component .....	3378
How to change the data source or columns dynamically in Vue Grid component .....	3386
Immutable mode in Vue Grid component .....	3388
Limitations.....	3394
Performance tips for Vue DataGrid Component .....	3394
How to improve loading performance by binding large dataset.....	3394
How to improve loading performance by binding large data by showing custom text or element .....	3395
How to improve loading performance by referring individual script and CSS .....	3395
How to update cell values without frequent server calls .....	3395
How to optimize server-side data operations with adaptors.....	3396
How to avoid MaxJsonLength error while passing large amount of records .....	3396

Microsoft excel limitation while exporting millions of records to excel file format.....	3397
Columns .....	3397
Columns in Vue Grid component.....	3397
Auto generated columns in Vue Grid component .....	3425
Headers in Vue Grid component.....	3429
Column template in Vue Grid component.....	3433
Complex data binding in Vue Grid component.....	3438
Foreign key column in Vue Grid component .....	3438
Column reorder in Vue Grid component .....	3445
Column resizing in Vue Grid component .....	3449
Column chooser in Vue Grid component.....	3459
Column menu in Vue Grid component .....	3461
Column spanning in Vue Grid component.....	3467
Row .....	3470
Row in Vue Grid component.....	3470
Row height in Vue Grid component.....	3476
Row template in Vue Grid component .....	3477
Detail template in Vue Grid component.....	3480
Row drag and drop in Vue Grid component .....	3483
Row spanning in Vue Grid component .....	3486
Cell .....	3490
Editing .....	3490
Edit in Vue Grid component.....	3490
Edit types in Vue Grid component .....	3497
In line editing in Vue Grid component.....	3513
Dialog editing in Vue Grid component.....	3523
Template editing in Vue Grid component .....	3528
Batch editing in Vue Grid component.....	3532
Command column editing in Vue Grid component .....	3540
Validation in Vue Grid component .....	3542
Persisting data in server in Vue Grid component .....	3546
Sorting in Vue Grid component .....	3555
Initial Sort.....	3556
Multi-column sorting .....	3557
Sort order .....	3558



Sort foreign key column based on Text .....	3558
Sorting Events .....	3560
Sort Comparer.....	3561
Touch Interaction.....	3562
See Also .....	3563
Grouping .....	3563
Grouping in Vue Grid component.....	3563
Caption template in Vue Grid component.....	3570
Reordering on grouped columns in Vue Grid component.....	3571
Lazy load grouping in Vue Grid component.....	3572
Filtering .....	3576
Filtering in Vue Grid component.....	3576
Filter bar in Vue Grid component .....	3581
Filter menu in Vue Grid component .....	3584
Excel like filter in Vue Grid component.....	3591
Searching in Vue Grid component .....	3594
Initial search .....	3595
Search operators.....	3596
Search by external button.....	3596
Search Specific Columns .....	3598
Clear search by external button.....	3598
Search on each key stroke .....	3600
Perform search operation in Grid using multiple keywords.....	3601
See also .....	3603
Paging in Vue Grid component .....	3603
Template .....	3604
Pager with Page Size Dropdown .....	3606
How to render Pager at the Top of the Grid.....	3607
See Also .....	3608
Scrolling in Vue Grid component .....	3608
Set width and height.....	3608
Responsive with parent container .....	3609
Sticky Header .....	3610
Scroll To Selected Row.....	3611
Hide the scrollbar when the content is not overflown .....	3612

Frozen in Vue Grid component .....	3613
Frozen Grid Limitations .....	3614
Freeze Direction .....	3614
Deprecated Methods .....	3615
Virtual scroll in Vue Grid component.....	3617
Row Virtualization .....	3617
Column Virtualization .....	3618
Virtualization with Grouping .....	3621
Limitations for virtual scrolling .....	3621
Browser height limitation in virtual scrolling and solution .....	3622
Infinite scroll in Vue Grid component.....	3627
InitialBlocks .....	3629
Cache Mode .....	3630
Limitations for Infinite Scrolling .....	3631
Selection.....	3632
Selection in Vue Grid component .....	3632
Row selection in Vue Grid component .....	3634
Cell selection in Vue Grid component.....	3640
Column selection in Vue Grid component .....	3642
Check box selection in Vue Grid component .....	3643
Aggregates .....	3646
Aggregates in Vue Grid component .....	3646
Footer aggregate in Vue Grid component .....	3646
Group and caption aggregate in Vue Grid component.....	3650
Custom aggregate in Vue Grid component.....	3652
Reactive aggregate in Vue Grid component .....	3653
Print in Vue Grid component .....	3657
Page Setup .....	3657
Print by external button.....	3658
Print visible Page .....	3658
Print the hierarchy grid .....	3659
Print the master detail grid .....	3661
Print large number of columns .....	3662
Show or Hide columns while Printing .....	3663
Limitations of Printing Large Data.....	3664

Adaptive in Vue Grid component .....	3664
Render adaptive dialogs.....	3664
Vertical row rendering .....	3668
Hierarchy grid in Vue Grid component .....	3674
ExpandAll by external button .....	3675
Expand child grid initially .....	3677
Dynamically load child grid data .....	3678
Bind hierarchy grid with different field.....	3679
Adding Record in ChildGrid .....	3680
Dynamically bind data to child grid based on parent row Data .....	3681
State Persistence.....	3683
State persistence in Vue Grid component .....	3683
Get or set local storage value in Vue Grid component .....	3686
Prevent to persist in Vue Grid component .....	3686
Add to persist in Vue Grid component .....	3688
Tool Bar .....	3690
Tool bar in Vue Grid component.....	3690
Tool bar items in Vue Grid component.....	3693
Custom tool bar in Vue Grid component.....	3699
Pdf Export.....	3700
Pdf export in Vue Grid component .....	3700
Export multiple grids in Vue Grid component .....	3705
Pdf export options in Vue Grid component .....	3708
Pdf cell style customization in Vue Grid component .....	3721
Adding header and footer in Vue Grid component .....	3723
Exporting hierarchy grid in Vue Grid component .....	3729
Exporting grid with templates in Vue Grid control .....	3730
Exporting grid in server in Vue Grid component .....	3740
Excel Export.....	3748
Excel exporting in Vue Grid component .....	3748
Export multiple grids in Vue Grid component .....	3753
Excel export options in Vue Grid component .....	3755
Excel cell style customization in Vue Grid component .....	3766
Adding header and footer in Vue Grid component .....	3771
Exporting hierarchy grid in Vue Grid component .....	3773

Exporting grid with templates in Vue Grid control .....	3774
Exporting grid in server in Vue Grid component .....	3784
Global local in Vue Grid component .....	3794
Localization .....	3794
Internationalization.....	3798
Right to Left - RTL.....	3800
See Also .....	3801
Accessibility in Vue Grid component .....	3801
WAI-ARIA attributes.....	3801
Keyboard interaction .....	3802
Ensuring accessibility .....	3804
See also .....	3804
Clipboard in Vue Grid component .....	3804
Copy to clipboard by external buttons .....	3805
AutoFill .....	3806
Paste.....	3807
Context menu in Vue Grid component .....	3808
Custom context menu items.....	3810
Show context menu on left click.....	3811
Enable or disable context menu items .....	3813
Loading animation in Vue Grid component.....	3814
Style And Appearance.....	3815
Style and appearance in Vue Grid component .....	3815
Header in Vue Grid component .....	3817
Paging in Vue Grid component .....	3817
Sorting in Vue Grid component .....	3819
Filtering in Vue Grid component.....	3819
Grouping in Vue Grid component.....	3821
Tool bar in Vue Grid component.....	3822
Editing in Vue Grid component.....	3823
Aggregate in Vue Grid component .....	3824
Selection in Vue Grid component .....	3825
How To .....	3825
Register the grid using vue component in Vue Grid component .....	3825
Enable disable grid and its actions in Vue Grid component .....	3826

Hide sorting in excel filter in Vue Grid component.....	3828
Cascading drop down list with grid editing in Vue Grid component .....	3829
Perform grid actions by keyboard short cut keys in Vue Grid component.....	3831
Exporting grid in cordova application in Vue Grid component.....	3833
Customize pager drop down in Vue Grid component .....	3834
Hide the expand collapse icon in parent row in Vue Grid component.....	3835
Complex column as foreign key column in Vue Grid component.....	3837
List of array of objects in Vue Grid component .....	3838
Add params for filtering in Vue Grid component .....	3839
Select grid rows based on certain condition in Vue Grid component .....	3840
Collapse grouped rows at initial render in Vue Grid component .....	3841
Grouped row page size in Vue Grid component.....	3842
Get row cell index in Vue Grid component.....	3843
Display null values at bottom in Vue Grid component .....	3844
Enable editing in single click in Vue Grid component.....	3845
Render date picker in frozen columns in Vue Grid component.....	3848
Edit template from another vue in Vue Grid component.....	3849
ShipAddress { .....	3851
Access grid instance in Vue Grid component.....	3854
Get parent column instance in Vue Grid component .....	3856
How to create a routing sample in Vue Grid component .....	3857
Testing the Vue application using Jest.....	3859
Customize the Empty Record Template in Vue Grid component.....	3865
HeatMap Chart.....	3866
Getting Started with the Vue HeatMap Component in Vue 2 .....	3866
Prerequisites .....	3866
Dependencies.....	3866
Setting up the Vue 2 project .....	3867
Add Syncfusion Vue packages.....	3868
Adding Syncfusion Vue HeatMap component .....	3868
Run the project .....	3869
Module injection .....	3870
Populate heat map with data .....	3870
Enable axis labels .....	3871
Add heat map title .....	3872

Enable legend.....	3873
Add data label.....	3874
Add custom cell palette .....	3875
Enable tooltip.....	3877
Getting Started with the Vue HeatMap Component in Vue 3.....	3878
Prerequisites .....	3878
Set up the Vite project .....	3878
Add Syncfusion Vue packages.....	3880
Add Syncfusion Vue component.....	3880
Run the project .....	3885
See also .....	3886
Working with data in Vue Heatmap chart component.....	3886
Data adaptor .....	3886
Empty points .....	3895
Binding nested JSON data .....	3896
See Also .....	3899
Bubble heatmap in Vue Heatmap chart component.....	3899
Bubble types .....	3899
Rendering mode in Vue Heatmap chart component.....	3914
Axis in Vue HeatMap chart component.....	3916
Types .....	3916
Inversed axis.....	3920
Opposed axis.....	3921
Axis labels customization .....	3923
Axis intervals .....	3930
Axis label increment.....	3932
Limiting labels in date-time axis.....	3933
Multilevel Labels .....	3936
Palette in Vue Heatmap chart component .....	3939
Palette types .....	3939
Defining color stops .....	3942
Color range.....	3943
See Also .....	3945
Legend in Vue Heatmap chart component.....	3945
Legend types .....	3947

Placement .....	3948
Alignment.....	3949
Legend dimensions .....	3951
Paging for legend .....	3952
Smart Legend .....	3954
Legend Selection .....	3955
Legend Title .....	3957
Appearance in Vue HeatMap chart component.....	3958
Cell customization.....	3958
Background color .....	3963
Margin.....	3964
Title .....	3965
Data label .....	3966
See Also .....	3977
Dimensions in Vue Heatmap chart component.....	3978
Size for container .....	3978
Size for heat map .....	3978
In pixel.....	3978
In percentage .....	3979
Tooltip in Vue Heatmap chart component .....	3980
Default tooltip.....	3980
Tooltip template .....	3981
Customize the appearance of Tooltip.....	3983
Selection in Vue HeatMap chart component.....	3984
Enable single cell selection .....	3986
Appearance in Vue HeatMap chart component.....	3988
Ensuring accessibility .....	3989
See also .....	3989
Events in Vue HeatMap chart component.....	3989
cellClick.....	3989
cellDoubleClick.....	3990
cellRender .....	3991
cellSelected .....	3993
created .....	3994
legendRender.....	3995

load .....	3996
loaded .....	3997
resized .....	3999
tooltipRender .....	4000
How To .....	4001
Tooltip template in Vue Heatmap chart component.....	4001
Legend label customization in Vue Heatmap chart component.....	4003
ImageEditor.....	4005
Getting Started with the Vue Image Editor Component in Vue 2 .....	4005
Dependencies.....	4005
Prerequisites .....	4005
Setting up the Vue 2 project .....	4005
Add Syncfusion Vue packages.....	4006
Import Syncfusion CSS styles .....	4006
Add Syncfusion Vue component.....	4007
Run the project .....	4008
End-user capabilities in the Image Editor component.....	4009
Open an image .....	4009
Zooming .....	4009
Panning .....	4011
Cropping and image transformation.....	4011
Annotations.....	4012
Filtering and fine-tune .....	4013
Undo and redo the operations .....	4014
Reset an image.....	4015
Export an image .....	4016
Open and save in the Vue Image Editor component.....	4017
Open.....	4017
Supported image formats .....	4018
Save .....	4018
File opened event .....	4020
Saving event.....	4020
Created event.....	4020
Destroyed event.....	4021
Reset an image.....	4021



Selection cropping in the Vue Image Editor component .....	4021
Insert custom / square / circle region.....	4021
Insert selection based on aspect ratio .....	4022
Resize selections .....	4024
Crop an image .....	4024
Cropping event.....	4025
Annotation in the Vue Image Editor component.....	4025
Text annotation.....	4025
Add a text .....	4026
Freehand drawing .....	4031
Shape annotation .....	4035
Delete a shape .....	4038
Image annotation.....	4039
Transform in the vue Image Editor component.....	4041
Rotate an image .....	4041
Flip an image .....	4042
Zoom in or out an image .....	4043
Panning an image.....	4045
Zooming event .....	4046
Rotating event.....	4047
Flipping event.....	4047
Toolbar in the Vue Image Editor component .....	4047
Built-in toolbar items .....	4047
Add a custom toolbar items.....	4048
Show or hide a toolbar.....	4049
Enable or disable a toolbar item.....	4050
Enable or disable a contextual toolbar item.....	4050
Show or hide a toolbar item .....	4050
Toolbar template .....	4051
Customize contextual toolbar.....	4053
Toolbar item clicked event.....	4054
Toolbar created event.....	4055
Add an additional contextual Toolbar item to text shape .....	4055
Quick access toolbar in the Vue Image Editor component.....	4056
Add a custom toolbar item .....	4056

Undo and redo actions in the Vue Image Editor.....	4057
Undo the action .....	4058
Redo the action.....	4058
Filters in the Vue Image Editor component.....	4059
Apply filter effect .....	4059
Image filtering event.....	4061
Finetune in the Vue Image Editor component.....	4061
Adjust the brightness, contrast, or sharpness .....	4061
Adjust the hue, exposure, blur, or opacity .....	4063
Finetune value changing event .....	4064
Frames.....	4064
Apply frame to the Image .....	4064
Frame changing event.....	4066
Resize .....	4067
Apply resize to the image.....	4067
Resizing event .....	4068
Localization in the Vue Image Editor component.....	4068
Accessibility in Vue Image Editor component.....	4071
Keyboard interaction .....	4072
Ensuring accessibility .....	4072
See also .....	4073
InplaceEditor .....	4073
Getting Started with the Vue Inplace editor Component in Vue 2.....	4073
Prerequisites .....	4073
Dependencies.....	4073
Setting up the Vue 2 project .....	4073
Add Syncfusion Vue packages.....	4074
Import Syncfusion CSS styles .....	4075
Add Syncfusion Vue component.....	4075
Add the In-place Editor with Textbox .....	4076
Configure DropDownList.....	4077
Integrate DatePicker .....	4077
Run the project .....	4080
Two-way binding.....	4080
Submitting data to the server (save) .....	4082

Refresh with modified value .....	4083
See Also .....	4084
Getting Started with the Vue In-place Editor Component in Vue 3 .....	4084
Prerequisites .....	4085
Set up the Vite project .....	4085
Add Syncfusion Vue packages.....	4086
Import Syncfusion CSS styles .....	4086
Add Syncfusion Vue component.....	4087
Run the project .....	4089
Configure DropDownList in Vue3.....	4090
Integrate DatePicker in Vue3 .....	4091
Components in Vue Inplace editor component.....	4093
Model configuration .....	4098
See Also .....	4099
Configuration in Vue Inplace editor component .....	4099
Rendering modes .....	4099
Event actions for editing .....	4103
Action on focus out.....	4105
Display modes.....	4106
See Also .....	4108
Buttons in Vue Inplace editor component.....	4108
See Also .....	4110
Server actions in Vue Inplace editor component.....	4110
See Also .....	4113
Data binding in Vue Inplace editor component.....	4113
Local .....	4113
Remote.....	4114
Integration in Vue Inplace editor component .....	4117
As a string.....	4117
As a selector .....	4117
As a template .....	4118
See Also .....	4119
Validation in Vue Inplace editor component .....	4119
Validation Rules .....	4119
Style in Vue Inplace editor component.....	4120

Customizing the In-place Editor text.....	4120
Customizing the In-place Editor action buttons .....	4121
Accessibility in Vue Inplace editor component.....	4121
Keyboard interaction .....	4122
Ensuring accessibility .....	4122
See also .....	4123
Localization in Vue Inplace editor component .....	4123
Localization .....	4123
Right to left .....	4125
Format.....	4126
How To .....	4127
Dynamic edit mode in Vue Inplace editor component.....	4127
Disable edit mode in Vue Inplace editor component .....	4129
Custom indication in Vue Inplace editor component .....	4131
Custom animation in Vue Inplace editor component.....	4132
Kanban .....	4134
Getting Started with the Vue Kanban Component in Vue 2 .....	4134
Prerequisites .....	4134
Setting up the Vue 2 project .....	4134
Add Syncfusion Vue packages.....	4135
Import Syncfusion CSS styles .....	4135
Add Syncfusion Vue component.....	4136
Initialize the Kanban.....	4137
Populating cards.....	4137
Enable swimlane .....	4138
Getting Started with the Vue Kanban Component in Vue 3 .....	4139
Prerequisites .....	4140
Set up the Vite project.....	4140
Add Syncfusion Vue packages.....	4141
Import Syncfusion CSS styles .....	4142
Add Syncfusion Vue component.....	4142
Run the project .....	4146
Columns in Vue Kanban component.....	4146
Single-key mapping .....	4147
Multi-key mapping .....	4148

Header text .....	4148
Header template .....	4149
Toggle columns .....	4152
Stacked headers .....	4154
Cards in Vue Kanban component.....	4155
Drag-and-drop.....	4155
Header.....	4156
Content .....	4158
Template .....	4158
Selection.....	4160
Data binding in Vue Kanban component .....	4161
Local data .....	4161
Remote data.....	4162
Loading data via ajax.....	4174
Swimlane in Vue Kanban component .....	4175
Render swimlane row .....	4175
Custom row text.....	4176
Template .....	4177
Sorting.....	4179
Drag-and-drop.....	4180
Create empty row .....	4181
Calculate cards count.....	4182
Enable frozen rows .....	4183
Drag and drop in Vue Kanban component .....	4184
Internal drag and drop .....	4184
External drag and drop .....	4187
Sort in Vue Kanban component .....	4195
Index.....	4195
DataSource Order .....	4198
Custom .....	4199
Change the direction.....	4200
Dialog in Vue Kanban component .....	4201
Default Dialog.....	4201
Custom Fields.....	4202
Prevent Dialog.....	4208

Persisting data in server.....	4209
Tooltip in Vue Kanban component .....	4216
Tooltip template .....	4216
Validation in Vue Kanban component .....	4218
Minimum card limit.....	4218
Maximum card limit.....	4218
Virtualization in Vue Kanban component .....	4219
Virtual scrolling .....	4219
Limitations for virtual scrolling .....	4224
Localization in Vue Kanban component.....	4224
Loading translations.....	4224
Right to left (RTL) .....	4226
Dimensions in Vue Kanban component.....	4227
Auto height and width .....	4227
Height and width in pixel .....	4228
Height and width in percentage .....	4229
Persistence in Vue Kanban component .....	4230
Responsive mode in Vue Kanban component .....	4231
Layouts .....	4231
Scrolling.....	4233
Selection.....	4234
Style in Vue Kanban component.....	4237
To set fixed position to the Kanban header.....	4240
Accessibility in Vue Kanban component.....	4240
WAI-ARIA attributes.....	4241
Keyboard interaction .....	4242
Ensuring accessibility .....	4242
See also .....	4242
How To .....	4243
Header double click in Vue Kanban component.....	4243
Dynamically change columns in Vue Kanban component .....	4244
Filter cards in Vue Kanban component.....	4245
Search cards in Vue Kanban component .....	4247

## Syncfusion Vue UI Components (Essential JS 2)

Syncfusion Vue is a modern UI Components library that has been built from the ground up to be lightweight, responsive, modular and touch friendly. It also includes complete support for Angular, React, ASP.NET MVC and ASP.NET Core frameworks.

### Components list

The Syncfusion Vue UI components are listed below.

<style>

### table

```
{
border:0 !important;
line-height: 2!important;
}
tr
{
border:0 !important;
}
td
{
border:0 !important;
vertical-align: top;
}
.controlanchorlink
{
text-decoration: none!important;
font-size: 14px!important;
text-align: left!important;
padding: 5px 0px;
letter-spacing: 1px;
}
.controlcategory
{
font-size: 14px!important;
text-align: left!important;
```

font-weight: bold!important;

letter-spacing: 0.7px;

}

}

</style>

	DATA VISUALIZATION	CALENDARS	DROPDOWNS
			<a href="#">AutoComplete</a>
GRIDS	<a href="#">Accumulation Chart</a>	<a href="#">Scheduler</a>	<a href="#">ListBox</a>
<a href="#">DataGrid</a>	<a href="#">Charts</a>	<a href="#">Gantt Chart</a>	<a href="#">ComboBox</a>
<a href="#">Pivot Table</a>	<a href="#">Stock Chart</a>	<a href="#">Calendar</a>	<a href="#">DropDown List</a>
<a href="#">TreeGrid</a>	<a href="#">Circular Gauge</a>	<a href="#">DatePicker</a>	<a href="#">Multiselect DropDown</a>
<a href="#">Spreadsheet</a>	<a href="#">Linear Gauge</a>	<a href="#">DateRangePicker</a>	<a href="#">DropDown Tree</a>
	<a href="#">Maps</a>	<a href="#">DateTime Picker</a>	<a href="#">Mention</a>
FILE VIEWERS & EDITORS	<a href="#">Diagram Component</a>	<a href="#">TimePicker</a>	NAVIGATION
	<a href="#">HeatMap Chart</a>	INPUTS	<a href="#">Accordion</a>
<a href="#">Document Editor</a>	<a href="#">Map</a>	<a href="#">TextBox</a>	<a href="#">Carousel</a>
<a href="#">Image Editor</a>	<a href="#">Range Selector</a>	<a href="#">Input Mask</a>	<a href="#">Context Menu</a>
<a href="#">RichTextEditor</a>	<a href="#">Smith Chart</a>	<a href="#">Numeric TextBox</a>	<a href="#">Menu Bar</a>
<a href="#">PDF Viewer</a>	<a href="#">Sparkline Charts</a>	<a href="#">Masked TextBox</a>	<a href="#">Ribbon</a>
<a href="#">Word Processor</a>	<a href="#">Barcode</a>	<a href="#">RadioButton</a>	<a href="#">Sidebar</a>
	<a href="#">TreeMap</a>	<a href="#">CheckBox</a>	<a href="#">Tabs</a>
LAYOUT	<a href="#">Bullet Chart</a>	<a href="#">Color Picker</a>	<a href="#">Toolbar</a>
<a href="#">Dialog</a>	<a href="#">Kanban</a>	<a href="#">File Upload</a>	<a href="#">TreeView</a>
<a href="#">ListView</a>	BUTTONS	<a href="#">Range Slider</a>	<a href="#">File Manager</a>
<a href="#">Predefined Dialogs</a>	<a href="#">Button</a>	<a href="#">Toggle Switch Button</a>	<a href="#">Breadcrumb</a>
<a href="#">Tooltip</a>	<a href="#">ButtonGroup</a>	<a href="#">Signature</a>	<a href="#">Pager</a>
<a href="#">Splitter</a>	<a href="#">DropDown Menu</a>	<a href="#">Rating</a>	<a href="#">AppBar</a>
<a href="#">Dashboard</a>	<a href="#">Progress Button</a>	FORMS	NOTIFICATION
<a href="#">Card</a>	<a href="#">SplitButton</a>	<a href="#">In-place Editor</a>	<a href="#">Toast</a>
<a href="#">Avatar</a>	<a href="#">Chips</a>	<a href="#">Query Builder</a>	<a href="#">Progress Bar</a>
	<a href="#">Floating Action Button</a>		<a href="#">Spinner</a>
	<a href="#">Speed Dial</a>		



			<a href="#">Badge</a>
			<a href="#">Skeleton</a>
			<a href="#">Message</a>

## How to best read this user guide

- The best way to get started would be to read the "Getting Started" section of the documentation for the component that you would like to start using first. The "Getting Started" guide gives just enough information that you need to know before starting to write code. This is the only section that we recommend reading end-to-end before starting to write code, all other information can be referred as needed.
- Now that you are familiar with the basics of using the component, the next step would be to start integrating the component into your application. A good starting point would be to refer to the code snippets in the [online sample browser](#) which contains hundreds of code samples, it is very likely that you will find a code sample that resembles your intended usage scenario.
- Another valuable resource is the API reference which provides detailed information on the object hierarchy as well as the settings available on every object.

## Getting help

If you are still not able to find the information that you are looking for at the self-help resources mentioned above then please contact us by creating a support ticket in our support site, or ask your query in StackOverflow with tag `syncfusion-ej2`.

Note: Syncfusion does not collect any kind of information when our components are used in customer applications.

## See also

- [Product Development Life Cycle](#)
- [Getting Started with Syncfusion Vue UI Components](#)
- [Getting Started with Syncfusion Vue UI Components in Vue 3](#)

## System Requirements for Vue UI Components

This section explains the basic system requirements to work with Syncfusion Vue UI components.

- Vue supported version  $\geq 2.6+$ .
- Required [node](#) version  $\geq 16+$ , and the supported [npm](#) version will be installed along with the node version. Use the following command to check the node's version.

```
`bash
```

```
node --version
```

```
`
```

The [vue-class-components](#) package is required before the 2023 Volume 1 (v21.1.35) release.

- \* **Vue 2.6** - Use the **7.2.6** version.
- \* **Vue 2.7** - This package is not needed.
- \* **Vue 3+** - Use **^8.0.0-rc.1** version.

## Browser support

The Syncfusion Vue UI components are supported only in modern browsers. For more information, refer to the [browser compatibility](#) section.

## Vue supported versions

The following table represents the supported Vue versions by different Syncfusion Vue UI components releases.

Version	Syncfusion Vue components version
-----	-----
<a href="#">Vue v2.7</a>	20.3.47 and above
<a href="#">Vue v3.0</a>	19.2.44 and above

## Browser Support

The Syncfusion Essential JS 2 components are supported only in modern browsers. This includes the following versions.

Chrome	Firefox	Opera	Edge	IE	Safari	iOS	Android	Windows Mobile
-----	-----	-----	-----	-----	-----	-----	-----	-----
Latest	Latest	Latest	13 +	11 +	9 +	9 +	4.4 +	IE 11 +

## Required polyfills

The following polyfills are required to run Essential JS 2 components in each browser.

Browser	Polyfills
-----	-----
Chrome(latest), Firefox(latest), Opera(latest), Edge, Safari 9+	NONE
IE 11	ES6 Promise

The Syncfusion Essential JS 2 components are supported in IE 11 browser with ES6 Promise polyfill.

## Using CDN

To add ES6 Promise polyfill using a CDN, include this in your HTML file.

```
`ts
```

```
<!-- Automatically provides/replaces Promise if missing or broken. -->
```

```
<script src="https://cdn.jsdelivr.net/npm/es6-promise@4/dist/es6-promise.js"></script>
```

```
<script src="https://cdn.jsdelivr.net/npm/es6-promise@4/dist/es6-promise.auto.js"></script>
```

```
<!-- Minified version of es6-promise-auto below. -->
```

```
<script src="https://cdn.jsdelivr.net/npm/es6-promise@4/dist/es6-promise.min.js"></script>
<script src="https://cdn.jsdelivr.net/npm/es6-promise@4/dist/es6-promise.auto.min.js"></script>
`
```

## Node.js

ES6 Promise polyfill can also be installed on the node.js.

To install:

```
`ts
yarn add es6-promise
(or)
npm install es6-promise
`
```

To Use:

```
`ts
var Promise = require('es6-promise').Promise;
`
```

For further details, refer to the link [here](#).

## Getting started

### Vue 3

Getting Started with Vue UI Components with JavaScript and Composition API

This article provides a step-by-step guide for setting up a [Vite](#) project with a JavaScript environment and integrating the Syncfusion Vue components using the [Composition API](#).

The **Composition API** is a new feature introduced in Vue.js 3 that provides an alternative way to organize and reuse component logic. It allows developers to write components as functions that use smaller, reusable functions called composition functions to manage their properties and behavior.

### Prerequisites

#### [System requirements for Syncfusion Vue UI components](#)

#### [Set up the Vite project](#)

A recommended approach for beginning with Vue is to scaffold a project using [Vite](#). To create a new Vite project, use one of the commands that are specific to either NPM or Yarn.

```
`bash
npm create vite@latest
`

or

`bash
yarn create vite
`
```

`

Using one of the above commands will lead you to set up additional configurations for the project as below:

1. Define the project name: We can specify the name of the project directly. Let's specify the name of the project as `my-project` for this article.

```
`bash
```

```
? Project name: » my-project
```

`

2. Select `Vue` as the framework. It will create a Vue 3 project.

```
`bash
```

```
? Select a framework: » - Use arrow-keys. Return to submit.
```

```
Vanilla
```

```
Vue
```

```
React
```

```
Preact
```

```
Lit
```

```
Svelte
```

```
Others
```

`

3. Choose `JavaScript` as framework variant to build this Vite project using JavaScript and Vue.

```
`bash
```

```
? Select a variant: » - Use arrow-keys. Return to submit.
```

```
JavaScript
```

```
TypeScript
```

```
Customize with create-vue ↗
```

```
Nuxt ↗
```

`

4. Upon completing the aforementioned steps to create the `my-project`, run the following command to install its dependencies:

```
`bash
```

```
cd my-project
```

```
npm install
```

`

or

```
`bash
cd my-project
yarn install
`
```

Now that **my-project** is ready to run with default settings, let's add Syncfusion components to the project.

#### *Add Syncfusion Vue packages*

Syncfusion Vue component packages are available at [npmjs.com](https://www.npmjs.com). To use Syncfusion Vue components in the project, install the corresponding npm package.

This article uses the [Vue Grid component](#) as an example. To use the Vue Grid component in the project, the `@syncfusion/ej2-vue-grids` package needs to be installed using the following command:

```
`bash
npm install @syncfusion/ej2-vue-grids --save
`
```

or

```
`bash
yarn add @syncfusion/ej2-vue-grids
`
```

#### *Import Syncfusion CSS styles*

You can import themes for the Syncfusion Vue component in various ways, such as using CSS or SASS styles from npm packages, CDN, CRG and [Theme Studio](#). Refer to [themes topic](#) to know more about built-in themes and different ways to refer to themes in a Vue project.

In this article, **Material** theme is applied using CSS styles, which are available in installed packages. The necessary **Material** CSS styles for the Grid component and its dependents were imported into the `<style>` section of **src/App.vue** file.

#### ~/SRC/APP.VUE

```
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-buttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-calendars/styles/material.css";
@import "../node_modules/@syncfusion/ej2-dropdowns/styles/material.css";
@import "../node_modules/@syncfusion/ej2-inputs/styles/material.css";
@import "../node_modules/@syncfusion/ej2-navigations/styles/material.css";
@import "../node_modules/@syncfusion/ej2-popups/styles/material.css";
@import "../node_modules/@syncfusion/ej2-splitbuttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
</style>
```

The order of importing CSS styles should be in line with its dependency graph.

### Add Syncfusion Vue component

Follow the below steps to add the Vue Grid component using **Composition API**:

1.First, add the **setup** attribute to the **script** tag to indicate that Vue will be using the **Composition API**. And import the Grid component in the **script** section of the **src/App.vue** file.

#### ~/SRC/APP.VUE

```
<script setup>
import { GridComponent as EjsGrid, ColumnsDirective as EColumns,
ColumnDirective as EColumn } from '@syncfusion/ej2-vue-grids';
</script>
```

2.In the **template** section, define the Grid component with the [dataSource](#) property and column definitions.

#### ~/SRC/APP.VUE

```
<template>
<ejs-grid :dataSource='data'>
<e-columns>
<e-column field='OrderID' width='100' textAlign="Right"></e-column>
<e-column field='CustomerID' width='100'></e-column>
<e-column field='EmployeeID' width='100' textAlign="Right"></e-column>
<e-column field='Freight' width='100' format="C2" textAlign="Right"></e-
column>
<e-column field='ShipCountry' width='100'></e-column>
</e-columns>
</ejs-grid>
</template>
```

3.Declare the values for the **dataSource** property in the **script** section.

#### ~/SRC/APP.VUE

```
<script setup>
const data = [
{
OrderID: 10248, CustomerID: 'VINET', EmployeeID: 5, ShipCountry: 'France',
Freight: 32.38
},
{
OrderID: 10249, CustomerID: 'TOMSP', EmployeeID: 6, ShipCountry: 'Germany',
Freight: 11.61
},
{
OrderID: 10250, CustomerID: 'HANAR', EmployeeID: 4, ShipCountry: 'Brazil',
Freight: 65.83
}
];
</script>
```

Here is the summarized code for the above steps in the **src/App.vue** file:

#### ~/SRC/APP.VUE

```

<template>
<ejs-grid :dataSource='data'>
<e-columns>
<e-column field='OrderID' width='100' textAlign="Right"></e-column>
<e-column field='CustomerID' width='100'></e-column>
<e-column field='EmployeeID' width='100' textAlign="Right"></e-column>
<e-column field='Freight' width='100' format="C2" textAlign="Right"></e-column>
<e-column field='ShipCountry' width='100'></e-column>
</e-columns>
</ejs-grid>
</template>
<script setup>
import { GridComponent as EjsGrid, ColumnsDirective as EColumns,
ColumnDirective as EColumn } from '@syncfusion/ej2-vue-grids';
const data = [
{
OrderID: 10248, CustomerID: 'VINET', EmployeeID: 5, ShipCountry: 'France',
Freight: 32.38
},
{
OrderID: 10249, CustomerID: 'TOMSP', EmployeeID: 6, ShipCountry: 'Germany',
Freight: 11.61
},
{
OrderID: 10250, CustomerID: 'HANAR', EmployeeID: 4, ShipCountry: 'Brazil',
Freight: 65.83
}
];
</script>
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-buttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-calendars/styles/material.css";
@import "../node_modules/@syncfusion/ej2-dropdowns/styles/material.css";
@import "../node_modules/@syncfusion/ej2-inputs/styles/material.css";
@import "../node_modules/@syncfusion/ej2-navigations/styles/material.css";
@import "../node_modules/@syncfusion/ej2-popups/styles/material.css";
@import "../node_modules/@syncfusion/ej2-splitbuttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
</style>

```

### Run the project

To run the project, use the following command:

```
`bash
```

```
npm run dev
```

```
,
```

or

```
`bash
```

```
yarn run dev
```

```
,
```

The output will appear as follows:

OrderID	CustomerID	EmployeeID	Freight	ShipCountry
10248	VINET	5	\$32.38	France
10249	TOMSP	6	\$11.61	Germany
10250	HANAR	4	\$65.83	Brazil

*See also*

- [Getting Started with Vue UI Components using Composition API and TypeScript](#)
- [Getting Started with Vue UI Components using Options API and JavaScript](#)
- [Getting Started with Vue UI Components using Options API and TypeScript](#)

### Getting Started with Vue UI Components with TypeScript and Composition API

This article provides a step-by-step guide for setting up a [Vite](#) project with a TypeScript environment and integrating the Syncfusion Vue components using the [Composition API](#).

The **Composition API** is a new feature introduced in Vue.js 3 that provides an alternative way to organize and reuse component logic. It allows developers to write components as functions that use smaller, reusable functions called composition functions to manage their properties and behavior.

#### *Prerequisites*

#### [System requirements for Syncfusion Vue UI components](#)

#### *Set up the Vite project*

A recommended approach for beginning with Vue is to scaffold a project using [Vite](#). To create a new Vite project, use one of the commands that are specific to either NPM or Yarn.

```
`bash
```

```
npm create vite@latest
```

```
,
```

or

```
`bash
```

```
yarn create vite
```

```
,
```

Using one of the above commands will lead you to set up additional configurations for the project:

1. Define the project name: We can specify the name of the project directly. Let's specify the name of the project as **my-project** for this article.

```
`bash
```

```
? Project name: » my-project
```

```
,
```

2. Select **Vue** as the framework. It will create a Vue 3 project.



```
`bash
```

```
? Select a framework: » - Use arrow-keys. Return to submit.
```

```
Vanilla
```

```
Vue
```

```
React
```

```
Preact
```

```
Lit
```

```
Svelte
```

```
Others
```

```
,
```

3. Choose **TypeScript** as framework variant to build this Vite project using TypeScript and Vue.

```
`bash
```

```
? Select a variant: » - Use arrow-keys. Return to submit.
```

```
JavaScript
```

```
TypeScript
```

```
Customize with create-vue ↗
```

```
Nuxt ↗
```

```
,
```

4. Upon completing the aforementioned steps to create the **my-project**, run the following command to install its dependencies:

```
`bash
```

```
cd my-project
```

```
npm install
```

```
,
```

```
or
```

```
`bash
```

```
cd my-project
```

```
yarn install
```

```
,
```

Now that **my-project** is ready to run with default settings, let's add Syncfusion components to the project.

#### *Add Syncfusion Vue packages*

Syncfusion Vue component packages are available at [npmjs.com](https://www.npmjs.com). To use Syncfusion Vue components in the project, install the corresponding npm package.

This article uses the [Vue Schedule component](#) as an example. To use the Vue Schedule component in the project, the `@syncfusion/ej2-vue-schedule` package needs to be installed using the following command:

```
`bash
npm install @syncfusion/ej2-vue-schedule --save
```

or

```
`bash
yarn add @syncfusion/ej2-vue-schedule
```

### *Import Syncfusion CSS styles*

You can import themes for the Syncfusion Vue component in various ways, such as using CSS or SASS styles from npm packages, CDN, CRG and [Theme Studio](#). Refer to [themes topic](#) to know more about built-in themes and different ways to refer to themes in a Vue project.

In this article, **Material** theme is applied using CSS styles, which are available in installed packages. The necessary **Material** CSS styles for the Schedule component and its dependents were imported into the `<style>` section of `src/App.vue` file.

### ~/SRC/APP.VUE

```
<style>
@import '../node_modules/@syncfusion/ej2-base/styles/material.css';
@import '../node_modules/@syncfusion/ej2-buttons/styles/material.css';
@import '../node_modules/@syncfusion/ej2-calendars/styles/material.css';
@import '../node_modules/@syncfusion/ej2-dropdowns/styles/material.css';
@import '../node_modules/@syncfusion/ej2-inputs/styles/material.css';
@import '../node_modules/@syncfusion/ej2-navigations/styles/material.css';
@import '../node_modules/@syncfusion/ej2-popups/styles/material.css';
@import '../node_modules/@syncfusion/ej2-vue-schedule/styles/material.css';
</style>
```

The order of importing CSS styles should be in line with its dependency graph.

### *Add Syncfusion Vue component*

Follow the below steps to add the Vue Schedule component using **Composition API**:

1.First, add the `setup` attribute to the `script` tag to indicate that Vue will be using the Composition API. And import the Schedule component in the `script` section of the `src/App.vue` file.

### ~/SRC/APP.VUE

```
<script setup>
import { ScheduleComponent as EjsSchedule, ViewsDirective as EViews,
ViewDirective as EView, ResourcesDirective as EResources, ResourceDirective
as EResource, EventSettingsModel, Day, Week, WorkWeek, Month, Agenda,
DragAndDrop, Resize } from "@syncfusion/ej2-vue-schedule";
</script>
```

2. In the `template` section, define the Schedule component with appointments. To populate the empty Scheduler with appointments, define either the local JSON data or remote data through the [dataSource](#) property available within the [eventSettings](#) option. Additionally, it is necessary to include the start and end time fields to define any appointments.

#### ~/SRC/APP.VUE

```
<template>
<div id='app'>
<ejs-schedule height='550px' width='100%' :selectedDate='selectedDate'
:eventSettings='eventSettings'>
<e-views>
<e-view option='Day'></e-view>
<e-view option='Week' startHour='07:00' endHour='15:00'></e-view>
<e-view option='WorkWeek' startHour='10:00' endHour='18:00'></e-view>
<e-view option='Month' showWeekend=false></e-view>
<e-view option='Agenda'></e-view>
</e-views>
<e-resources>
<e-resource field="OwnerId" title="Owner" name="Owners"
:dataSource="ownerDataSource" textField="OwnerText"
idField="Id" colorField="OwnerColor">
</e-resource>
</e-resources>
</ejs-schedule>
</div>
</template>
```

3. Declare the values for the `dataSource` property in the `script` section.

#### ~/SRC/APP.VUE

```
<script setup>
const eventSettings: EventSettingsModel = {
dataSource: [
{
Id: 1,
Subject: 'Surgery - Andrew',
EventType: 'Confirmed',
StartTime: new Date(2021, 7, 10, 9, 0),
EndTime: new Date(2021, 7, 10, 10, 0),
OwnerId: 2
},
{
Id: 2,
Subject: 'Consulting - John',
EventType: 'Confirmed',
StartTime: new Date(2021, 7, 11, 10, 0),
EndTime: new Date(2021, 7, 11, 11, 30),
OwnerId: 3
},
{
Id: 3,
Subject: 'Therapy - Robert',
EventType: 'Requested',
StartTime: new Date(2021, 7, 12, 11, 30),
```

```
EndTime: new Date(2021, 7, 12, 12, 30),
OwnerId: 1
}
]
};
</script>
```

4.To generate a Schedule with particular views, need to inject the relevant modules into the Schedule. This can be accomplished by utilizing the `provide` method within the `src/App.vue` file.

#### ~/SRC/APP.VUE

```
<script setup lang="ts">
import { provide } from "vue";
provide('schedule', [Day, Week, WorkWeek, Month, Agenda, DragAndDrop,
Resize]);
</script>
```

Here is the summarized code for the above steps in the `src/App.vue` file:

#### ~/SRC/APP.VUE

```
<template>
<div id='app'>
<ejs-schedule height='550px' width='100%' :selectedDate='selectedDate'
:eventSettings='eventSettings'>
<e-views>
<e-view option='Day'></e-view>
<e-view option='Week' startHour='07:00' endHour='15:00'></e-view>
<e-view option='WorkWeek' startHour='10:00' endHour='18:00'></e-view>
<e-view option='Month' showWeekend=false></e-view>
<e-view option='Agenda'></e-view>
</e-views>
<e-resources>
<e-resource field="OwnerId" title="Owner" name="Owners"
:dataSource="ownerDataSource" textField="OwnerText"
idField="Id" colorField="OwnerColor">
</e-resource>
</e-resources>
</ejs-schedule>
</div>
</template>
<script setup lang="ts">
import { ScheduleComponent as EjsSchedule, Day, Week, WorkWeek, Month,
Agenda, DragAndDrop, Resize, ViewsDirective as EViews, ViewDirective as
EView, ResourcesDirective as EResources, ResourceDirective as EResource }
from "@syncfusion/ej2-vue-schedule";
import { provide } from "vue";
provide('schedule', [Day, Week, WorkWeek, Month, Agenda, DragAndDrop,
Resize]);
const selectedDate: Date = new Date(2021, 7, 12);
const allowMultiple: boolean = true;
const ownerDataSource: Record<string, any>[] = [
{ OwnerText: 'Nancy', Id: 1, OwnerColor: '#ffaa00' },
{ OwnerText: 'Steven', Id: 2, OwnerColor: '#f8a398' },
{ OwnerText: 'Michael', Id: 3, OwnerColor: '#7499e1' }];
```

```

const eventSettings: EventSettingsModel = {
  dataSource: [
    {
      Id: 1,
      Subject: 'Surgery - Andrew',
      EventType: 'Confirmed',
      StartTime: new Date(2021, 7, 10, 9, 0),
      EndTime: new Date(2021, 7, 10, 10, 0),
      OwnerId: 2
    },
    {
      Id: 2,
      Subject: 'Consulting - John',
      EventType: 'Confirmed',
      StartTime: new Date(2021, 7, 11, 10, 0),
      EndTime: new Date(2021, 7, 11, 11, 30),
      OwnerId: 3
    },
    {
      Id: 3,
      Subject: 'Therapy - Robert',
      EventType: 'Requested',
      StartTime: new Date(2021, 7, 12, 11, 30),
      EndTime: new Date(2021, 7, 12, 12, 30),
      OwnerId: 1
    }
  ]
};
</script>
<style>
@import '../node_modules/@syncfusion/ej2-base/styles/material.css';
@import '../node_modules/@syncfusion/ej2-buttons/styles/material.css';
@import '../node_modules/@syncfusion/ej2-calendars/styles/material.css';
@import '../node_modules/@syncfusion/ej2-dropdowns/styles/material.css';
@import '../node_modules/@syncfusion/ej2-inputs/styles/material.css';
@import '../node_modules/@syncfusion/ej2-navigations/styles/material.css';
@import '../node_modules/@syncfusion/ej2-popups/styles/material.css';
@import '../node_modules/@syncfusion/ej2-vue-schedule/styles/material.css';
</style>

```

### Run the project

To run the project, use the following command:

```
`bash
```

```
npm run dev
```

```
`
```

or

```
`bash
```

```
yarn run dev
```

```
`
```

The output will appear as follows:

<	>	August 08 - 14, 2021	TODAY	DAY	WEEK	WORK WEEK	MONTH	⋮
	Sun 8	Mon 9	Tue 10	Wed 11	Thu 12	Fri 13	Sat 14	
9:00 AM			Surgery - Andrew 9:00 AM - ...					
10:00 AM				Consultin g - John 10:00 AM...				
11:00 AM								

*See also*

- [Getting Started with Vue UI Components using Composition API and JavaScript](#)
- [Getting Started with Vue UI Components using Options API and TypeScript](#)
- [Getting Started with Vue UI Components using Options API and JavaScript](#)

### Getting Started with Vue UI Components with JavaScript and Options API

This article provides a step-by-step guide for setting up a [Vite](#) project with a JavaScript environment and integrating the Syncfusion Vue components using the [Options API](#).

The **Options API** is the traditional way of writing Vue.js components, where the component logic is organized into a series of options that define the component's properties and behavior. These options include data, methods, computed properties, watchers, lifecycle hooks, and more.

#### *Prerequisites*

#### [System requirements for Syncfusion Vue UI components](#)

#### *Set up the Vite project*

A recommended approach for beginning with Vue is to scaffold a project using [Vite](#). To create a new Vite project, use one of the commands that are specific to either NPM or Yarn.

```
`bash
```

```
npm create vite@latest
```

```
,
```

or

```
`bash
```

```
yarn create vite
```

```
,
```

Using one of the above commands will lead you to set up additional configurations for the project as below:

1. Define the project name: We can specify the name of the project directly. Let's specify the name of the project as `my-project` for this article.

```
`bash
```

```
? Project name: » my-project
```

```
,
```

2. Select **Vue** as the framework. It will create a Vue 3 project.

```
`bash
```

```
? Select a framework: » - Use arrow-keys. Return to submit.
```

```
Vanilla
```

```
Vue
```

```
React
```

```
Preact
```

```
Lit
```

```
Svelte
```

```
Others
```

```
,
```

3. Choose **JavaScript** as framework variant to build this Vite project using JavaScript and Vue.

```
`bash
```

```
? Select a variant: » - Use arrow-keys. Return to submit.
```

```
JavaScript
```

```
TypeScript
```

```
Customize with create-vue ↗
```

```
Nuxt ↗
```

```
,
```

4. Upon completing the aforementioned steps to create the **my-project**, run the following command to install its dependencies:

```
`bash
```

```
cd my-project
```

```
npm install
```

```
,
```

```
or
```

```
`bash
```

```
cd my-project
```

```
yarn install
```

```
,
```

Now that `my-project` is ready to run with default settings, let's add Syncfusion components to the project.

#### Add Syncfusion Vue packages

Syncfusion Vue component packages are available at [npmjs.com](https://www.npmjs.com). To use Syncfusion Vue components in the project, install the corresponding npm package.

This article uses the [Vue Grid component](#) as an example. To use the Vue Grid component in the project, the `@syncfusion/ej2-vue-grids` package needs to be installed using the following command:

```
`bash
```

```
npm install @syncfusion/ej2-vue-grids --save
```

```
,
```

```
or
```

```
`bash
```

```
yarn add @syncfusion/ej2-vue-grids
```

```
,
```

#### Import Syncfusion CSS styles

You can import themes for the Syncfusion Vue component in various ways, such as using CSS or SASS styles from npm packages, CDN, CRG and [Theme Studio](#). Refer to [themes topic](#) to know more about built-in themes and different ways to refer to themes in a Vue project.

In this article, Material theme is applied using CSS styles, which are available in installed packages. The necessary Material CSS styles for the Grid component and its dependents were imported into the `<style>` section of `src/App.vue` file.

#### ~/SRC/APP.VUE

```
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-buttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-calendars/styles/material.css";
@import "../node_modules/@syncfusion/ej2-dropdowns/styles/material.css";
@import "../node_modules/@syncfusion/ej2-inputs/styles/material.css";
@import "../node_modules/@syncfusion/ej2-navigations/styles/material.css";
@import "../node_modules/@syncfusion/ej2-popups/styles/material.css";
@import "../node_modules/@syncfusion/ej2-splitbuttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
</style>
```

The order of importing CSS styles should be in line with its dependency graph.

#### Add Syncfusion Vue component

Follow the below steps to add the Vue Grid component using Options API:

1.First, import the Grid component in the `script` section of the `src/App.vue` file.

#### ~/SRC/APP.VUE

```
<script>
```



```
import { GridComponent, ColumnsDirective, ColumnDirective } from
 '@syncfusion/ej2-vue-grids';
</script>
```

2.The next step is to register the Grid component and its child directives in Vue.

#### ~/SRC/APP.VUE

```
import { GridComponent, ColumnsDirective, ColumnDirective } from
 '@syncfusion/ej2-vue-grids';
//Component registration
export default {
  name: "App",
  components: {
    'ejs-grid': GridComponent,
    'e-columns': ColumnsDirective,
    'e-column': ColumnDirective
  }
}
```

3.In the **template** section, define the Grid component with the [dataSource](#) property and column definitions.

#### ~/SRC/APP.VUE

```
<template>
<ejs-grid :dataSource='data'>
<e-columns>
<e-column field='OrderID' width='100' textAlign="Right"></e-column>
<e-column field='CustomerID' width='100'></e-column>
<e-column field='EmployeeID' width='100' textAlign="Right"></e-column>
<e-column field='Freight' width='100' format="C2" textAlign="Right"></e-
column>
<e-column field='ShipCountry' width='100'></e-column>
</e-columns>
</ejs-grid>
</template>
```

4.Declare the values for the **dataSource** property in the **script** section.

#### ~/SRC/APP.VUE

```
data() {
  return {
    data:[
      {
        OrderID: 10248, CustomerID: 'VINET', EmployeeID: 5, ShipCountry: 'France',
        Freight: 32.38
      },
      {
        OrderID: 10249, CustomerID: 'TOMSP', EmployeeID: 6, ShipCountry: 'Germany',
        Freight: 11.61
      },
      {
        OrderID: 10250, CustomerID: 'HANAR', EmployeeID: 4, ShipCountry: 'Brazil',
        Freight: 65.83
      }
    ]
  }
}
```

```

}
],
};
}

```

Here is the summarized code for the above steps in the **src/App.vue** file:

### ~/SRC/APP.VUE

```

<template>
<ejs-grid :dataSource='data'>
<e-columns>
<e-column field='OrderID' width='100' textAlign="Right"></e-column>
<e-column field='CustomerID' width='100'></e-column>
<e-column field='EmployeeID' width='100' textAlign="Right"></e-column>
<e-column field='Freight' width='100' format="C2" textAlign="Right"></e-
column>
<e-column field='ShipCountry' width='100'></e-column>
</e-columns>
</ejs-grid>
</template>
<script>
import { GridComponent, ColumnsDirective, ColumnDirective } from
'@syncfusion/ej2-vue-grids';
// Component registration
export default {
name: "App",
// Declaring component and its directives
components: {
'ejs-grid': GridComponent,
'e-columns': ColumnsDirective,
'e-column': ColumnDirective
},
// Bound properties declarations
data() {
return {
data:[
{
OrderID: 10248, CustomerID: 'VINET', EmployeeID: 5, ShipCountry: 'France',
Freight: 32.38
},
{
OrderID: 10249, CustomerID: 'TOMSP', EmployeeID: 6, ShipCountry: 'Germany',
Freight: 11.61
},
{
OrderID: 10250, CustomerID: 'HANAR', EmployeeID: 4, ShipCountry: 'Brazil',
Freight: 65.83
}
],
};
};
</script>
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";

```

```
@import "../node_modules/@syncfusion/ej2-buttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-calendars/styles/material.css";
@import "../node_modules/@syncfusion/ej2-dropdowns/styles/material.css";
@import "../node_modules/@syncfusion/ej2-inputs/styles/material.css";
@import "../node_modules/@syncfusion/ej2-navigations/styles/material.css";
@import "../node_modules/@syncfusion/ej2-popups/styles/material.css";
@import "../node_modules/@syncfusion/ej2-splitbuttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
</style>
```

### Run the project

To run the project, use the following command:

```
`bash
```

```
npm run dev
```

```
,
```

or

```
`bash
```

```
yarn run dev
```

```
,
```

The output will appear as follows:

OrderID	CustomerID	EmployeeID	Freight	ShipCountry
10248	VINET	5	\$32.38	France
10249	TOMSP	6	\$11.61	Germany
10250	HANAR	4	\$65.83	Brazil

### See also

- [Getting Started with Vue UI Components using Composition API and JavaScript](#)
- [Getting Started with Vue UI Components using Composition API and TypeScript](#)
- [Getting Started with Vue UI Components using Options API and TypeScript](#)

### Getting Started with Vue UI Components with TypeScript and Options API

This article provides a step-by-step guide for setting up a [Vite](#) project with a TypeScript environment and integrating the Syncfusion Vue components using the [Options API](#).

The **Options API** is the traditional way of writing Vue.js components, where the component logic is organized into a series of options that define the component's properties and behavior. These options include data, methods, computed properties, watchers, lifecycle hooks, and more.

### Prerequisites

#### [System requirements for Syncfusion Vue UI components](#)

*Set up the Vite project*

A recommended approach for beginning with Vue is to scaffold a project using [Vite](#). To create a new Vite project, use one of the commands that are specific to either NPM or Yarn.

```
`bash
```

```
npm create vite@latest
```

```
,
```

or

```
`bash
```

```
yarn create vite
```

```
,
```

Using one of the above commands will lead you to set up additional configurations for the project:

1. Define the project name: We can specify the name of the project directly. Let's specify the name of the project as **my-project** for this article.

```
`bash
```

```
? Project name: » my-project
```

```
,
```

2. Select **Vue** as the framework. It will create a Vue 3 project.

```
`bash
```

```
? Select a framework: » - Use arrow-keys. Return to submit.
```

```
Vanilla
```

```
Vue
```

```
React
```

```
Preact
```

```
Lit
```

```
Svelte
```

```
Others
```

```
,
```

3. Choose **TypeScript** as framework variant to build this Vite project using TypeScript and Vue.

```
`bash
```

```
? Select a variant: » - Use arrow-keys. Return to submit.
```

```
JavaScript
```

```
TypeScript
```

```
Customize with create-vue ↗
```

```
Nuxt ↗
```

4. Upon completing the aforementioned steps to create the `my-project`, run the following command to install its dependencies:

```
`bash
cd my-project
npm install
```

or

```
`bash
cd my-project
yarn install
```

Now that `my-project` is ready to run with default settings, let's add Syncfusion components to the project.

#### *Add Syncfusion Vue packages*

Syncfusion Vue component packages are available at [npmjs.com](https://www.npmjs.com). To use Syncfusion Vue components in the project, install the corresponding npm package.

This article uses the [Vue Schedule component](#) as an example. To use the Vue Schedule component in the project, the `@syncfusion/ej2-vue-schedule` package needs to be installed using the following command:

```
`bash
npm install @syncfusion/ej2-vue-schedule --save
```

or

```
`bash
yarn add @syncfusion/ej2-vue-schedule
```

#### *Import Syncfusion CSS styles*

You can import themes for the Syncfusion Vue component in various ways, such as using CSS or SASS styles from npm packages, CDN, CRG and [Theme Studio](#). Refer to [themes topic](#) to know more about built-in themes and different ways to refer to themes in a Vue project.

In this article, `Material` theme is applied using CSS styles, which are available in installed packages. The necessary `Material` CSS styles for the Schedule component and its dependents were imported into the `<style>` section of `src/App.vue` file.

To import the necessary CSS styles for the Schedule component, as well as its dependent styles, into the `src/App.vue` file, you can use the provided code snippet within the `<style>` section. Let's import the `Material` theme for the Schedule component.

~/SRC/APP.VUE

```
<style>
@import '../node_modules/@syncfusion/ej2-base/styles/material.css';
@import '../node_modules/@syncfusion/ej2-buttons/styles/material.css';
@import '../node_modules/@syncfusion/ej2-calendars/styles/material.css';
@import '../node_modules/@syncfusion/ej2-dropdowns/styles/material.css';
@import '../node_modules/@syncfusion/ej2-inputs/styles/material.css';
@import '../node_modules/@syncfusion/ej2-navigations/styles/material.css';
@import '../node_modules/@syncfusion/ej2-popups/styles/material.css';
@import '../node_modules/@syncfusion/ej2-vue-schedule/styles/material.css';
</style>
```

The order of importing CSS styles should be in line with its dependency graph.

*Add Syncfusion Vue component*

Follow the below steps to add the Vue Schedule component using Options API:

1.First, import the Schedule component in the **script** section of the **src/App.vue** file.

~/SRC/APP.VUE

```
<script>
import { ScheduleComponent as EjsSchedule, ViewsDirective as EViews,
ViewDirective as EView, ResourcesDirective as EResources, ResourceDirective
as EResource, EventSettingsModel, Day, Week, WorkWeek, Month, Agenda,
DragAndDrop, Resize } from '@syncfusion/ej2-vue-schedule';
</script>
```

2.The next step is to register the Schedule component and its child directives in Vue.

~/SRC/APP.VUE

```
import { ScheduleComponent, ViewsDirective, ViewDirective,
ResourcesDirective, ResourceDirective, EventSettingsModel, Day, Week,
WorkWeek, Month, Agenda, DragAndDrop, Resize } from '@syncfusion/ej2-vue-
schedule';
//Component registration
export default {
name: 'App',
// Declaring component and its directives
components: {
'ejs-schedule': ScheduleComponent,
'e-views': ViewsDirective,
'e-view': ViewDirective,
'e-resources': ResourcesDirective,
'e-resource': ResourceDirective
}
}
```

3.In the **template** section, define the Schedule component with appointments. To populate the empty Scheduler with appointments, define either the local JSON data or remote data through the [dataSource](#) property available within the [eventSettings](#) option. Additionally, it is necessary to include the start and end time fields to define any appointments.

~/SRC/APP.VUE

```

<template>
<div id='app'>
<ejs-schedule height='550px' width='100%' :selectedDate='selectedDate'
:eventSettings='eventSettings'>
<e-views>
<e-view option='Day'></e-view>
<e-view option='Week' startHour='07:00' endHour='15:00'></e-view>
<e-view option='WorkWeek' startHour='10:00' endHour='18:00'></e-view>
<e-view option='Month' showWeekend=false></e-view>
<e-view option='Agenda'></e-view>
</e-views>
<e-resources>
<e-resource field="OwnerId" title="Owner" name="Owners"
:datasource="ownerDataSource" textField="OwnerText"
idField="Id" colorField="OwnerColor">
</e-resource>
</e-resources>
</ejs-schedule>
</div>
</template>

```

4. Declare the values for the `dataSource` property in the `script` section.

~/SRC/APP.VUE

```

<script lang="ts">
export default {
name: "App",
...
data() {
return {
...
eventSettings: {
dataSource: [
{
Id: 1,
Subject: 'Surgery - Andrew',
EventType: 'Confirmed',
StartTime: new Date(2021, 7, 10, 9, 0),
EndTime: new Date(2021, 7, 10, 10, 0),
OwnerId: 2
},
{
Id: 2,
Subject: 'Consulting - John',
EventType: 'Confirmed',
StartTime: new Date(2021, 7, 11, 10, 0),
EndTime: new Date(2021, 7, 11, 11, 30),
OwnerId: 3
},
{
Id: 3,
Subject: 'Therapy - Robert',
EventType: 'Requested',
StartTime: new Date(2021, 7, 12, 11, 30),

```

```

EndTime: new Date(2021, 7, 12, 12, 30),
OwnerId: 1
}
]
} as EventSettingsModel,
};
}
...
};
</script>

```

5.To generate a Schedule with particular views, need to inject the relevant modules into the Schedule. This can be accomplished by utilizing the `provide` method within the `src/App.vue` file.

#### ~/SRC/APP.VUE

```

<script lang="ts">
export default {
name: "App",
...
provide: {
schedule: [Day, Week, WorkWeek, Month, Agenda, DragAndDrop, Resize]
}
};
</script>

```

Here is the summarized code for the above steps in the `src/App.vue` file:

#### ~/SRC/APP.VUE

```

<template>
<div id='app'>
<ejs-schedule height='550px' width='100%' :selectedDate='selectedDate'
:eventSettings='eventSettings'>
<e-views>
<e-view option='Day'></e-view>
<e-view option='Week' startHour='07:00' endHour='15:00'></e-view>
<e-view option='WorkWeek' startHour='10:00' endHour='18:00'></e-view>
<e-view option='Month' showWeekend=false></e-view>
<e-view option='Agenda'></e-view>
</e-views>
<e-resources>
<e-resource field="OwnerId" title="Owner" name="Owners"
:dataSource="ownerDataSource" textField="OwnerText"
idField="Id" colorField="OwnerColor">
</e-resource>
</e-resources>
</ejs-schedule>
</div>
</template>
<script lang="ts">
import { ScheduleComponent, ViewsDirective, ViewDirective,
ResourcesDirective, ResourceDirective, Day, Week, WorkWeek, Month, Agenda,
DragAndDrop, Resize, EventSettingsModel } from "@syncfusion/ej2-vue-
schedule";
export default {

```



```

name: "App",
// Declaring component and its directives
components: {
  'ejs-schedule': ScheduleComponent,
  'e-views': ViewsDirective,
  'e-view': ViewDirective,
  'e-resources': ResourcesDirective,
  'e-resource': ResourceDirective
},
// Bound properties declaration
data() {
  return {
    selectedDate: new Date(2021, 7, 12) as Date,
    allowMultiple: true as Boolean,
    ownerDataSource: [
      { OwnerText: 'Nancy', Id: 1, OwnerColor: '#ffaa00' },
      { OwnerText: 'Steven', Id: 2, OwnerColor: '#f8a398' },
      { OwnerText: 'Michael', Id: 3, OwnerColor: '#7499e1' }] as Record<string,
any>[],
    eventSettings: {
      dataSource: [
        {
          Id: 1,
          Subject: 'Surgery - Andrew',
          EventType: 'Confirmed',
          StartTime: new Date(2021, 7, 10, 9, 0),
          EndTime: new Date(2021, 7, 10, 10, 0),
          OwnerId: 2
        },
        {
          Id: 2,
          Subject: 'Consulting - John',
          EventType: 'Confirmed',
          StartTime: new Date(2021, 7, 11, 10, 0),
          EndTime: new Date(2021, 7, 11, 11, 30),
          OwnerId: 3
        },
        {
          Id: 3,
          Subject: 'Therapy - Robert',
          EventType: 'Requested',
          StartTime: new Date(2021, 7, 12, 11, 30),
          EndTime: new Date(2021, 7, 12, 12, 30),
          OwnerId: 1
        }
      ]
    } as EventSettingsModel,
  };
},
provide: {
  schedule: [Day, Week, WorkWeek, Month, Agenda, DragAndDrop, Resize]
};
</script>
<style>
@import '../node_modules/@syncfusion/ej2-base/styles/material.css';
@import '../node_modules/@syncfusion/ej2-buttons/styles/material.css';

```

```
@import '../node_modules/@syncfusion/ej2-calendars/styles/material.css';
@import '../node_modules/@syncfusion/ej2-dropdowns/styles/material.css';
@import '../node_modules/@syncfusion/ej2-inputs/styles/material.css';
@import '../node_modules/@syncfusion/ej2-navigations/styles/material.css';
@import '../node_modules/@syncfusion/ej2-popups/styles/material.css';
@import '../node_modules/@syncfusion/ej2-vue-schedule/styles/material.css';
</style>
```

### Run the project

To run the project, use the following command:

```
`bash
```

```
npm run dev
```

```
,
```

or

```
`bash
```

```
yarn run dev
```

```
,
```

The output will appear as follows:

<	>	August 08 - 14, 2021	TODAY	DAY	WEEK	WORK WEEK	MONTH	:
	Sun 8	Mon 9	Tue 10	Wed 11	Thu 12	Fri 13	Sat 14	
9:00 AM			Surgery - Andrew 9:00 AM - ...					
10:00 AM				Consultin g - John 10:00 AM...				
11:00 AM								

See also

- [Getting Started with Vue UI Components using Composition API and JavaScript](#)
- [Getting Started with Vue UI Components using Composition API and TypeScript](#)
- [Getting Started with Vue UI Components using Options API and JavaScript](#)

### Getting Started with Syncfusion Vue UI Components in Vue 3

This section explains how to use Syncfusion Vue components in Vue 3 application. To get started with Vue 2 application, refer to the [getting started with Vue 2](#) section.

#### Prerequisites

#### [System requirements for Syncfusion Vue UI components](#)

### Create the Vue 3 application

The best way to create a Vue 3 application is to use the [vue create](#) command.

```
`bash
npm install -g @vue/cli
vue create quickstart
cd quickstart
npm run serve
`
```

Initiating a new project prompts us to choose the type of project to be used for the current application. Select the option **Default ([Vue 3] babel, eslint)** from the menu.

```
E:\samples>vue create quickstart

Vue CLI v5.0.8
? Please pick a preset:
> Default ([Vue 3] babel, eslint)
  Default ([Vue 2] babel, eslint)
  Manually select features
```

### Add Syncfusion packages

Once the Vue 3 application is created, install the required Syncfusion Vue component package in the application. All the available Syncfusion Vue packages are published in the [npmjs.com](#) registry. Choose the component to be installed. In this article, the Grid component is used as an example.

Check out the [installation and upgrade](#) section to learn about the different ways of installing the packages. Here, the Grid component package is installed using the following **npm** command.

```
`bash
npm install @syncfusion/ej2-vue-grids --save
`
```

### Import the Syncfusion CSS styles

After installing the Syncfusion component packages in the application, add the required theme based on the components used.

Check out the [themes](#) section to know more about built-in themes and different ways (npm packages, CDN and CRG) to refer the themes in the Vue application.

Here the themes are referred through the installed npm packages which contains the built-in themes of Syncfusion Vue component. Let's import the **Material** theme for the Grid component and its dependencies to the **<style>** section of the **App.vue** file as follows.

#### ~/SRC/APP.VUE

```
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-buttons/styles/material.css";
```

```
@import "../node_modules/@syncfusion/ej2-calendars/styles/material.css";
@import "../node_modules/@syncfusion/ej2-dropdowns/styles/material.css";
@import "../node_modules/@syncfusion/ej2-inputs/styles/material.css";
@import "../node_modules/@syncfusion/ej2-navigations/styles/material.css";
@import "../node_modules/@syncfusion/ej2-popups/styles/material.css";
@import "../node_modules/@syncfusion/ej2-splitbuttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
</style>
```

Grid components use other Syncfusion components as well, so CSS references for the dependent component must be added in order to use all grid functionalities. Use this same order to display the Syncfusion Grid component's predefined appearance.

#### *Register the Syncfusion Vue component*

Import the Grid component along with the required child directives from the installed packages into the `<script>` section of the `src/App.vue` file. Register the Grid component along with the required child directives using following code.

#### ~/SRC/APP.VUE

```
import { GridComponent, ColumnsDirective, ColumnDirective } from
'@syncfusion/ej2-vue-grids';
// Component registration
export default {
  name: "App",
  components: {
    'ejs-grid' : GridComponent,
    'e-columns' : ColumnsDirective,
    'e-column' : ColumnDirective
  }
}
```

Now, the Grid and column directives are registered to use it in this application.

#### *Add Syncfusion Vue component to the application*

Add the Vue Grid to the `<template>` section of the `App.vue` file in the `src` directory. To display the Grid with records, add the Grid component and bind the [dataSource](#) to it. Here, the simple data is mapped to the `dataSource` property.

#### ~/SRC/APP.VUE

```
<template>
<ejs-grid :dataSource="data">
<e-columns>
<e-column field="OrderID" headerText="Order ID" textAlign="Right"
:isPrimaryKey="true" width="100"></e-column>
<e-column field="CustomerID" headerText="Customer ID" width="80"></e-
column>
<e-column field="ShipCountry" headerText="Ship Country" width="90"></e-
column>
</e-columns>
</ejs-grid>
</template>
<script>
```

```

import { GridComponent, ColumnsDirective, ColumnDirective} from
"@syncfusion/ej2-vue-grids";
export default {
  name: "App",
  // Declaring component and its directives
  components: {
    "ejs-grid": GridComponent,
    "e-columns": ColumnsDirective,
    "e-column": ColumnDirective,
  },
  // Bound properties declarations
  data() {
    return {
      data: [
        {
          OrderID: 10248,
          CustomerID: "VINET",
          ShipCountry: "France",
        },
        {
          OrderID: 10249,
          CustomerID: "TOMSP",
          ShipCountry: "Germany",
        },
      ],
    };
  },
};
</script>
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-buttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-calendars/styles/material.css";
@import "../node_modules/@syncfusion/ej2-dropdowns/styles/material.css";
@import "../node_modules/@syncfusion/ej2-inputs/styles/material.css";
@import "../node_modules/@syncfusion/ej2-navigations/styles/material.css";
@import "../node_modules/@syncfusion/ej2-popups/styles/material.css";
@import "../node_modules/@syncfusion/ej2-splitbuttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
</style>

```

### Run the application

Run the application using the following command.

```
`bash
```

```
npm run serve
```

```
`
```

Web server will be initiated. Open the quick start app in the browser at port `localhost:8080`.

Order ID	Customer ID	Ship Country
10248	VINET	France
10249	TOMSP	Germany

Refer the following sample, [vue3-grid-getting-started](#).

### Migration from Vue 2 to Vue 3

#### Registering Vue component

It is required to register the component and any child directives used within the component separately in Vue 3. The difference in registering components in Vue 2 and Vue 3 can be found below.

- Component registration in Vue 2

#### ~/SRC/APP.VUE

```
import * as Vue from 'vue';
import { GridPlugin } from '@syncfusion/ej2-vue-grids';
// Registering component and directives as a single plugin.
Vue.use(ButtonPlugin);
```

- Component registration in Vue 3

#### ~/SRC/APP.VUE

```
import { GridComponent, ColumnsDirective, ColumnDirective } from
 '@syncfusion/ej2-vue-grids';
//Component registration
export default {
  name: "App",
  components: {
    'ejs-grid' : GridComponent,
    'e-columns' : ColumnsDirective,
    'e-column' : ColumnDirective
  }
}
```

In the above code, `ejs-grid` denotes the Grid component tag, `e-columns` and `e-column` denotes the child column directives tag which is used for Column definition declaration.

Registering the child directives is not needed if they are not used.

#### Template usage:

Before using the template in the Vue application, enable the [runtime compiler](#). Create the `Vue.config.js` file in the root folder if it does not exist and add the following code

```
`js
module.exports = {
  runtimeCompiler: true
```

```

}
`

```

Due to changes in the [Vue 3 API](#), the registration of templates in Vue 3 is different from Vue 2.

Vue 2	Vue 3
declare templates using <code>Vue.component</code> module.	Use the <code>createApp</code> method from Vue to declare templates.

In template declaration, the component name must match the property binding name. In the following example, the Grid column `template` property is assigned with the name `colTemplate`.

#### ~/SRC/APP.VUE

```

<template>
<ejs-grid ref='grid' :dataSource="data" height=310 >
<e-columns>
<e-column headerText='Employee Name' width='150' textAlign='Center'
:template='colTemplate'></e-column>
<e-column field='EmployeeID' width='125' textAlign='Right'></e-column>
</e-columns>
</ejs-grid>
<template>
<script>
import { GridComponent, ColumnsDirective, ColumnDirective} from
"@syncfusion/ej2-vue-grids";
import { createApp } from "vue";
const app = createApp();
// Template declaration
var colVue = app.component('colTemplate', {
data: () => ({}),
template: `<b>Name:{{data.EmployeeID}}</b>`;
export default {
data() {
return {
data: [
{
EmployeeID: 10248,
EmployeeName: "VINET"
},
{
EmployeeID: 10249,
EmployeeName: "TOMSP"
},
],
colTemplate: function() {
return { template: colVue };
}
};
}
};
</script>
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";

```

```
@import "../node_modules/@syncfusion/ej2-buttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-calendars/styles/material.css";
@import "../node_modules/@syncfusion/ej2-dropdowns/styles/material.css";
@import "../node_modules/@syncfusion/ej2-inputs/styles/material.css";
@import "../node_modules/@syncfusion/ej2-navigations/styles/material.css";
@import "../node_modules/@syncfusion/ej2-popups/styles/material.css";
@import "../node_modules/@syncfusion/ej2-splitbuttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
</style>
```

### Using Syncfusion Vue components inside template properties

To use other Syncfusion Vue components inside the templates, register the components in the template declaration also.

The following sample uses the Button component within the grid's template property. To use the Button component within the template, register the Button component in the template declaration.

#### ~/SRC/APP.VUE

```
<template>
<ejs-grid ref='grid' :dataSource="data">
<e-columns>
<e-column headerText='EmployeeName' width='150' textAlign='Center'
:template='colTemplate'></e-column>
<e-column field='EmployeeID' width='125' textAlign='Right'></e-column>
</e-columns>
</ejs-grid>
</template>
<script>
import { ButtonComponent } from "@syncfusion/ej2-vue-buttons";
import { GridComponent, ColumnsDirective, ColumnDirective } from
"@syncfusion/ej2-vue-grids";
import { createApp } from "vue";
const app = createApp();
// Template declaration
var colVue = app.component('colTemplate', {
data: () => ({}),
template: `
<ejs-Button cssClass="e-primary">{{data.EmployeeName}}</ejs-Button>`,
// Declaring component which is used inside template property
components: {
"ejs-Button" : ButtonComponent
}
});
export default {
name: 'Vue3-App',
components: {
"ejs-grid": GridComponent,
"e-columns": ColumnsDirective,
"e-column": ColumnDirective,
},
data() {
return {
data: [
{
EmployeeID: 10248,
```



```

EmployeeName: "VINET"
},
{
EmployeeID: 10249,
EmployeeName: "TOMSP"
},
],
colTemplate: function() {
return { template: colVue };
}
};
}
};
</script>
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-buttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-calendars/styles/material.css";
@import "../node_modules/@syncfusion/ej2-dropdowns/styles/material.css";
@import "../node_modules/@syncfusion/ej2-inputs/styles/material.css";
@import "../node_modules/@syncfusion/ej2-navigations/styles/material.css";
@import "../node_modules/@syncfusion/ej2-popups/styles/material.css";
@import "../node_modules/@syncfusion/ej2-splitbuttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
</style>

```

*See also*

- [Getting started with Vue 2 application](#)
- [Getting started with Vue component using direct script](#)

## Vue 2

### Getting Started with Syncfusion Vue UI Components and Vue CLI

This section explains how to use Syncfusion Vue components in Vue 2 application. To get started with Vue 3 application, refer to the [getting started with Vue 3](#) topic.

#### *Prerequisites*

#### [System requirements for Syncfusion Vue UI components](#)

#### *Create the Vue 2 application*

The best way to create a Vue 2 application is to use the [vue create](#) command.

```
`bash
```

```
npm install -g @vue/cli
```

```
vue create quickstart
```

```
cd quickstart
```

```
npm run serve
```

```
`
```

Initiating a new project prompts us to choose the type of project to be used for the current application. Select the option Default ([Vue 2] babel, eslint) from the menu.

```
E:\samples>vue create quickstart

Vue CLI v5.0.8
? Please pick a preset:
  Default ([Vue 3] babel, eslint)
> Default ([Vue 2] babel, eslint)
  Manually select features
```

#### *Add Syncfusion packages*

Once the Vue 2 application is created, install the required Syncfusion Vue component package in the application. All the available Syncfusion Vue packages are published in the [npmjs.com](https://www.npmjs.com) registry. Choose the component to be installed. In this article, the Grid component is used as an example.

Check out the [installation and upgrade](#) section to learn about the different ways of installing the packages. Here, the Grid component package is installed using the following `npm` command.

```
`bash
```

```
npm install @syncfusion/ej2-vue-grids --save
```

#### *Import the Syncfusion CSS styles*

After installing the Syncfusion component packages in the application, add the required theme based on the components used.

Check out the [themes](#) section to know more about built-in themes and different ways (npm packages, CDN and CRG) to refer the themes in the Vue application.

Here the themes are referred through the installed npm packages which contains the built-in themes of Syncfusion Vue component. Let's import the Material theme for the Grid component and its dependencies to the `<style>` section of the `App.vue` file as follows.

#### ~/SRC/APP.VUE

```
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-buttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-calendars/styles/material.css";
@import "../node_modules/@syncfusion/ej2-dropdowns/styles/material.css";
@import "../node_modules/@syncfusion/ej2-inputs/styles/material.css";
@import "../node_modules/@syncfusion/ej2-navigations/styles/material.css";
@import "../node_modules/@syncfusion/ej2-popups/styles/material.css";
@import "../node_modules/@syncfusion/ej2-splitbuttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
</style>
```

Grid components use other Syncfusion components as well, so CSS references for the dependent component must be added in order to use all grid functionalities. Use this same order to display the Syncfusion Grid component's predefined appearance.

### Register the Syncfusion Vue component

Vue has two ways to register the Vue components in the Vue 2 application. Use one of the following ways to register the Syncfusion Vue components:

- [Vue.use\(\)](#) - It registers the Vue component and all its child directives globally.
- [Vue.component\(\)](#) - It registers the Vue component only. It will not register the child directives automatically. The child directives should be registered separately.

#### Using Vue.use()

Import the component plugin from the Vue package and register it using `Vue.use()` with the component plugin as its argument.

Refer to the following code snippet.

#### ~/SRC/APP.VUE

```
import { GridPlugin } from '@syncfusion/ej2-vue-grids';
Vue.use(GridPlugin);
```

#### Using Vue.component()

Import the component and component plugin from the Vue package and register them using `Vue.component()` with the name of the component from the component plugin and the Vue component as its arguments.

Refer to the following the code snippet.

#### ~/SRC/APP.VUE

```
import { GridPlugin, GridComponent, ColumnsDirective, ColumnsPlugin,
ColumnDirective, ColumnPlugin } from '@syncfusion/ej2-vue-grids';
Vue.component(GridPlugin.name, GridComponent);
Vue.component(ColumnsPlugin.name, ColumnsDirective);
Vue.component(ColumnPlugin.name, ColumnDirective);
```

### Add Syncfusion Vue component to the application

Add the Vue Grid to the `<template>` section of the `App.vue` file in the `src` directory. To display the Grid with records, add the Grid component and bind the [dataSource](#) to it. Here, the simple data is mapped to the `dataSource` property.

#### ~/SRC/APP.VUE

```
<template>
<div id="app">
<ejs-grid :dataSource="data">
<e-columns>
<e-column field="OrderID" headerText="Order ID" textAlign="Right"
:isPrimaryKey="true" width="100"></e-column>
<e-column field="CustomerID" headerText="Customer ID" width="80"></e-
column>
<e-column field="ShipCountry" headerText="Ship Country" width="90"></e-
column>
</e-columns>
</ejs-grid>
</div>
```

```

</template>
<script>
import Vue from 'vue';
import { GridPlugin } from '@syncfusion/ej2-vue-grids';
Vue.use(GridPlugin);
export default {
  name: 'app',
  data () {
    return {
      data: [
        {
          OrderID: 10248,
          CustomerID: "VINET",
          ShipCountry: "France",
        },
        {
          OrderID: 10249,
          CustomerID: "TOMSP",
          ShipCountry: "Germany",
        },
      ],
    }
  }
}
</script>

```

### Run the application

Run the application using the following command.

```
`bash
```

```
npm run serve
```

```
,
```

The output will appear as follows:

### ~/SRC/APP.VUE

```

<template>
  <div id="app">
    <ejs-grid :dataSource="data">
      <e-columns>
        <e-column field="OrderID" headerText="Order ID" textAlign="Right"
:isPrimaryKey="true" width="100"></e-column>
        <e-column field="CustomerID" headerText="Customer ID" width="80"></e-
column>
        <e-column field="ShipCountry" headerText="Ship Country"
width="90"></e-column>
      </e-columns>
    </ejs-grid>
  </div>
</template>
<script>
import Vue from 'vue';
import { GridPlugin } from '@syncfusion/ej2-vue-grids';
Vue.use(GridPlugin);

```

```

export default {
  data () {
    return {
      data: [
        {
          OrderID: 10248,
          CustomerID: "VINET",
          ShipCountry: "France",
        },
        {
          OrderID: 10249,
          CustomerID: "TOMSP",
          ShipCountry: "Germany",
        },
      ],
    }
  }
}
</script>
<style>
  @import "../node_modules/@syncfusion/ej2-base/styles/material.css";
  @import "../node_modules/@syncfusion/ej2-buttons/styles/material.css";
  @import "../node_modules/@syncfusion/ej2-calendars/styles/material.css";
  @import "../node_modules/@syncfusion/ej2-dropdowns/styles/material.css";
  @import "../node_modules/@syncfusion/ej2-inputs/styles/material.css";
  @import "../node_modules/@syncfusion/ej2-navigations/styles/material.css";
  @import "../node_modules/@syncfusion/ej2-popups/styles/material.css";
  @import "../node_modules/@syncfusion/ej2-splitbuttons/styles/material.css";
  @import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/common/default-cs1" %}

*See also*

- [Getting started with Vue 3 application](#)
- [Getting started with Vue component using direct script](#)

Getting Started with Syncfusion Vue UI Components and Vue CLI

You can use [Vue CLI](#) to setup your Vue applications.

To install Vue CLI, use the following command.

```
`bash
```

```
npm install -g @vue/cli
```

```
npm install -g @vue/cli-init
```

```
`
```

Start a new project using the following Vue CLI command.

```
`bash
```

```
vue init webpack-simple quickstart
```

```
cd quickstart
```

```
npm install
```

```
,
```

### *Adding Syncfusion packages*

All the available Syncfusion Vue packages are published in the [npmjs.com](https://www.npmjs.com) registry.

Choose the component to be installed. In this application, the button component is installed.

To install the button component, use the following command.

```
`bash
```

```
npm install @syncfusion/ej2-vue-grids --save
```

```
,
```

### *Registering Vue component*

The two ways to register the Vue component are:

- `Vue.use()`
- `Vue.component()`

#### *Using `Vue.use()`*

Import the component plugin from the EJ2 Vue package and register it using `Vue.use()` with component plugin as its argument.

Refer to the following code snippet.

#### **~/SRC/APP.VUE**

```
import { GridPlugin, Page } from "@syncfusion/ej2-vue-grids";  
Vue.use(GridPlugin);
```

Note: By registering the component plugin in Vue, all child directives can also be globally registered.

#### *Using `Vue.component()`*

Import the component and component plugin from EJ2 Vue package and register it using the `Vue.component()` with name of component from component plugin and the EJ2 Vue component as its arguments.

Refer to the following the code snippet.

#### **~/SRC/APP.VUE**

```
import { GridComponent, GridPlugin } from "@syncfusion/ej2-vue-grids";  
Vue.use(GridPlugin.name, GridComponent);
```

Note: By using the `Vue.component()`, only the EJ2 Vue component can be registered. The child directives should be registered separately.

*Creating Vue sample*

Add the EJ2 Vue button in the `<template>` section of the `App.vue` file in `src` directory using `<ejs-button>`. The content attribute of the button component is provided as name in the data option of the `<script>` section.

**~/SRC/APP.VUE**

```
<template>
<div id="app">
  <ejs-grid :dataSource="data" :allowPaging="true"
    :pageSettings="pageSettings">
    <e-columns>
      <e-column field="OrderID" headerText="Order ID" textAlign="Right"
        width="90"></e-column>
      <e-column field="CustomerID" headerText="Customer ID" width="120"></e-
        column>
      <e-column field="Freight" headerText="Freight" textAlign="Right" format="C2"
        width="90"></e-column>
    </e-columns>
  </ejs-grid>
</div>
</template>
<script>
import Vue from "vue";
import { GridPlugin, Page, GridComponent } from "@syncfusion/ej2-vue-grids";
Vue.use(GridPlugin);
export default {
  data() {
    return {
      data: [
        { OrderID: 10248, CustomerID: "VINET", Freight: 32.38 },
        { OrderID: 10249, CustomerID: "TOMSP", Freight: 11.61 },
        { OrderID: 10250, CustomerID: "HANAR", Freight: 65.83 },
        { OrderID: 10251, CustomerID: "VICTE", Freight: 41.34 },
        { OrderID: 10252, CustomerID: "SUPRD", Freight: 51.3 },
        { OrderID: 10253, CustomerID: "HANAR", Freight: 58.17 },
        { OrderID: 10254, CustomerID: "CHOPS", Freight: 22.98 }
      ],
      pageSettings: { pageSize: 5 }
    };
  },
  provide: {
    grid: [Page]
  }
};
</script>
```

*Adding SCSS reference*

Add the styles of Grid component to the `<style>` section of the `App.vue` file as follows.

**~/SRC/APP.VUE**

```
<style lang="scss">
// syncfusion styles
@import "../node_modules/@syncfusion/ej2-base/styles/material.scss";
@import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.scss";
```

```
</style>
```

### Adding `includePaths` option

While using scss files for style reference, you need to configure the `includePaths` in sass-loader options in the `webpack.config.js` like below:

#### ~/WEBPACK.CONFIG.JS

```
module: {
  rules: [
    ....
    ....
    {
      test: /\.vue$/,
      loader: 'vue-loader',
      options: {
        loaders: {
          // Since sass-loader (weirdly) has SCSS as its default parse mode, we map
          // the "scss" and "sass" values for the lang attribute to the right configs
          // here.
          'scss': [
            'vue-style-loader',
            'css-loader',
            {
              loader: "sass-loader",
              options: {
                includePaths: [
                  path.resolve(__dirname, "../node_modules/@syncfusion")
                ]
              }
            }
          ]
        }
      }
    }
    ....
    ....
  ]
}
```

### Running the application

Now, run the `npm run dev` command in the console to build your application and open it in the browser.

#### ~/SRC/APP.VUE

```
<template>
  <div id="app">
    <ejs-grid :dataSource="data" :allowPaging="true"
    :pageSettings='pageSettings'>
      <e-columns>
        <e-column field='OrderID' headerText='Order ID'
        textAlign='Right' width=90></e-column>
```



```

        <e-column field='CustomerID' headerText='Customer ID'
width=120></e-column>
        <e-column field='Freight' headerText='Freight' textAlign='Right'
format='C2' width=90></e-column>
        <e-column field='ShipCity' headerText='Ship City' width=150></e-
column>
    </e-columns>
</ejs-grid>
</div>
</template>
<script>
import Vue from "vue";
import { GridPlugin, Page } from "@syncfusion/ej2-vue-grids";
import { data } from "../datasource.js";
Vue.use(GridPlugin);
export default {
  data() {
    return {
      data: data,
      pageSettings: { pageSizes: true, pageSize: 9 }
    };
  },
  provide: {
    grid: [Page]
  }
}
</script>
<style lang="scss">
// syncfusion styles
@import "../node_modules/@syncfusion/ej2-base/styles/material.scss";
@import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.scss";
</style>

```

{% previewsample "page.domainurl/code-snippet/common/sass-cs1" %}

#### How To override styles

In Syncfusion Vue components, you can override control styles by replacing sass variable values like below:

#### ~/SRC/APP.VUE

```

<style lang="scss">
// SASS Variable override
$accent: black;
$primary: blue;
// syncfusion styles
@import "../node_modules/@syncfusion/ej2-base/styles/material.scss";
@import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.scss";
</style>

```

## Getting Started with Vue UI Components with the Nuxt Framework

This article provides a step-by-step guide for setting up a [Nuxt](#) project and integrating the Syncfusion Vue components using the [Composition API](#).

Nuxt.js is a powerful framework for building universal Vue.js applications. It is built on top of Vue.js and provides a higher-level structure and conventions to simplify the development of Vue applications.

### Prerequisites

#### [System requirements for Syncfusion Vue UI components](#)

#### Set up the Nuxt project

To initiate the creation of a new [Nuxt](#) project, use the following commands:

```
`bash
npx nuxi@latest init my-project
cd my-project
npm install
`
```

Now that **my-project** is ready to run with default settings, let's add Syncfusion components to the project.

#### Add Syncfusion Vue packages

Syncfusion Vue component packages are available at [npmjs.com](https://www.npmjs.com). To use Syncfusion Vue components in the project, install the corresponding npm package.

This article uses the [Vue Grid component](#) as an example. To use the Vue Grid component in the project, the **@syncfusion/ej2-vue-grids** package needs to be installed using the following command:

```
`bash
npm install @syncfusion/ej2-vue-grids --save
`
```

#### Import Syncfusion CSS styles

You can import themes for the Syncfusion Vue component in various ways, such as using CSS or SASS styles from npm packages, CDN, CRG and [Theme Studio](#). Refer to the [themes topic](#) to learn more about built-in themes and different ways to refer to themes in a Vue project.

In this article, the **Material** theme is applied using CSS styles, which are available in installed packages. The necessary **Material** CSS styles for the Grid component and its dependents were imported into the **<style>** section of the **app.vue** file.

#### ~/APP.VUE

```
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-buttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-calendars/styles/material.css";
@import "../node_modules/@syncfusion/ej2-dropdowns/styles/material.css";
@import "../node_modules/@syncfusion/ej2-inputs/styles/material.css";
@import "../node_modules/@syncfusion/ej2-navigations/styles/material.css";
@import "../node_modules/@syncfusion/ej2-popups/styles/material.css";
@import "../node_modules/@syncfusion/ej2-splitbuttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
</style>
```

The order of importing CSS styles should be in line with their dependency graph.

### Add the Syncfusion Vue component

Follow the below steps to add the Vue Grid component:

1\ First, add the `setup` attribute to the `script` tag to indicate that Vue will be using the `Composition API`. And import the Grid component in the `script` section of the `app.vue` file.

#### ~/APP.VUE

```
<script setup>
import { GridComponent as EjsGrid, ColumnsDirective as EColumns,
ColumnDirective as EColumn } from '@syncfusion/ej2-vue-grids';
</script>
```

2\ In the `template` section, define the Grid component with the `dataSource` property and column definitions.

#### ~/APP.VUE

```
<template>
<ejs-grid :dataSource='data'>
<e-columns>
<e-column field='OrderID' width='100' textAlign="Right"></e-column>
<e-column field='CustomerID' width='100'></e-column>
<e-column field='EmployeeID' width='100' textAlign="Right"></e-column>
<e-column field='Freight' width='100' format="C2" textAlign="Right"></e-
column>
<e-column field='ShipCountry' width='100'></e-column>
</e-columns>
</ejs-grid>
</template>
```

3\ Declare the values for the `dataSource` property in the `script` section.

#### ~/APP.VUE

```
<script setup>
const data = [
{
OrderID: 10248, CustomerID: 'VINET', EmployeeID: 5, ShipCountry: 'France',
Freight: 32.38
},
{
OrderID: 10249, CustomerID: 'TOMSP', EmployeeID: 6, ShipCountry: 'Germany',
Freight: 11.61
},
{
OrderID: 10250, CustomerID: 'HANAR', EmployeeID: 4, ShipCountry: 'Brazil',
Freight: 65.83
}
];
</script>
```

Here is the summarized code for above steps in the `app.vue` file:

~/APP.VUE

```

<template>
<ejs-grid :dataSource='data'>
<e-columns>
<e-column field='OrderID' width='100' textAlign="Right"></e-column>
<e-column field='CustomerID' width='100'></e-column>
<e-column field='EmployeeID' width='100' textAlign="Right"></e-column>
<e-column field='Freight' width='100' format="C2" textAlign="Right"></e-
column>
<e-column field='ShipCountry' width='100'></e-column>
</e-columns>
</ejs-grid>
</template>
<script setup>
import { GridComponent as EjsGrid, ColumnsDirective as EColumns,
ColumnDirective as EColumn } from '@syncfusion/ej2-vue-grids';
const data = [
{
OrderID: 10248, CustomerID: 'VINET', EmployeeID: 5, ShipCountry: 'France',
Freight: 32.38
},
{
OrderID: 10249, CustomerID: 'TOMSP', EmployeeID: 6, ShipCountry: 'Germany',
Freight: 11.61
},
{
OrderID: 10250, CustomerID: 'HANAR', EmployeeID: 4, ShipCountry: 'Brazil',
Freight: 65.83
}
];
</script>
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-buttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-calendars/styles/material.css";
@import "../node_modules/@syncfusion/ej2-dropdowns/styles/material.css";
@import "../node_modules/@syncfusion/ej2-inputs/styles/material.css";
@import "../node_modules/@syncfusion/ej2-navigations/styles/material.css";
@import "../node_modules/@syncfusion/ej2-popups/styles/material.css";
@import "../node_modules/@syncfusion/ej2-splitbuttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
</style>

```

## Run the project

To run the project, use the following command:

```
`bash
```

```
npm run dev
```

```
`
```

The output will appear as follows:

Order ID	Customer ID	Ship Country
10248	VINET	France
10249	TOMSP	Germany

## Getting Started with Vue UI Components with Vite and PNPM

This article provides a step-by-step guide for setting up a [Vite](#) project with a JavaScript environment and integrating the Syncfusion Vue components using [PNPM](#)

**PNPM** is an alternative package manager for Node.js that shares dependencies between packages and provides hard links to save disc space usage and installation times.

### Prerequisites

#### [System requirements for Syncfusion Vue UI components](#)

#### Set up the Vite project using PNPM

To initiate the creation of a new [Vite](#) project with [PNPM](#), use the below commands:

```
`bash
npm install -g pnpm
pnpm create vite@latest
`
```

Using the above command will lead you to set up additional configurations for the project as below:

1\ Define the project name: We can specify the name of the project directly. Let's specify the name of the project as **my-project** for this article.

```
`bash
? Project name: » my-project
`
```

2\ Select **Vue** as the framework. It will create a Vue 3 project.

```
`bash
? Select a framework: » - Use arrow-keys. Return to submit.
Vanilla
Vue
React
Preact
Lit
Svelte
Others
`
```

3\ Choose **JavaScript** as framework variant to build this Vite project using JavaScript and Vue.

```
`bash
```

? Select a variant: » - Use arrow-keys. Return to submit.

JavaScript

TypeScript

Customize with create-vue ↗

Nuxt ↗

`

4\ Upon completing the aforementioned steps to create **my-project**, run the following command to install its dependencies:

```
`bash
```

```
cd my-project
```

```
pnpm install
```

```
`
```

Now that **my-project** is ready to run with default settings, let's add Syncfusion components to the project.

#### Add Syncfusion Vue packages

Syncfusion Vue component packages are available at [npmjs.com](https://www.npmjs.com). To use Syncfusion Vue components in the project, install the corresponding npm package.

This article uses the [Vue Grid component](#) as an example. To use the Vue Grid component in the project, the **@syncfusion/ej2-vue-grids** package needs to be installed using the following command:

```
`bash
```

```
pnpm install @syncfusion/ej2-vue-grids --save
```

```
`
```

#### Import Syncfusion CSS styles

You can import themes for the Syncfusion Vue component in various ways, such as using CSS or SASS styles from npm packages, CDN, CRG and [Theme Studio](#). Refer to [themes topic](#) to learn more about built-in themes and different ways to refer to themes in a Vue project.

In this article, the **Material** theme is applied using CSS styles, which are available in installed packages. The necessary Material CSS styles for the Grid component and its dependents were imported into the **<style>** section of the **src/App.vue** file.

The dependency packages for Syncfusion in a PNPM vite project are situated within the **.pnpm/node\_modules** directory.

#### ~/SRC/APP.VUE

```
<style>
@import "../node_modules/.pnpm/node_modules/@syncfusion/ej2-
base/styles/material.css";
```

```
@import "../node_modules/.pnpm/node_modules/@syncfusion/ej2-
buttons/styles/material.css";
@import "../node_modules/.pnpm/node_modules/@syncfusion/ej2-
calendars/styles/material.css";
@import "../node_modules/.pnpm/node_modules/@syncfusion/ej2-
dropdowns/styles/material.css";
@import "../node_modules/.pnpm/node_modules/@syncfusion/ej2-
inputs/styles/material.css";
@import "../node_modules/.pnpm/node_modules/@syncfusion/ej2-
navigations/styles/material.css";
@import "../node_modules/.pnpm/node_modules/@syncfusion/ej2-
popups/styles/material.css";
@import "../node_modules/.pnpm/node_modules/@syncfusion/ej2-
splitbuttons/styles/material.css";
@import "../node_modules/.pnpm/node_modules/@syncfusion/ej2-
grids/styles/material.css";
</style>
```

The order of importing CSS styles should be in line with its dependency graph.

#### Add Syncfusion Vue component

Follow the below steps to add the Vue Grid component using the [Composition API](#):

1\, First, add the `setup` attribute to the `script` tag to indicate that Vue will be using the `Composition API`. And import the Grid component in the `script` section of the `src/App.vue` file.

#### ~/SRC/APP.VUE

```
<script setup>
import { GridComponent as EjsGrid, ColumnsDirective as EColumns,
ColumnDirective as EColumn } from '@syncfusion/ej2-vue-grids';
</script>
```

2\, In the `template` section, define the Grid component with the [dataSource](#) property and column definitions.

#### ~/SRC/APP.VUE

```
<template>
<ejs-grid :dataSource='data'>
<e-columns>
<e-column field='OrderID' width='100' textAlign="Right"></e-column>
<e-column field='CustomerID' width='100'></e-column>
<e-column field='EmployeeID' width='100' textAlign="Right"></e-column>
<e-column field='Freight' width='100' format="C2" textAlign="Right"></e-
column>
<e-column field='ShipCountry' width='100'></e-column>
</e-columns>
</ejs-grid>
</template>
```

3\, Declare the values for the `dataSource` property in the `script` section.

#### ~/SRC/APP.VUE

```
<script setup>
const data = [
  { OrderID: 10248, CustomerID: 'VINET', EmployeeID: 5, ShipCountry: 'France',
    Freight: 32.38 },
  { OrderID: 10249, CustomerID: 'TOMSP', EmployeeID: 6, ShipCountry:
    'Germany', Freight: 11.61 },
  { OrderID: 10250, CustomerID: 'HANAR', EmployeeID: 4, ShipCountry: 'Brazil',
    Freight: 65.83 }
];
</script>
```

Here is the summarized code for the above steps in the **src/App.vue** file:

### ~/SRC/APP.VUE

```
<template>
<ejs-grid :dataSource='data'>
<e-columns>
<e-column field='OrderID' width='100' textAlign="Right"></e-column>
<e-column field='CustomerID' width='100'></e-column>
<e-column field='EmployeeID' width='100' textAlign="Right"></e-column>
<e-column field='Freight' width='100' format="C2" textAlign="Right"></e-
column>
<e-column field='ShipCountry' width='100'></e-column>
</e-columns>
</ejs-grid>
</template>
<script setup>
// Import component and its directives
import { GridComponent as EjsGrid, ColumnsDirective as EColumns,
ColumnDirective as EColumn } from '@syncfusion/ej2-vue-grids';
const data = [
  { OrderID: 10248, CustomerID: 'VINET', EmployeeID: 5, ShipCountry: 'France',
    Freight: 32.38 },
  { OrderID: 10249, CustomerID: 'TOMSP', EmployeeID: 6, ShipCountry:
    'Germany', Freight: 11.61 },
  { OrderID: 10250, CustomerID: 'HANAR', EmployeeID: 4, ShipCountry: 'Brazil',
    Freight: 65.83 }
];
</script>
<style>
@import "../node_modules/.pnpm/node_modules/@syncfusion/ej2-
base/styles/material.css";
@import "../node_modules/.pnpm/node_modules/@syncfusion/ej2-
buttons/styles/material.css";
@import "../node_modules/.pnpm/node_modules/@syncfusion/ej2-
calendars/styles/material.css";
@import "../node_modules/.pnpm/node_modules/@syncfusion/ej2-
dropdowns/styles/material.css";
@import "../node_modules/.pnpm/node_modules/@syncfusion/ej2-
inputs/styles/material.css";
@import "../node_modules/.pnpm/node_modules/@syncfusion/ej2-
navigations/styles/material.css";
@import "../node_modules/.pnpm/node_modules/@syncfusion/ej2-
popups/styles/material.css";
```



```
@import "../node_modules/.pnpm/node_modules/@syncfusion/ej2-splitbuttons/styles/material.css";
@import "../node_modules/.pnpm/node_modules/@syncfusion/ej2-grids/styles/material.css";
</style>
```

### Run the project

To run the project, use the following command:

```
`bash
pnpm dev
`
```

The output will appear as follows:

![vue3-js-composition](../appearance/images/vue3-js-composition.png)

See also

- [Getting Started with Vue UI Components using Composition API and JavaScript](#)
- [Getting Started with Vue UI Components using Composition API and TypeScript](#)
- [Getting Started with Vue UI Components using Options API and JavaScript](#)
- [Getting Started with Vue UI Components using Options API and TypeScript](#)

### Getting started with testing Vue UI components in the Vitest project

This article provides a step-by-step guide for setting up a [Vitest](#) project, integrating Syncfusion Vue components, and perform comprehensive testing of the components.

**Vitest** is a blazing fast unit test framework powered by [Vite](#) that makes it easy to write and run tests for your Vue.js components. It is designed to be fast, easy to use, and compatible with Jest.

#### Prerequisites

[System requirements for Syncfusion Vue UI components](#)

#### Set up the Vitest project

To initiate the creation of a new **Vitest** project, use the following [create vue](#) command.

```
`bash
npm create vue@latest
`
```

Using the above commands will lead you to set up additional configurations for the project:

1\ Define the project name: We can specify the name of the project directly. Let's specify the name of the project as **my-project** for this article.

```
`bash
Vue.js - The Progressive JavaScript Framework
√ Project name: ... my-project
`
```

2\ To enable unit testing (Vitest) and configure the project, set the Vitest configuration to Yes.

```
`bash
```

```
√ Add TypeScript? ... No
```

```
√ Add JSX Support? ... No
```

```
√ Add Vue Router for Single Page Application development? ... No
```

```
√ Add Pinia for state management? ... No
```

```
√ Add Vitest for Unit Testing? ... Yes
```

```
√ Add an End-to-End Testing Solution? » No
```

```
√ Add ESLint for code quality? ... No
```

```
`
```

3\ Upon completing the aforementioned steps to create my-project, run the following command to install its dependencies:

```
`bash
```

```
cd my-project
```

```
npm install
```

```
`
```

4\ The default setup of Vitest utilizes JSDOM, which may not fully support all the APIs available in the window object. However, Syncfusion Vue components rely on certain APIs of the window object internally. Therefore, in order to ensure compatibility, it is necessary to configure Vitest with happy-dom. To install it, execute the following command:

```
`bash
```

```
npm i happy-dom --save-dev
```

```
`
```

5\ To add the happy-dom environment in the vitest.config.js file, replace the existing JSDOM value in the environment option with happy-dom. This will ensure that the happy-dom environment is used for your Vitest project.

#### ~/VITEST.CONFIG.JS

```
test: {  
  environment: 'happy-dom'  
}
```

Now that my-project is ready to run with default settings, let's add Syncfusion components to the project.

#### Add the Syncfusion packages

Syncfusion Vue component packages are available at [npmjs.com](https://www.npmjs.com). To use Syncfusion Vue components in the project, install the corresponding npm package.

This article uses the [Vue Grid component](#) as an example. To use the Vue Grid component in the project, the `@syncfusion/ej2-vue-grids` package needs to be installed using the following command:

```
`bash
```

```
npm install @syncfusion/ej2-vue-grids --save
```

```
,
```

### Add the Syncfusion Vue component

Follow the below steps to add the Vue Grid component:

1\ First, define the Grid component with the [dataSource](#) property and column definitions in the `src/components/HelloWorld.vue` file.

#### HELLOWORLD.VUE

```
<template>
<ejs-grid :dataSource="data">
<e-columns>
<e-column field='OrderID'></e-column>
<e-column field='CustomerID'></e-column>
<e-column field='EmployeeID'></e-column>
<e-column field='ShipCountry'></e-column>
<e-column field='Freight'></e-column>
</e-columns>
</ejs-grid>
</template>
<script>
import { GridComponent, ColumnsDirective, ColumnDirective } from
 '@syncfusion/ej2-vue-grids';
export default {
  components: {
    'ejs-grid': GridComponent,
    'e-columns': ColumnsDirective,
    'e-column': ColumnDirective
  },
  data() {
    return {
      data: [
        {
          OrderID: 10248, CustomerID: 'VINET', EmployeeID: 5, ShipCountry: 'France',
          Freight: 32.38
        },
        {
          OrderID: 10249, CustomerID: 'TOMSP', EmployeeID: 6, ShipCountry: 'Germany',
          Freight: 11.61
        },
        {
          OrderID: 10250, CustomerID: 'HANAR', EmployeeID: 4, ShipCountry: 'Brazil',
          Freight: 65.83
        }
      ]
    }
  }
};
</script>
```

2\ Next, add the unit testing cases for the component using the **Vitest** framework in the **src/components/tests/HelloWorld.spec.js** file.

### **HELLOWORLD.SPEC.JS**

```
import { describe, it, expect, vi } from 'vitest'
import { mount } from '@vue/test-utils'
import HelloWorld from '../HelloWorld.vue'
describe('EJSGrid', () => {
  it('Rows render correctly', async () => {
    window.crypto = vi.fn();
    window.crypto.getRandomValues = vi.fn();
    const wrapper = mount(HelloWorld);
    // Wait untill the component mount completely
    await new Promise((res) => setTimeout(res, 50))
    const rows = wrapper.findAll('.e-row');
    expect(rows.length).toBe(wrapper.vm.data.length);
    wrapper.unmount();
  });
  it('Columns render correctly', async () => {
    window.crypto = vi.fn();
    window.crypto.getRandomValues = vi.fn();
    const wrapper = mount(HelloWorld);
    // Wait untill the component mount completely
    await new Promise((res) => setTimeout(res, 50))
    const colHeader = wrapper.findAll('.e-headertext');
    for (let i = 0; i < Object.keys(wrapper.vm.data[0]).length; i++) {
      expect(colHeader[i].element.innerText).toBe(Object.keys(wrapper.vm.data[0])[i]);
    }
    wrapper.unmount();
  });
});
```

### Run the project

To run the project, use the following command:

```
`bash
```

```
npm run test:unit
```

```
`
```

The output will appear as follows:

```
`bash
```

```
DEV v0.31.4 D:/Testing/vitest
```

```
✓ src/components/tests/HelloWorld.spec.js (2)
```

```
Test Files 1 passed (1)
```

```
Tests 2 passed (2)
```

```
Start at 13:06:12
```

```
Duration 3.05s (transform 126ms, setup 0ms, collect 1.80s, tests 221ms, environment 602ms, prepare 130ms)
```

Getting started    Getting Started with Syncfusion Vue UI Components using direct scripts in a quickstart application

PASS Waiting for file changes...

press h to show help, press q to quit

,

<!-- markdownlint-disable MD024 -->

## Getting Started with Syncfusion Vue UI Components using direct scripts in a quickstart application

Vue provides native script support, allowing users to directly include the Vue.js library in an HTML file without the need for a build process or module bundler. This feature is useful for simpler projects or prototypes, enabling quick and easy implementation of Vue.js without setting up a complex build workflow.

Similarly, Syncfusion offers direct script support for its Vue components. Developers can seamlessly include Syncfusion Vue components in their HTML files and leverage them within their Vue.js applications. This allows for straightforward integration of Syncfusion Vue components without the need for additional build processes. Now, let's delve into the process of utilizing Syncfusion Vue components through direct script inclusion.

### Prerequisites

- Any IDE, such as [Visual Studio Code](#)

### Set up the Vue project

To demonstrate the usage of the **Grid** component through direct scripting, follow these steps:

1\. Begin by creating a folder named **quickstart** for the project. 2\. Inside the **quickstart** folder, create an HTML file named **index.html**.

Include the appropriate version of the Vue.js library in the **index.html** file based on whether to use [Vue 2](#) or [Vue 3](#) in the project. Then, create a new Vue instance with the required configuration options.

The [Vue class component](#) package is required before the 2023 Volume 1 (v21.1.35) release. So, add the [Vue class component](#) script to the head section of the **index.html** file for Vue 3 direct script.

#### **VUE 2 (~/INDEX.HTML)**

```
<div id="app">
<!-- Vue components goes here -->
</div>
<script
src="https://cdn.jsdelivr.net/npm/vue@2.7.14/dist/vue.min.js"></script>
<script>
new Vue({
el: '#app',
});
</script>
```

#### **VUE 3 (~/INDEX.HTML)**

```
<div id="app">
<!-- Vue components goes here -->
</div>
```

```
<script src="https://unpkg.com/vue@3/dist/vue.global.js"></script>
<script>
Vue.createApp({
  el: '#app',
}).mount('#app');
</script>
```

### Import Syncfusion CSS styles

You can import themes for the Syncfusion Vue component in various ways, such as using CSS from CDN, [CRG] and [Theme Studio](#). Refer to [themes topic](#) to know more about built-in themes and different ways to refer to themes in a Vue project.

In this article, **Material** theme is applied using CSS styles, which are available in CDN. Add the **Material** CSS styles to the head section of the **index.html** file.

#### ~/INDEX.HTML

```
<link rel="stylesheet"
href="https://cdn.syncfusion.com/ej2/21.2.3/material.css" rel="stylesheet"
type="text/css" />
```

### Import Syncfusion Vue scripts

To integrate Syncfusion components into your application, add the required Syncfusion Vue direct scripts to the head section of the **index.html** file to set up the Vue instance.

#### ~/INDEX.HTML

```
<script src="https://cdn.syncfusion.com/ej2/21.2.3/ej2-vue-es5/dist/ej2-
vue.min.js"></script>
```

### Add Syncfusion Vue component

1\ First, register the Grid component and its child directives in Vue.

#### VUE 2 (~/INDEX.HTML)

```
<script>
Vue.use(ejs.grids.GridPlugin);
</script>
```

#### VUE 3 (~/INDEX.HTML)

```
<script>
Vue.createApp({
  el: '#app',
  components: {
    'ejs-grid' : ej.grids.GridComponent,
    'e-columns' : ej.grids.ColumnsDirective,
    'e-column' : ej.grids.ColumnDirective
  }
}).mount('#app');
</script>
```

Getting started    Getting Started with Syncfusion Vue UI Components using direct scripts in a quickstart application

2\.. Add the component to the `section` of the **index.html** file. Bind the [data-source](#) property and inject the **Page** module. Follow the [Getting Started](#) documentation for further details.

While using Syncfusion Vue components in a direct script way, camel-cased property (`isPrimaryKey`) names need to be specified in the kebab-cased (`is-primary-key`) equivalents.

## VUE 2 (~/INDEX.HTML)

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
  <title>Syncfusion Vue (ES5) UI Components</title>
  <!-- Essential JS2 for Vue (All components Styles) -->
  <link href="https://cdn.syncfusion.com/ej2/21.2.3/material.css"
rel="stylesheet" type="text/css" />
  <!-- Vue library file-->
  <script src="https://cdn.jsdelivr.net/npm/vue@2.7.14/dist/vue.min.js"
type="text/javascript"></script>
  <script src="https://cdn.syncfusion.com/ej2/21.2.3/ej2-vue-es5/dist/ej2-
vue.min.js" type="text/javascript"></script>
</head>
<body>
  <h2>Syncfusion Vue 2 Grid Component</h2>
  <div id="app">
    <ejs-grid :data-source="data" :allow-paging="true" :page-
settings='pageSettings'>
      <e-columns>
        <e-column field="OrderID" header-text="Order ID" text-
align="Right" :is-primary-key="true"
          width="100"></e-column>
        <e-column field="CustomerID" header-text="Customer ID"
width="80"></e-column>
        <e-column field="Freight" header-text="Freight"
width="90"></e-column>
      </e-columns>
    </ejs-grid>
  </div>
  <script>
    Vue.use(ejs.grids.GridPlugin);
    new Vue({
      el: '#app',
      provide: {
        grid: [ejs.grids.Page]
      },
      data() {
        return {
          data: [
            { OrderID: 10248, CustomerID: 'VINET', Freight:
32.38 },
            { OrderID: 10249, CustomerID: 'TOMSP', Freight:
11.61 },
            { OrderID: 10250, CustomerID: 'HANAR', Freight:
65.83 },
            { OrderID: 10251, CustomerID: 'VICTE', Freight:
41.34 },
            { OrderID: 10252, CustomerID: 'SUPRD', Freight: 51.3
},
          ],
        }
      }
    })
  </script>
</body>
</html>
```

```

        { OrderID: 10253, CustomerID: 'HANAR', Freight:
58.17 },
        { OrderID: 10254, CustomerID: 'CHOPS', Freight:
22.98 },
        { OrderID: 10255, CustomerID: 'RICSU', Freight:
148.33 },
        { OrderID: 10256, CustomerID: 'WELLI', Freight:
13.97 }
    ],
    pageSettings: { pageSize: 5 }
  }
});
</script>
</body>
</html>

```

### VUE 3 (~/INDEX.HTML)

```

<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
  <title>Syncfusion Vue (ES5) UI Components</title>
  <!-- Essentail JS2 for Vue (All components Styles) -->
  <link href="https://cdn.syncfusion.com/ej2/21.2.3/material.css"
rel="stylesheet" type="text/css" />
  <!-- Vue library file-->
  <script src="https://unpkg.com/vue@3/dist/vue.global.js"></script>
  <script src="https://cdn.syncfusion.com/ej2/21.2.3/ej2-vue-es5/dist/ej2-
vue.min.js" type="text/javascript"></script>
</head>
<body>
  <h2>Syncfusion Vue 3 Grid Component</h2>
  <div id="app">
    <ejs-grid :data-source="data" :allow-paging="true" :page-
settings='pageSettings'>
      <e-columns>
        <e-column field="OrderID" header-text="Order ID" text-
align="Right" :is-primary-key="true"
width="100"></e-column>
        <e-column field="CustomerID" header-text="Customer ID"
width="80"></e-column>
        <e-column field="Freight" header-text="Freight"
width="90"></e-column>
      </e-columns>
    </ejs-grid>
  </div>
  <script>
    Vue.createApp({
      el: '#app',
      components: {
        'ejs-grid': ej.s.grids.GridComponent,
        'e-columns': ej.s.grids.ColumnsDirective,
        'e-column': ej.s.grids.ColumnDirective
      },
      provide: {

```



```

        grid: [ejs.grids.Page]
      },
      data() {
        return {
          data: [
            { OrderID: 10248, CustomerID: 'VINET', Freight:
32.38 },
            { OrderID: 10249, CustomerID: 'TOMSP', Freight:
11.61 },
            { OrderID: 10250, CustomerID: 'HANAR', Freight:
65.83 },
            { OrderID: 10251, CustomerID: 'VICTE', Freight:
41.34 },
            { OrderID: 10252, CustomerID: 'SUPRD', Freight: 51.3
},
            { OrderID: 10253, CustomerID: 'HANAR', Freight:
58.17 },
            { OrderID: 10254, CustomerID: 'CHOPS', Freight:
22.98 },
            { OrderID: 10255, CustomerID: 'RICSU', Freight:
148.33 },
            { OrderID: 10256, CustomerID: 'WELLI', Freight:
13.97 }
          ],
          pageSettings: { pageSize: 5 }
        }
      }
    }) .mount('#app');
  </script>
</body>
</html>

```

### Run the project

Run the **index.html** file in the web browser. The output will appear as follows:

```
{% previewsample "page.domainurl/code-snippet/common/getting-started-es5-vue3-cs1" %}
```

## Installation and Upgrade

<!-- markdownlint-disable MD024 -->

### Installation

#### Install by using npm CLI

Syncfusion Vue (Essential JS 2) packages are published in [npm](#). You can install the necessary packages from npm's install command. For example, vue grid package can be installed using following command.

,

```
npm install @syncfusion/ej2-vue-grids --save
```

,

### Install by using package.json

1. Add the Syncfusion Vue (Essential JS 2) package references in the `dependencies` of `~/package.json` file.

```
,  
{  
"dependencies": {  
"@syncfusion/ej2-vue-grids": "*",  
"@syncfusion/ej2-vue-charts": "*"  
}  
}  
,
```

The `*` indicates the latest version of npm package. Refer the [documentation](#) for more details about npm versioning.

2. Now, open the command prompt and run the `npm install` command line. This will install all npm dependencies in a single command line.

Refer the [documentation](#) for more details about npm package.json

### Download JavaScript – EJ2 Installer

The Syncfusion JavaScript - EJ2 installer can be downloaded from the Syncfusion website. You can either download the licensed installer or try our trial installer depending on your license.

- Trial Installer
- Licensed Installer

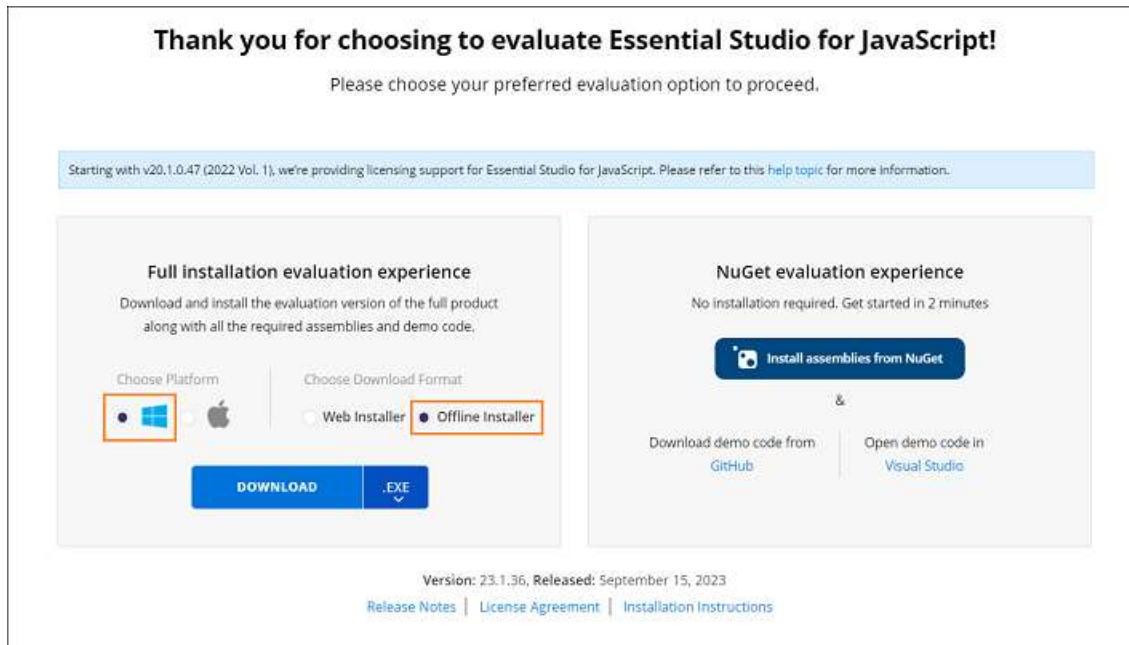
### Download the trial version

Our 30-day trial can be downloaded in two ways.

- Download free trial setup
- Start trials if using components [through npm](#)

### Download free trial setup

1. You can evaluate our 30-day free trial by visiting the [Download Free Trial](#) page and select the JavaScript platform.
2. After completing the required form or logging in with your registered Syncfusion account, you can download the JavaScript - EJ2 trial installer from the confirmation page. (See the screenshot below.)

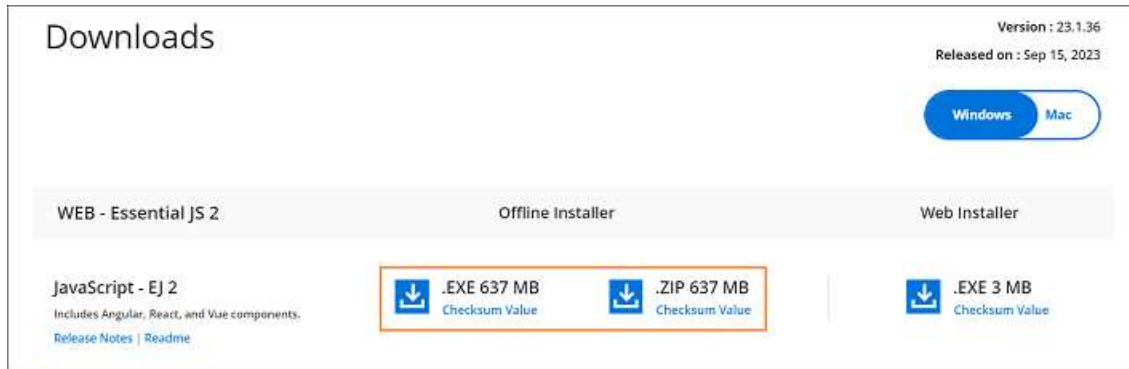


3. With a trial license, only the latest version's trial installer can be downloaded.
4. After downloading, the Syncfusion JavaScript - EJ2 trial installer can be unlocked using either the trial unlock key or the Syncfusion registered login credential. More information on generating an unlock key can be found in this article.
5. Before the trial expires, you can download the trial installer at any time from your registered account's Trials & Downloads page (See the screenshot below.)
6. Click the Download (element 1 in the screenshot below) button to get the Syncfusion Essential Studio JavaScript – EJ2 web installer.

## Trial Downloads and Unlock Keys



7. Click the More Download Options (element 2 in the above screenshot) button to get the Essential Studio JavaScript – EJ2 Offline trial installer which is available in EXE and ZIP format.



Downloads

Version : 23.1.36  
Released on : Sep 15, 2023

Windows Mac

WEB - Essential JS 2      Offline Installer      Web Installer

JavaScript - EJ 2  
Includes Angular, React, and Vue components.  
[Release Notes](#) | [Readme](#)

.EXE 637 MB  
Checksum Value

.ZIP 637 MB  
Checksum Value

.EXE 3 MB  
Checksum Value

### Start trials if using components through [npm](#)

You should initiate an evaluation if you have already obtained our components through [npm](#)

1. You can start your 30-day free trial for JavaScript – EJ2 from the [Start Trial](#) page from your account.



WEB

Blazor  
Includes 80+ Blazor components for ASP.NET Core application development.  
[TRIAL IN PROGRESS](#)

JavaScript  
Includes 80+ JavaScript controls for developing modern web applications. It also includes complete support for Angular, React, and Vue frameworks.  
[START TRIAL](#)

2. To access this page, you must sign up/log in with your Syncfusion account.
3. Begin your trial by selecting the JavaScript – EJ2 product.

Note: If you've already used the trial products and they haven't expired, you won't be able to start the trial for the same product again.

4. After you've started the trial, go to the [Trials & Downloads](#) page to get the latest version trial installer. You can generate the [unlock](#) key here at any time before the trial period expires. (See the screenshot below.)

### Trial Downloads and Unlock Keys



Starting with v20.1.0.47 (2022 Vol. 1), we're providing licensing support for Essential Studio for JavaScript(Essential JS 2). Please refer to this [help topic](#) for more information.

JavaScript (Essential JS 2) - [Redacted]

Trial Version : [Redacted]

[Release Notes](#) | [Get License Key](#) ⓘ | [Get Unlock Key](#) ⓘ | [Security Management Report](#)

[Download](#) 1

[More Download Options](#) 2

5. You can find your current active trial products on the [Trials & Downloads](#) page.

#### *Download the license version*

1. Syncfusion licensed products will be available in the [License & Downloads](#) page under your registered Syncfusion account.
2. You can view all the licenses (both active and expired) associated with your account.
3. Click the Download (element 1 in the screenshot below) button to download the respective product's installer.
4. The most recent version of the installer will be downloaded from this page.
5. To download older version installers, go to [Downloads Older Versions](#) (element 2 in the screenshot below).
6. You can download other platform/add-on installers by going to More Downloads Options (element 3 in the screenshot below).
7. For Windows OS, EXE and Zip formats are available for download. They are both Offline Installers.



You can also refer to the [Online installer](#) and [Offline installer](#) links for step-by-step installation guidelines.

### Installation using Web Installer

You can refer to the [Download](#) section to learn how to get the JavaScript – EJ2 trial or licensed installer.

#### Overview

For the Essential Studio JavaScript – EJ2 product, Syncfusion offers a Web Installer. This installer alleviates the burden of downloading a larger installer. You can simply download and run the online installer, which will be smaller in size and will download and install the Essential Studio products you have chosen. You can get the most recent version of Essential Studio Web Installer [here](#).

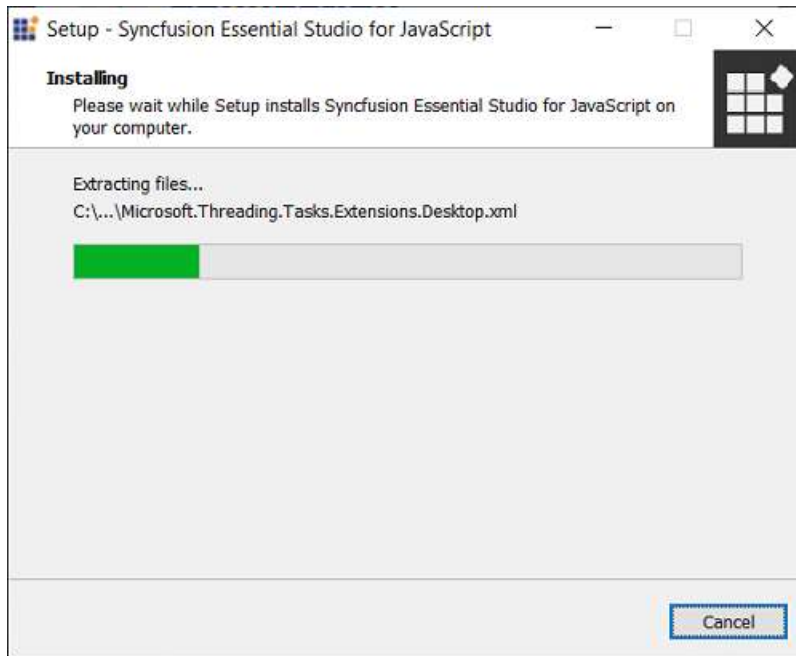
The frameworks listed below are supported in this installer.

- JavaScript
- Angular
- React
- Vue
- JavaScript (ES5)

### Installation

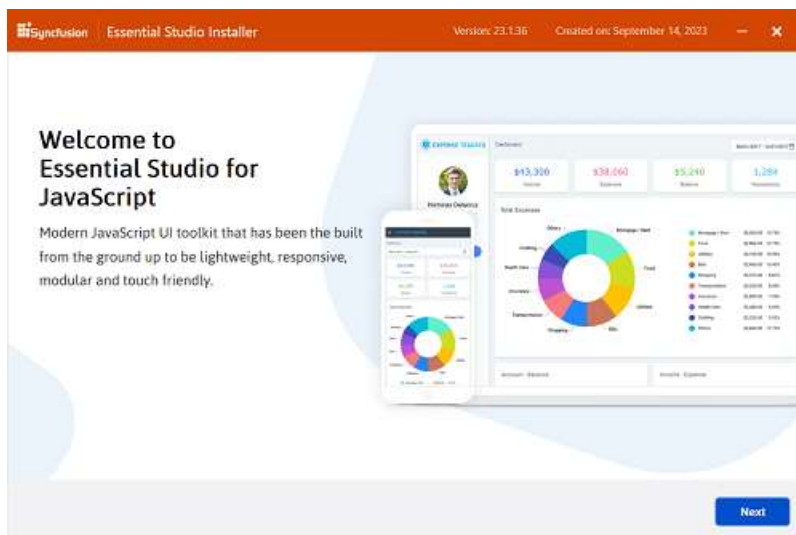
The steps below show how to install Essential Studio JavaScript – EJ2 Web Installer.

1. Open the Syncfusion Essential Studio JavaScript – EJ2 Web Installer file from downloaded location by double-clicking it. The Installer Wizard automatically opens and extracts the package.



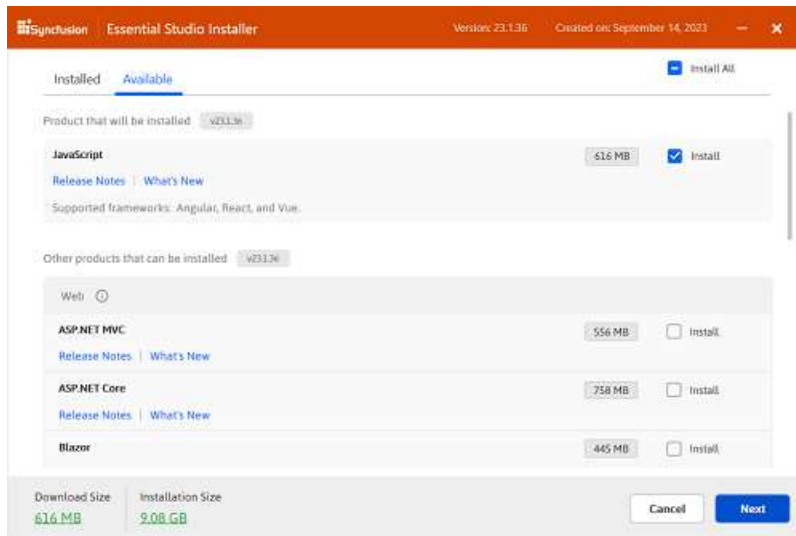
Note: The installer wizard extracts the syncfusionejs2webinstaller\_{version}.exe dialog, which displays the package's unzip operation.

2. The Syncfusion JavaScript - EJ2 Web Installer's welcome wizard will be displayed. Click the Next button.



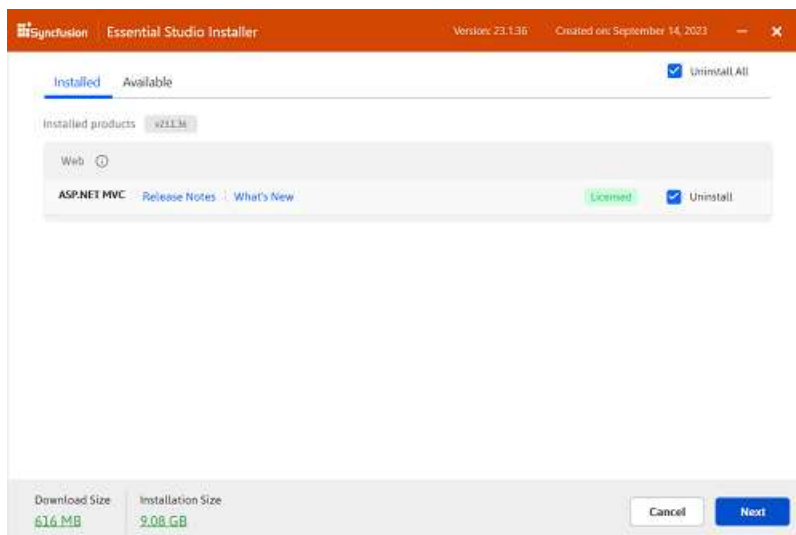
3. The Platform Selection Wizard will appear. From the **Available** tab, select the products to be installed. Select the Install All checkbox to **install all** products.

## Available



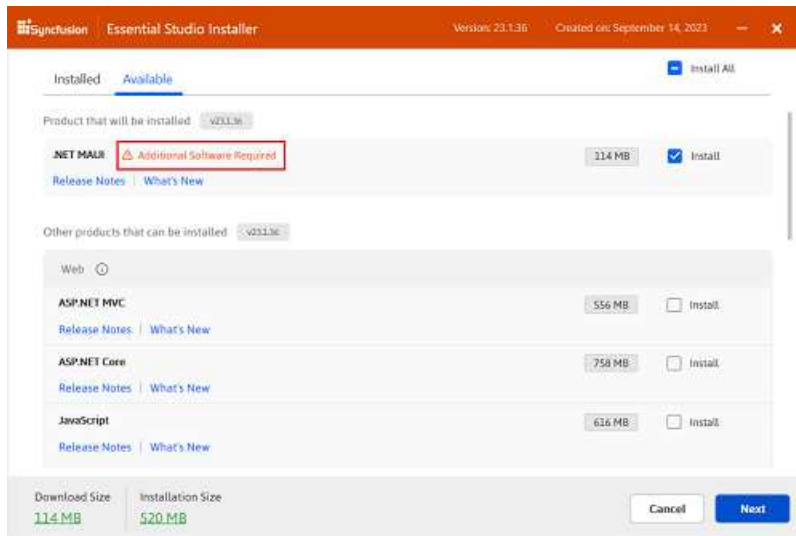
If you have multiple products installed in the same version, they will be listed under the **Installed** tab. You can also select which products to uninstall from the same version. Click the Next button.

### Installed

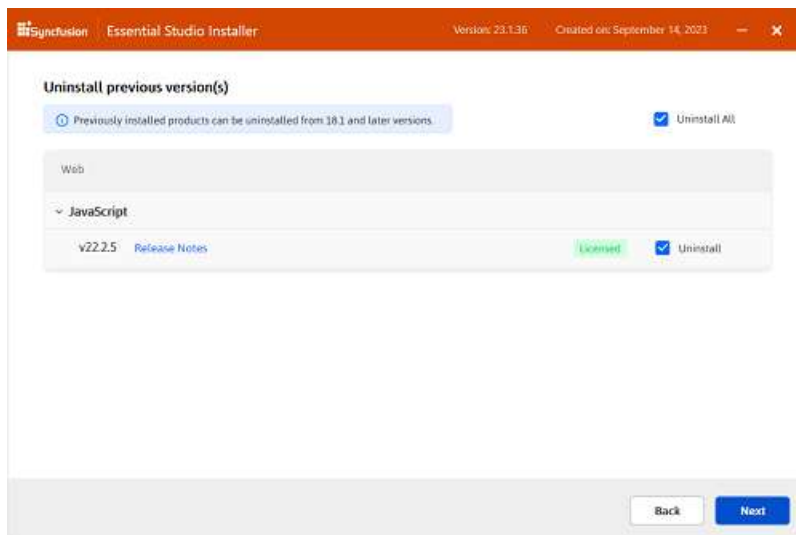


Important: If the required software for the selected product isn't already installed, the **Additional Software Required** alert will appear. You can, however, continue the installation and install the necessary software later.

### Required software



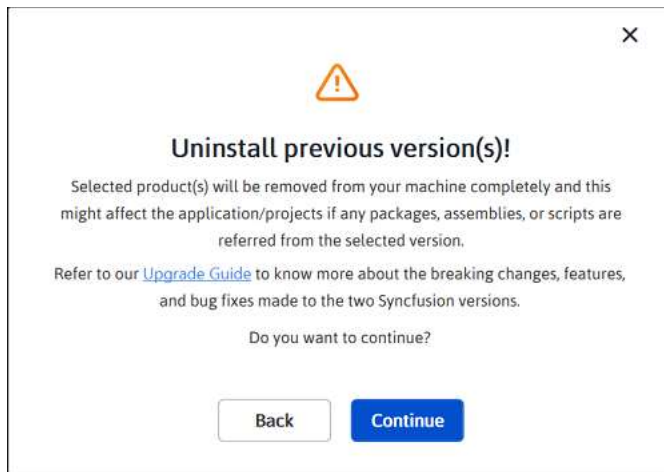
4. If previous version(s) for the selected products are installed, the Uninstall previous version wizard will be displayed. You can see the list of previously installed versions for the products you've chosen here. To remove all versions, check the **Uninstall All** checkbox. Click the Next button.



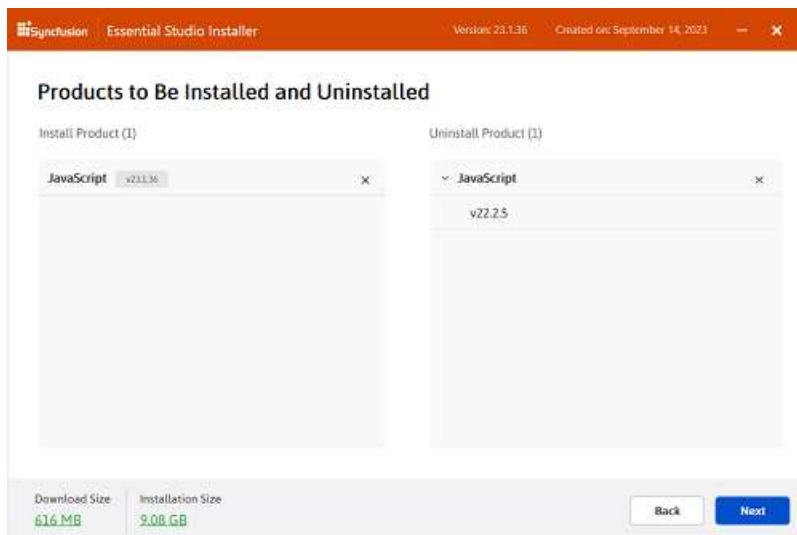
Note: From the 2021 Volume 1 release, Syncfusion has provided option to uninstall the previous versions from 18.1 while installing the new version.

5. Pop up screen will be displayed to get the confirmation to uninstall selected previous versions.



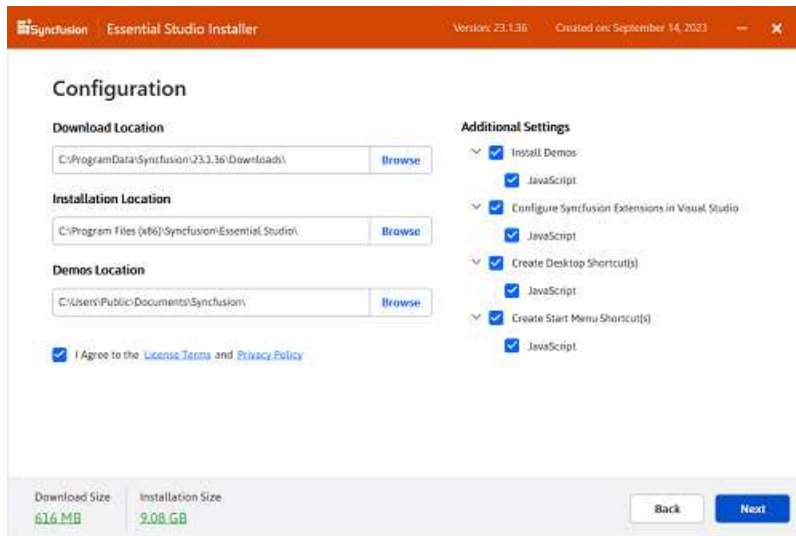


6. The Confirmation Wizard will appear with the list of products to be installed/uninstalled. You can view and modify the list of products that will be installed and uninstalled from this page.



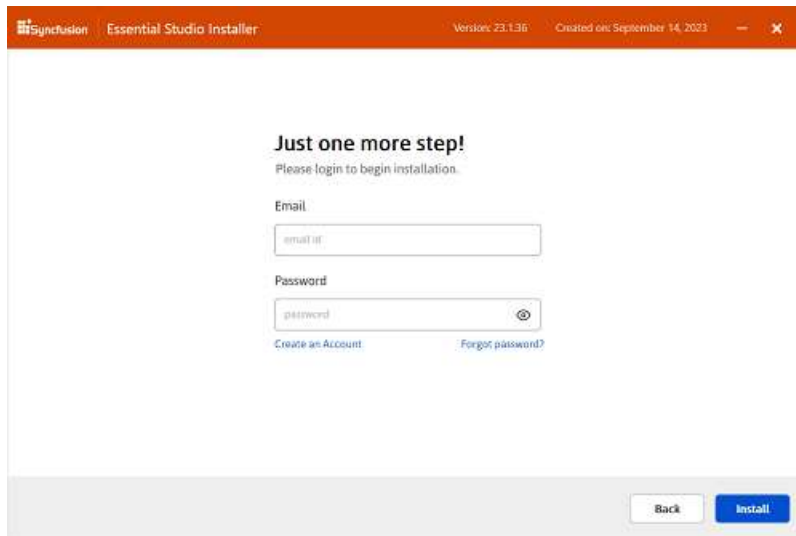
Note: By clicking the **Download Size** and **Installation Size** links, you can determine the approximate size of the download and installation

7. The Configuration Wizard will appear. You can change the Download, Install, and Demos locations from here. You can also change the Additional settings on a product-by-product basis. Click Next to install with the default settings.



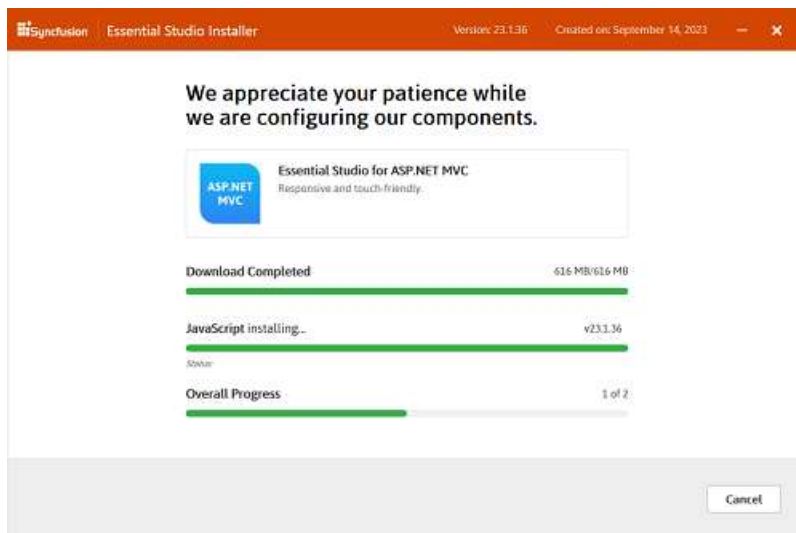
### Additional settings

- Select the **Install Demos** check box to install Syncfusion samples, or leave the check box unchecked, if you do not want to install Syncfusion samples
- Select the **Configure Syncfusion Extensions controls in Visual Studio** checkbox to configure the Syncfusion Extensions in Visual Studio or clear this check box when you do not want to configure the Syncfusion Extensions in Visual Studio.
- Check the **Create Desktop Shortcut** checkbox to add a desktop shortcut for Syncfusion Control Panel
- Check the **Create Start Menu Shortcut** checkbox to add a shortcut to the start menu for Syncfusion Control Panel
  8. After reading the License Terms and Conditions, check the **I agree to the License Terms and Privacy Policy** check box. Click the Next button.
  9. The login wizard will appear. You must enter your Syncfusion email address and password. If you do not already have a Syncfusion account, you can create one by clicking on **Create an Account**. If you have forgotten your password, click **Forgot Password** to create a new one. Click the Install button.

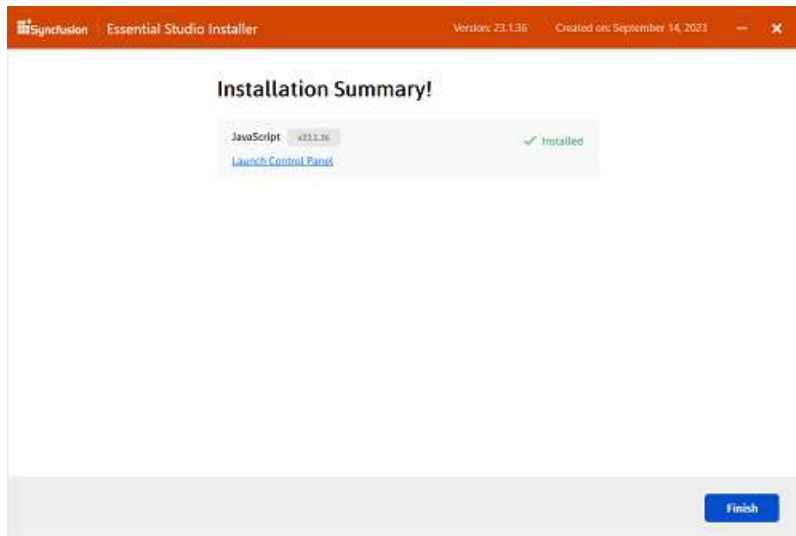


Important: The products you have chosen will be installed based on your Syncfusion License (Trial or Licensed).

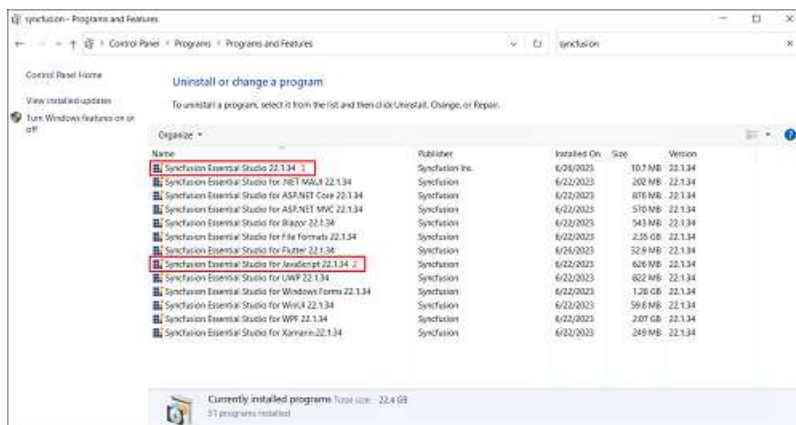
10. The download and installation/uninstallation progress will be displayed as shown below.



11. When the installation is finished, the **Summary** wizard will appear. Here you can see the list of products that have been installed successfully and those that have failed. To close the Summary wizard, click Finish.



- To open the Syncfusion Control Panel, click **Launch Control Panel**.
12. After installation, there will be two Syncfusion control panel entries, as shown below. The Essential Studio entry will manage all Syncfusion products installed in the same version, while the Product entry will only uninstall the specific product setup.



## Uninstallation

Syncfusion JavaScript – EJ2 installer can be uninstalled in two ways.

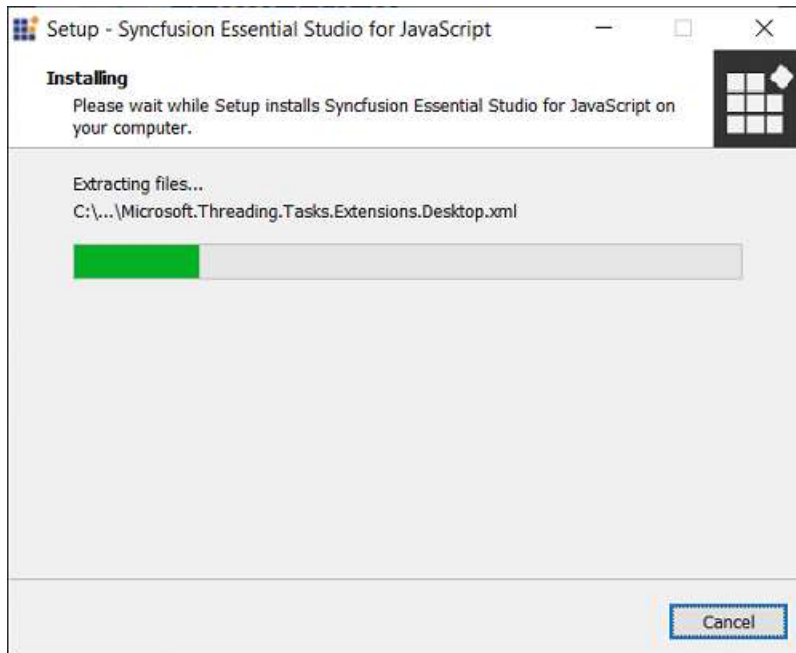
- Uninstall the JavaScript – EJ2 using the Syncfusion JavaScript – EJ2 web installer
- Uninstall the JavaScript – EJ2 from Windows Control Panel

Follow either one of the option below to uninstall Syncfusion Essential Studio JavaScript – EJ2 installer

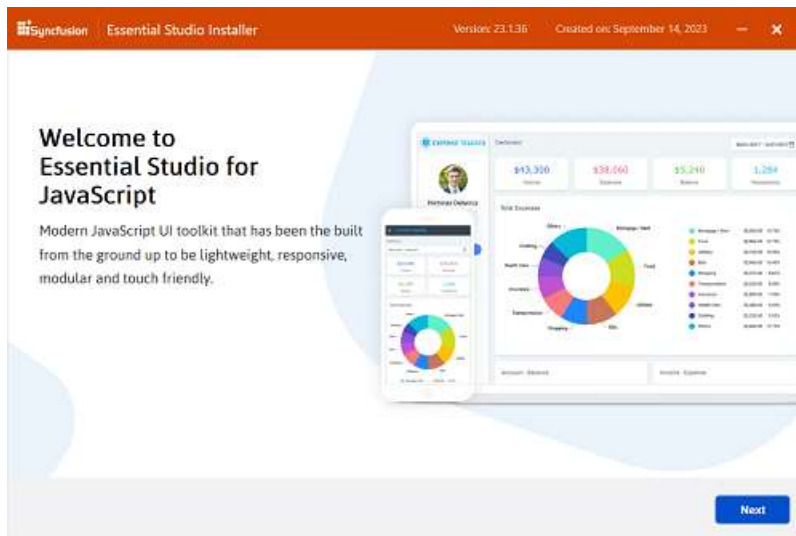
### *Option 1: Uninstall the JavaScript–EJ2 using the Syncfusion JavaScript–EJ2 web installer*

Syncfusion provides the option to uninstall products of the same version directly from the Web Installer application. Select the products to be uninstalled from the list, and Web Installer will uninstall them one by one.

Open the Syncfusion Essential Studio JavaScript – EJ2 Online Installer file from downloaded location by double-clicking it. The Installer Wizard automatically opens and extracts the package

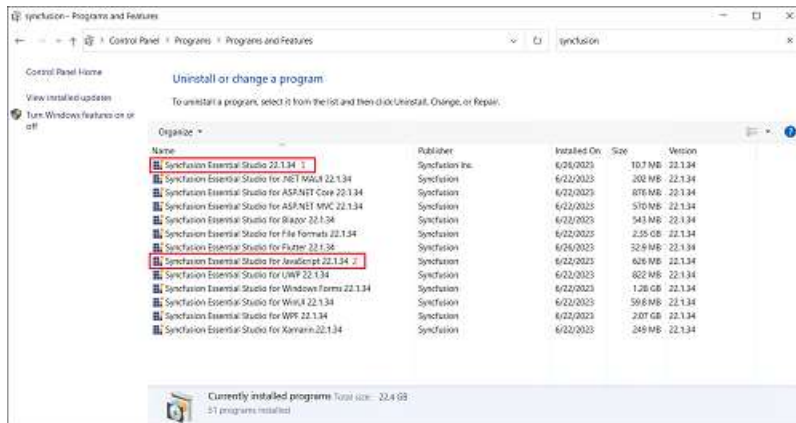


The Syncfusion JavaScript – EJ2 Web Installer’s welcome wizard will be displayed. Click the Next button



#### *Option 2: Uninstall the JavaScript–EJ2 from windows control panel*

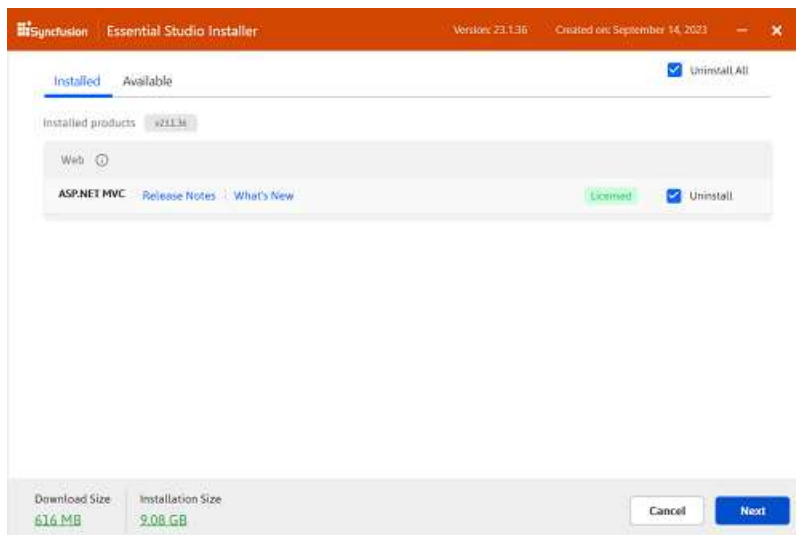
You can uninstall all the installed products by selecting the **Syncfusion Essential Studio {version}** entry (element 1 in the below screenshot) from the Windows control panel, or you can uninstall JavaScript – EJ2 alone by selecting the **Syncfusion Essential Studio for JavaScript – EJ2 {version}** entry (element 2 in the below screenshot) from the Windows control panel.



Note: If the **Syncfusion Essential Studio** for JavaScript {version} entry is selected from the Windows control panel, the Syncfusion Essential Studio JavaScript – EJ2 alone will be removed and the below default MSI uninstallation window will be displayed.

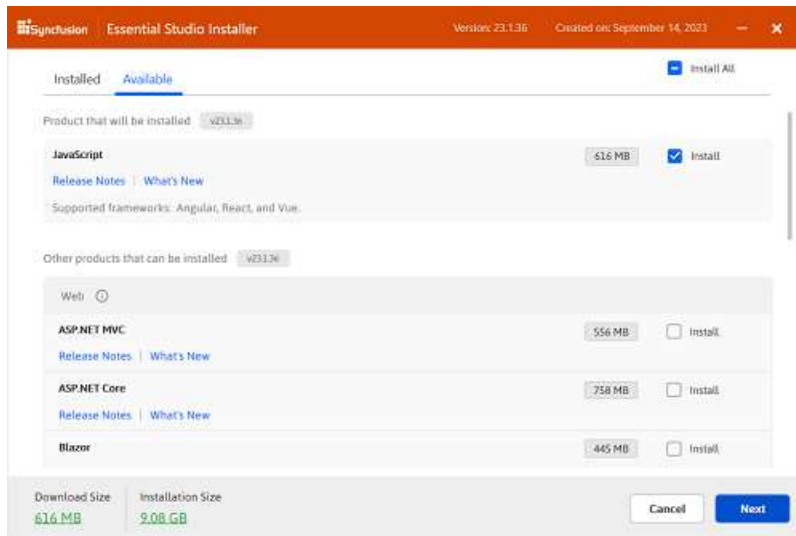
1. The Platform Selection Wizard will appear. From the **Installed** tab, select the products to be uninstalled. To select all products, check the **Uninstall All** checkbox. Click the Next button.

## Installed

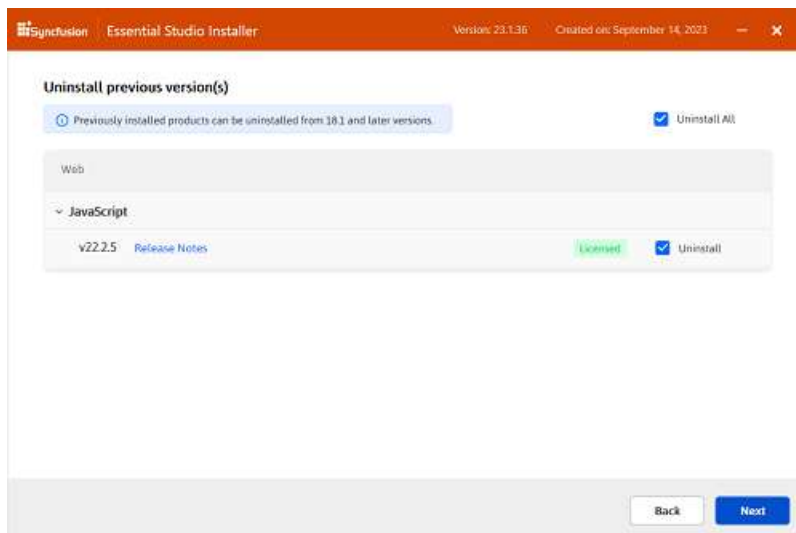


You can also select the products to be installed from the **Available** tab. Click the Next button.

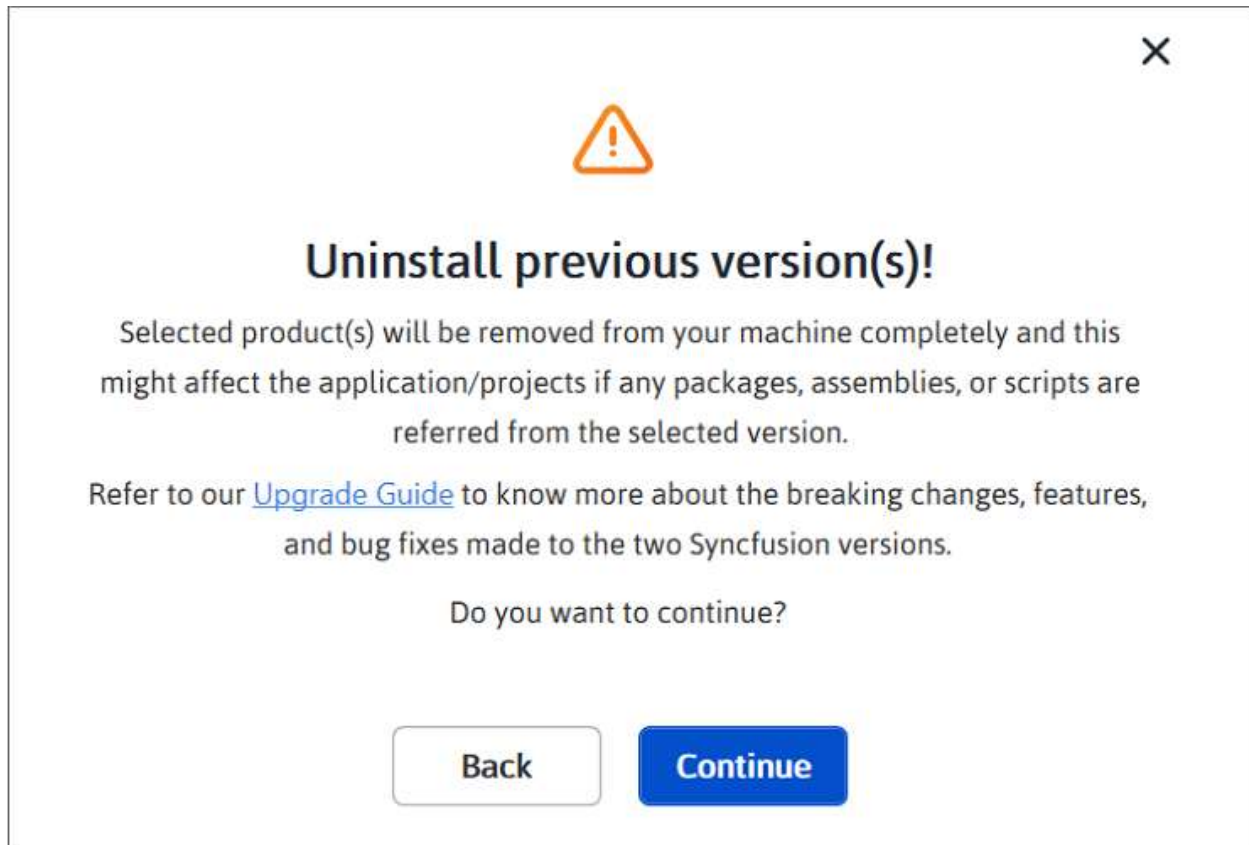
## Available



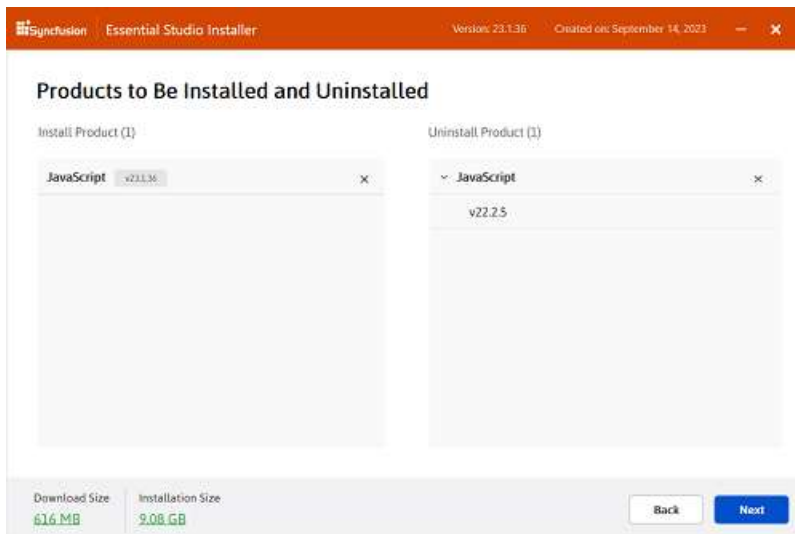
2. If any other products selected for installation, Uninstall previous version wizard will be displayed with previous version(s) installed for the selected products. Here you can view the list of installed previous versions for the selected products. Select **Uninstall All** checkbox to select all the versions. Click Next.



3. Pop up screen will be displayed to get the confirmation to uninstall selected previous versions.



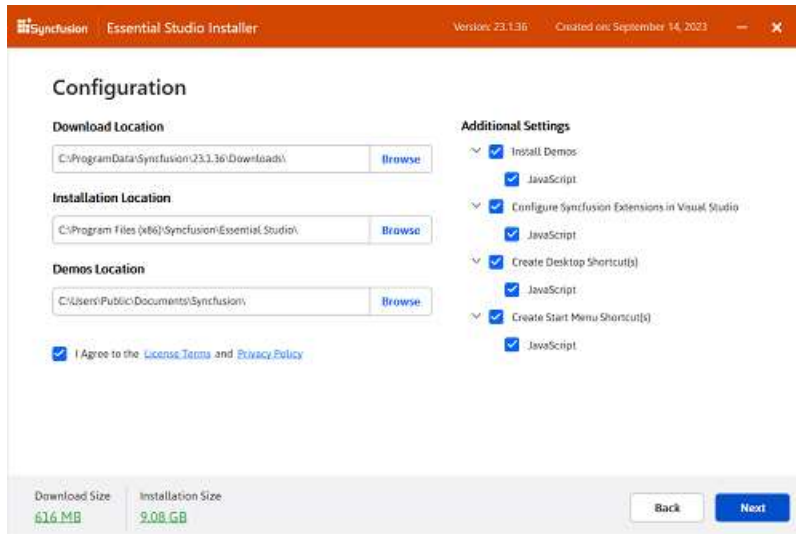
4. The Confirmation Wizard will appear with the list of products to be installed/uninstalled. Here you can view and modify the list of products that will be installed/uninstalled.



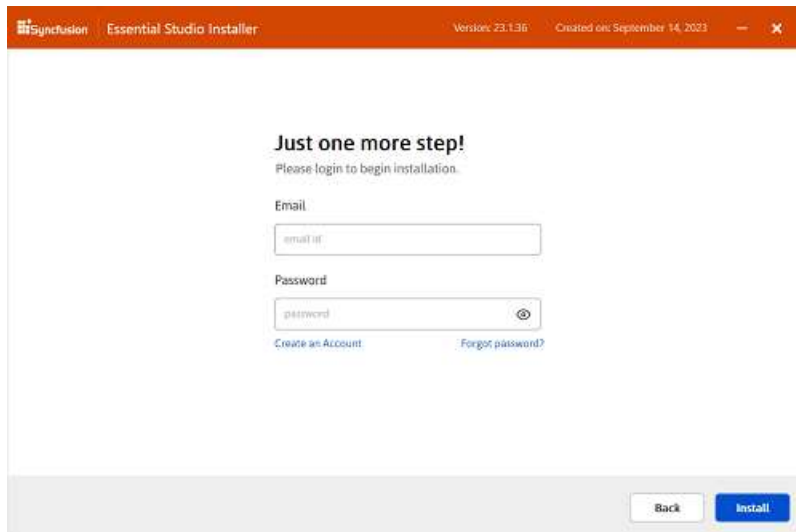
Note: By clicking the **Download Size** and **Installation Size** links, you can determine the approximate size of the download and installation



- The Configuration Wizard will appear. You can change the Download, Install, and Demos locations from here. You can also change the Additional settings on a product-by-product basis. Click Next to install with the default settings.

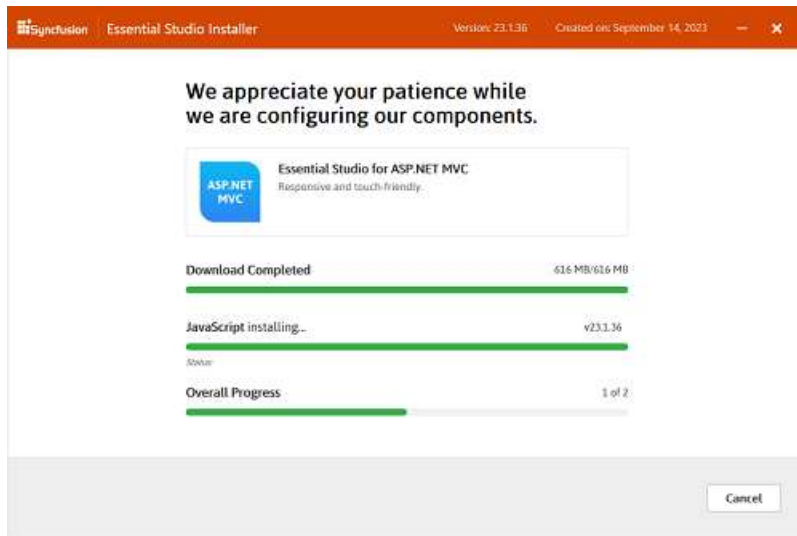


- After reading the License Terms and Conditions, check the **I agree to the License Terms and Privacy Policy** check box. Click the Next button.
- The login wizard will appear. You must enter your Syncfusion email address and password. If you do not already have a Syncfusion account, you can create one by clicking on **Create an Account**. If you have forgotten your password, click **Forgot Password** to create a new one. Click the Install button.

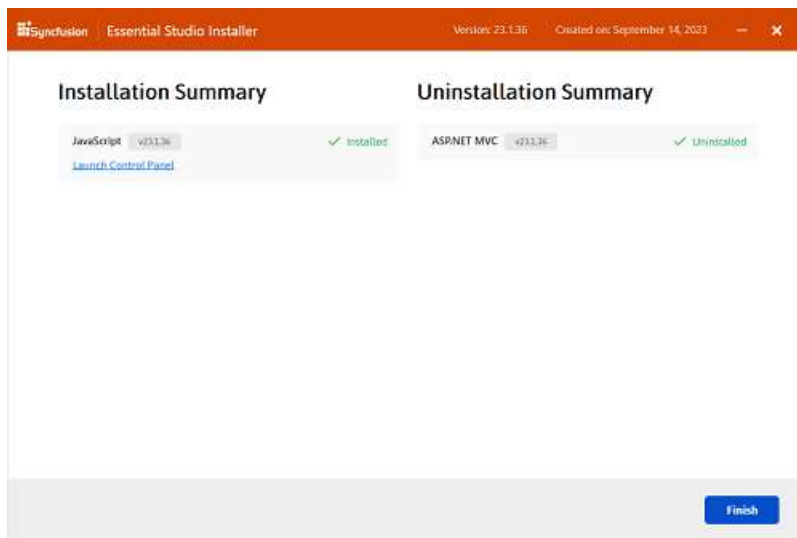


Important: The products you have chosen will be installed based on your Syncfusion License (Trial or Licensed).

- The download, installation, and uninstallation progresses will be shown.



9. When the installation is finished, the **Summary** wizard will appear. Here you can see the list of products that have been successfully and unsuccessfully installed/uninstalled. To close the Summary wizard, click Finish.



- To open the Syncfusion Control Panel, click **Launch Control Panel**

### Installation using Offline Installer

You can refer to the [Download](#) section to learn how to get the JavaScript – EJ2 trial or licensed installer.

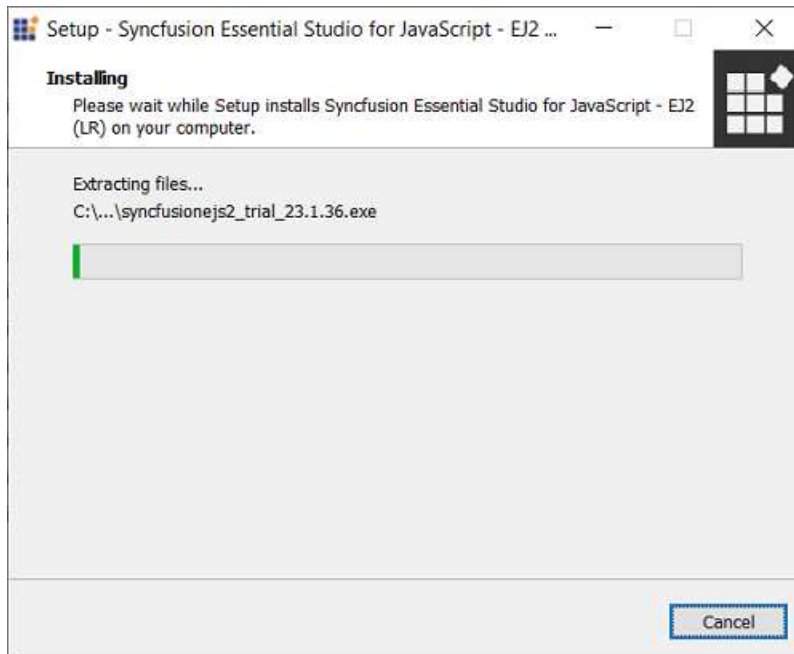
The frameworks listed below are supported in this installer.

- JavaScript
- Angular
- React
- Vue
- JavaScript (ES5)

## Installing with UI

The steps below show how to install the Essential Studio JavaScript – EJ2 installer.

1. Open the Syncfusion JavaScript – EJ2 offline installer file from downloaded location by double-clicking it. The Installer Wizard automatically opens and extracts the package



Note: The Installer wizard extracts the syncfusionjs2 (version).exe dialog, which displays the package's unzip operation.

2. To unlock the Syncfusion offline installer, you have two options:
  - Login To Install
  - Use Unlock Key

### Login To Install

You must enter your Syncfusion email address and password. If you don't already have a Syncfusion account, you can sign up for one by clicking **"Create an account"**. If you have forgotten your password, click on **"Forgot Password"** to create a new one. Once you've entered your Syncfusion email and password, click Next.

**Syncfusion**

## Essential Studio for JavaScript

Version: 23.1.36, Date: September 14, 2023

Supported frameworks: Angular, React, and Vue.

Thank you for choosing Syncfusion Essential Studio. Please enter the required information to proceed further.

**Login To Install**    [Use Unlock Key](#)

Email \*

Password \*

[Forgot Password?](#)

Don't have an account? [Create an account](#)

☐ I agree to the [License Terms](#) and [Privacy Policy](#)

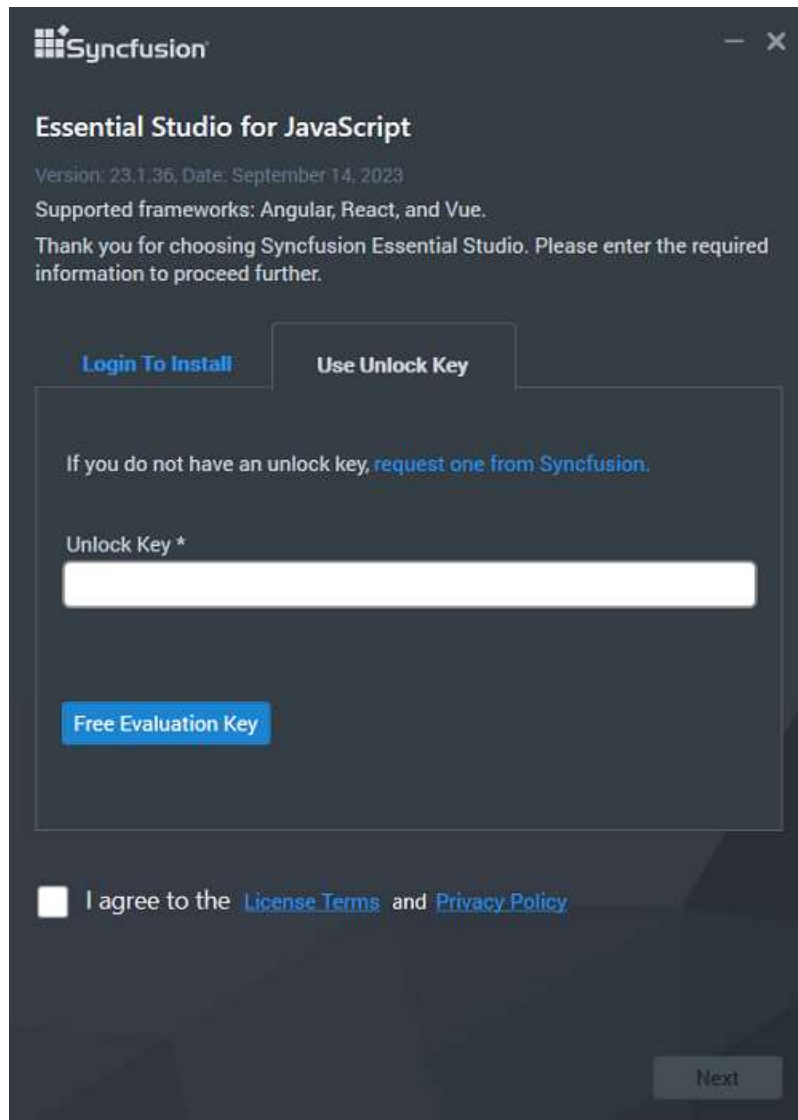
Next

### Use Unlock Key

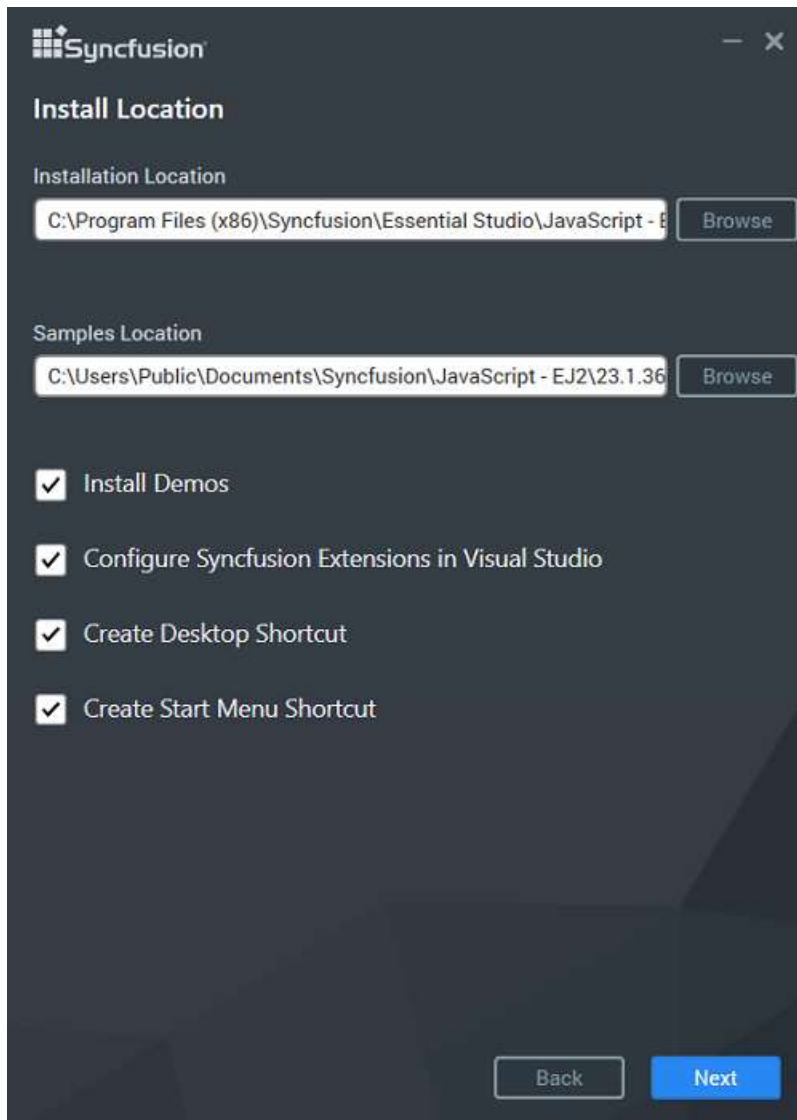
Unlock keys are used to unlock the Syncfusion offline installer, and they are platform and version specific. You should use either Syncfusion licensed or trial Unlock key to unlock Syncfusion JavaScript – EJ2 installer.

The trial unlock key is only valid for 30 days, and the installer will not accept an expired trial key.

To learn how to generate an unlock key for both trial and licensed products, see [this](#) Knowledge Base article.

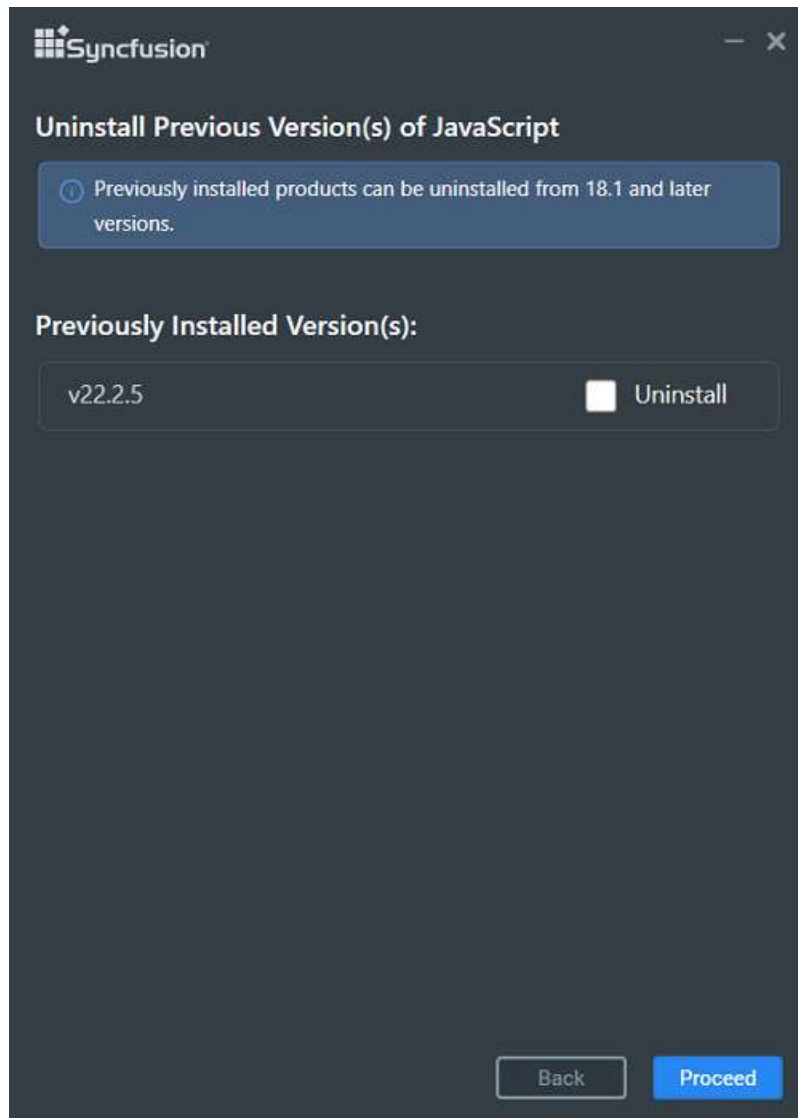


3. After reading the License Terms and Privacy Policy, check the **"I agree to the License Terms and Privacy Policy"** check box. Click the Next button.
4. Change the install and sample locations here. You can also change the Additional settings. Click Next/Install to install with the default settings.



### Additional settings

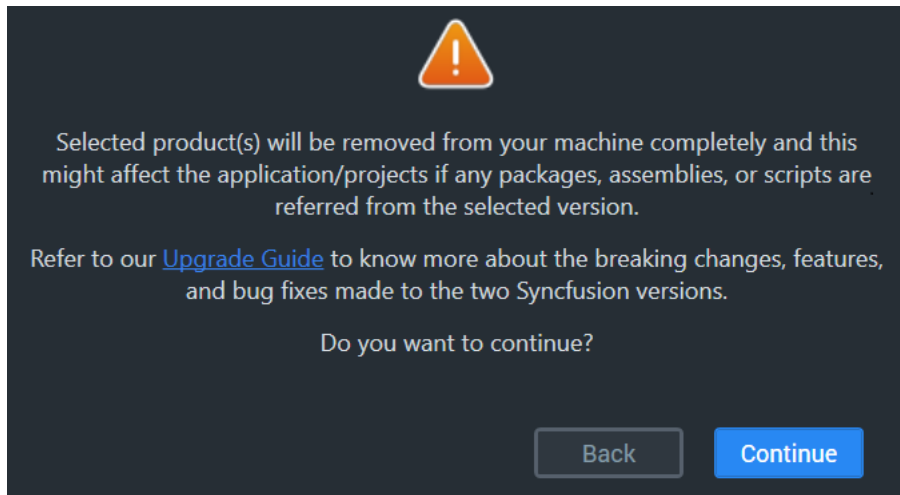
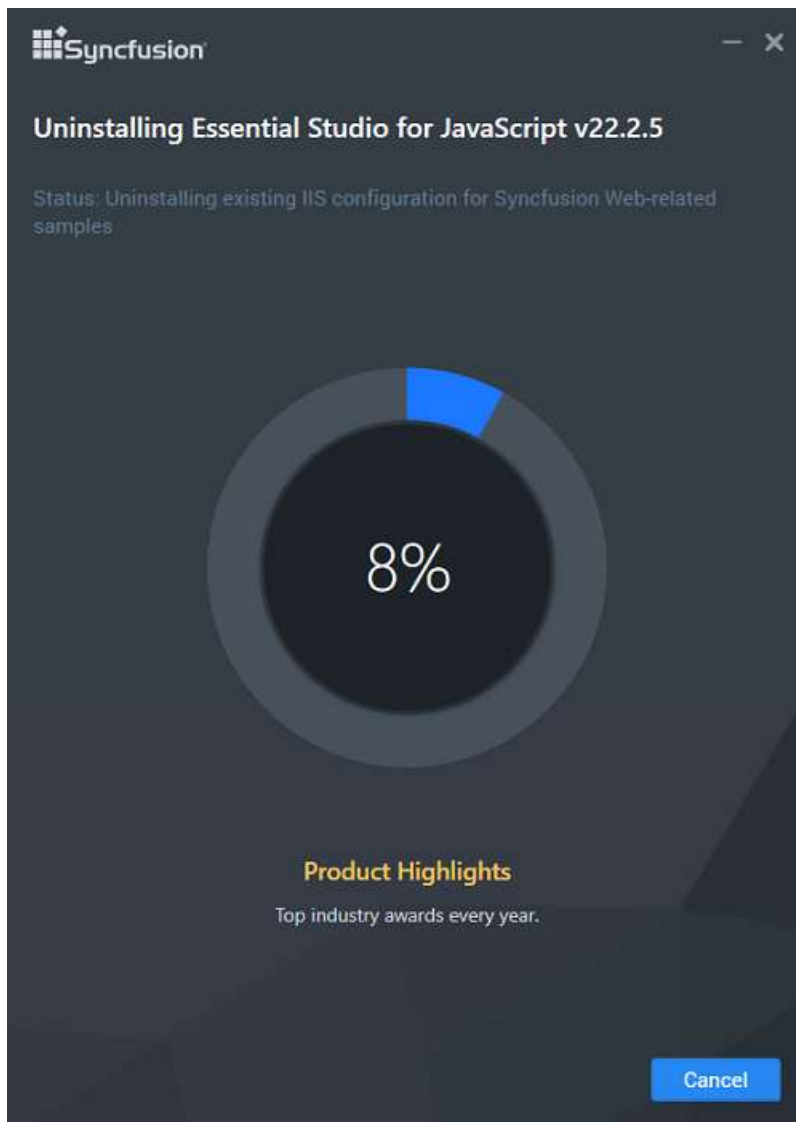
- Select the **Install Demos** check box to install Syncfusion samples, or leave the check box unchecked, if you do not want to install Syncfusion samples
  - Select the **Configure Syncfusion Extensions in Visual Studio** checkbox to configure the Syncfusion Extensions in Visual Studio or clear this check box when you do not want to configure the Syncfusion Extensions in Visual Studio.
  - Check the **Create Desktop Shortcut** checkbox to add a desktop shortcut for Syncfusion Control Panel
  - Check the **Create Start Menu Shortcut** checkbox to add a shortcut to the start menu for Syncfusion Control Panel
5. If any previous versions of the current product is installed, the **Uninstall Previous Version(s)** wizard will be opened. Select Uninstall checkbox to uninstall the previous versions and then click the Proceed button.



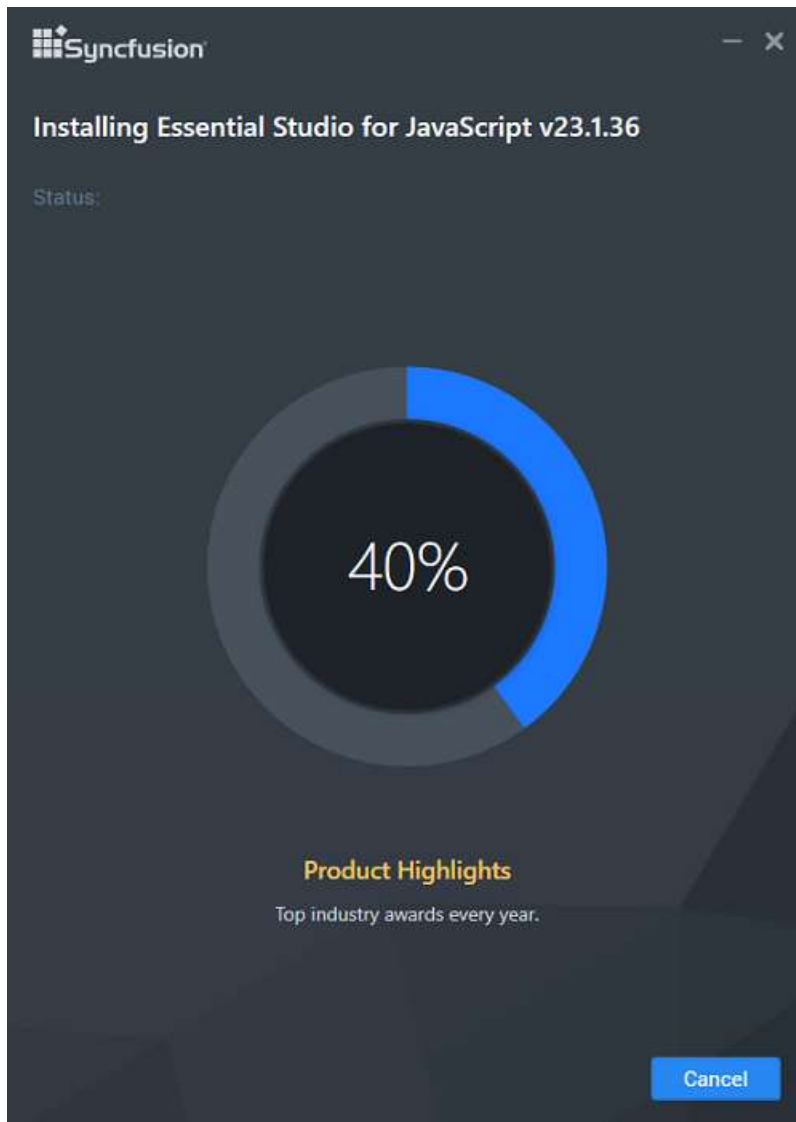
Note: From the 2021 Volume 1 release, Syncfusion has added the option to uninstall previous versions from 18.1 while installing the new version.

Note: If any version is selected to uninstall, a confirmation screen will appear; if continue is selected, the Progress screen will display the uninstall and install progress, respectively. If none of the versions are chosen to be uninstalled, only the installation progress will be displayed.

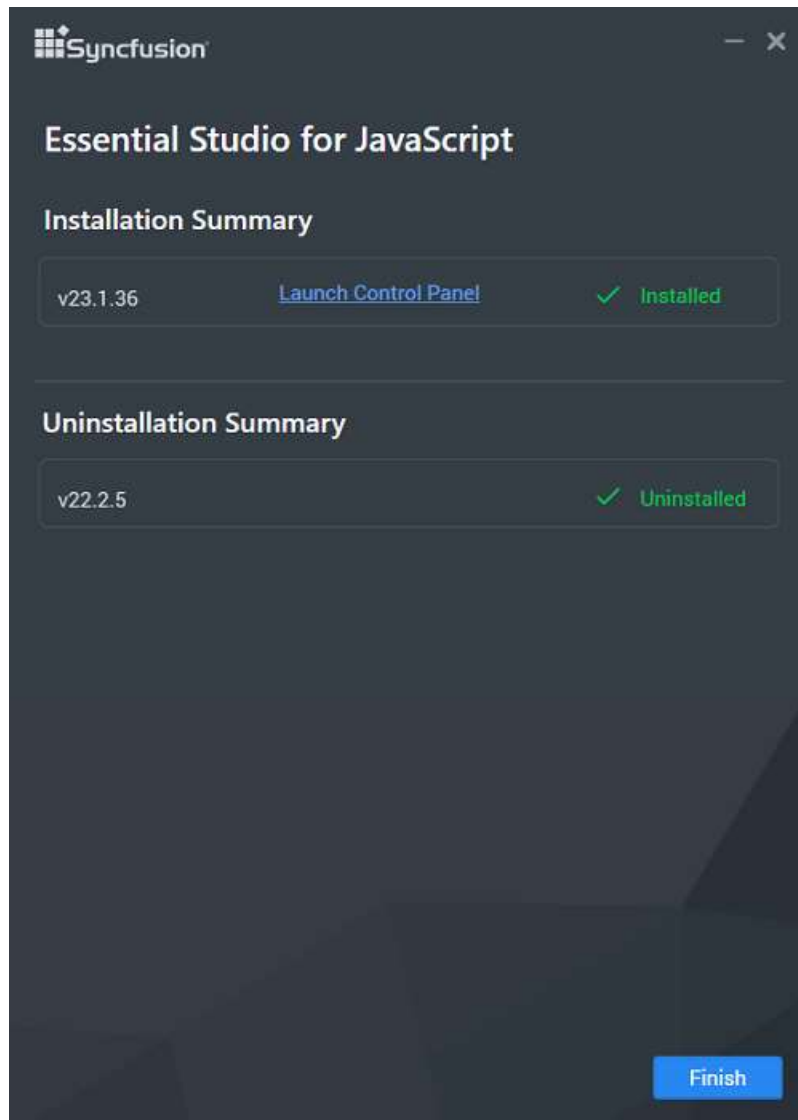
#### Confirmation alert

**Uninstall progress:**



**Install progress**

Note: The Completed screen is displayed once the JavaScript – EJ2 product is installed. If any version is selected to uninstall, The completed screen will display both install and uninstall status.



6. After installing, click the Launch Control Panel link to open the Syncfusion Control Panel.
7. Click the Finish button. Your system has been installed with the Syncfusion Essential Studio JavaScript – EJ2 product.

### Installing in silent mode

The Syncfusion Essential Studio JavaScript – EJ2 Installer supports installation and uninstallation via the command line.

#### Command line installation

To install through the Command Line in Silent mode, follow the steps below.

1. Run the Syncfusion JavaScript – EJ2 installer by double-clicking it. The Installer Wizard automatically opens and extracts the package.
2. The file `syncfusionejs2_(version).exe` file will be extracted into the Temp directory.
3. Run `%temp%`. The Temp folder will be opened. The `syncfusionejs2_(version).exe` file will be located in one of the folders.

4. Copy the extracted syncfusionejs2\_(version).exe file in local drive.
5. Exit the Wizard.
6. Run Command Prompt in administrator mode and enter the following arguments.

Arguments: "installer file path/SyncfusionEssentialStudio(product)\_(version).exe" /Install silent /UNLOCKKEY:"(product unlock key)" [/log "{Log file path}"] [/InstallPath:{Location to install}] [/InstallSamples:{true/false}] [/InstallAssemblies:{true/false}] [/UninstallExistAssemblies:{true/false}] [/InstallToolbox:{true/false}]

Note: [...] – Arguments inside the square brackets are optional.

**Example:** "D:/Temp/syncfusionejs2x.x.x.x.exe" /Install silent /UNLOCKKEY:"product unlock key" /log "C:/Temp/EssentialStudioPlatform.log" /InstallPath:C:/Syncfusion/x.x.x.x /InstallSamples:true /InstallAssemblies:true /UninstallExistAssemblies:true /InstallToolbox:true

7. Essential Studio for JavaScript (Essential JS2) is installed.

Note: x.x.x.x should be replaced with the Essential Studio version and the Product Unlock Key needs to be replaced with the Unlock Key for that version.

#### *Command line uninstallation*

Syncfusion Essential JavaScript – EJ2 can be uninstalled silently using the Command Line.

1. Run the Syncfusion JavaScript – EJ2 installer by double-clicking it. The Installer Wizard automatically opens and extracts the package.
2. The syncfusionejs2\_(version).exe file will be extracted into the Temp directory.
3. Run %temp%. The Temp folder will be opened. The syncfusionejs2\_(version).exe file will be located in one of the folders.
4. Copy the extracted syncfusionejs2\_(version).exe file in local drive.
5. Exit the Wizard.
6. Run Command Prompt in administrator mode and enter the following arguments.

Arguments: "Copied installer file path/ syncfusionejs2\_(version).exe" /uninstall silent

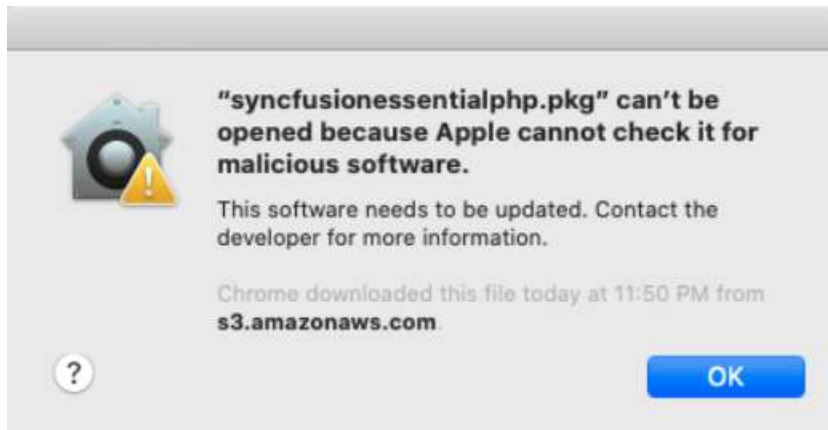
Example: "D:/Temp/syncfusionejs2\_x.x.x.x.exe" /uninstall silent

7. Essential Studio for JavaScript (Essential JS2) is uninstalled.

#### *Installing Syncfusion JavaScript – EJ2 Mac Installer*

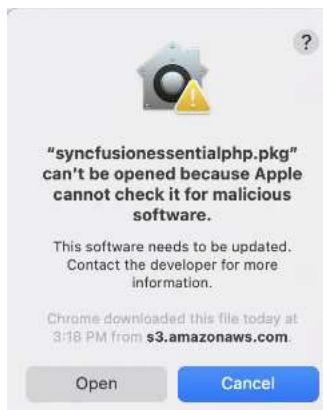
*Steps to resolve the warning message in Catalina OS or later*

While running Essential Studio JavaScript - EJ2 Mac Installer on Catalina MacOS or later, the below alert will be displayed.



If you receive this alert, follow the below steps for the easiest solution.

1. Right-click the downloaded pkg file.
2. Select the "Open With" option and choose "Installer (Default)". The following pop-up appears.

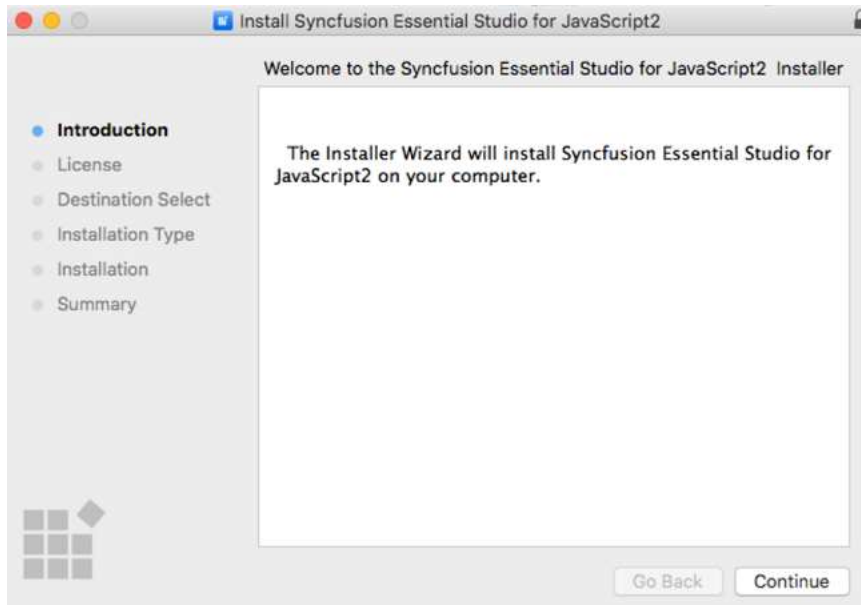


3. When you click "Open" the installer window will be opened.

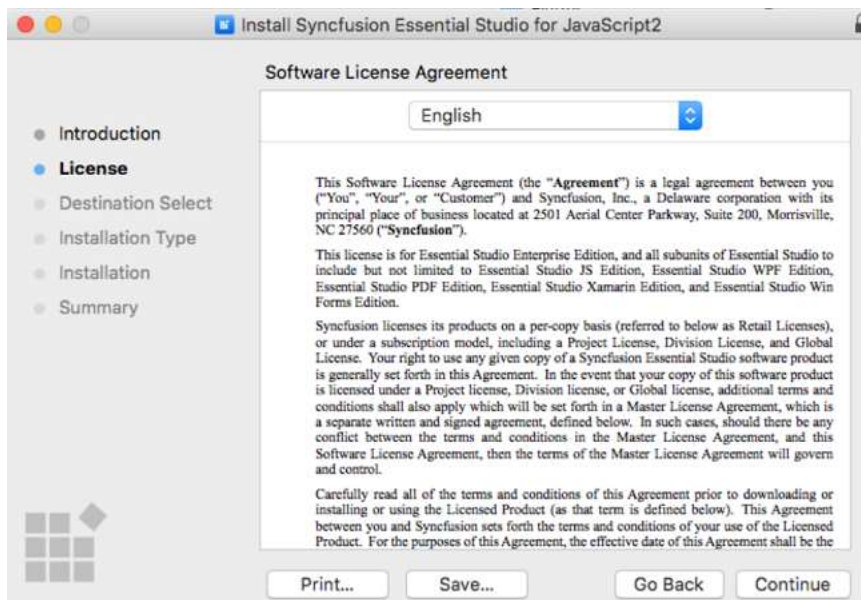
### Step-by-Step Installation

The steps below show how to install the Essential Studio JavaScript - EJ2 Mac installer.

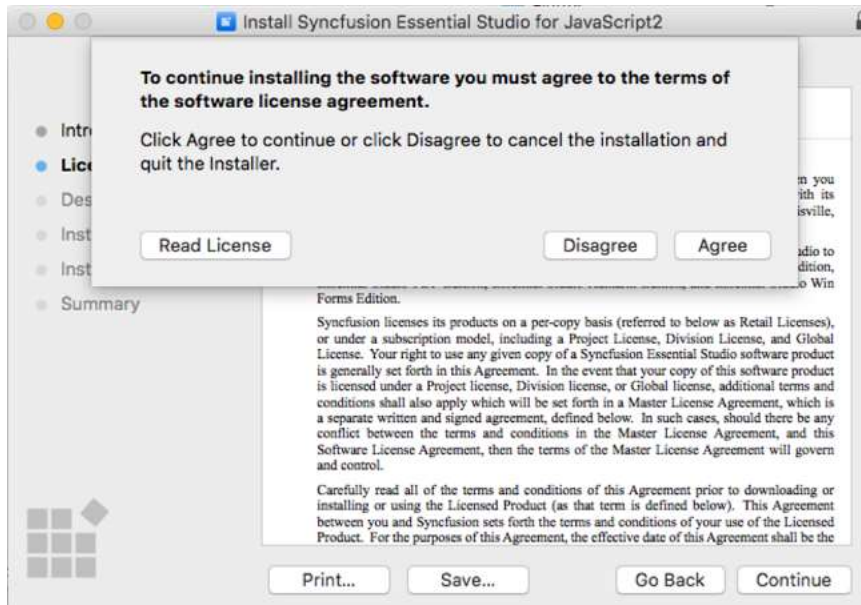
1. Double-click the Syncfusion Essential Studio JavaScript - EJ2 Mac installer(.pkg) file. The installer Wizard opens. Click Continue.



2. The Software License Agreement wizard will appear. Click the Continue button.

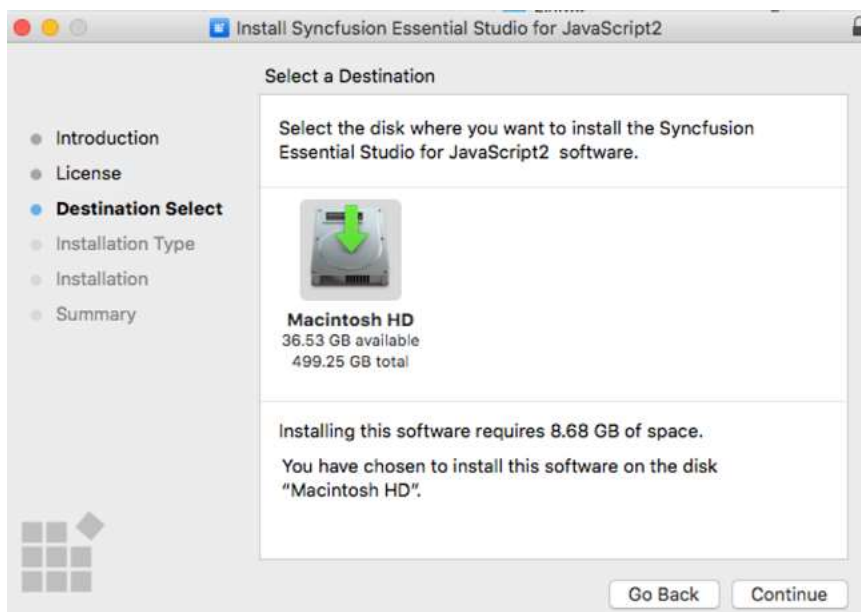


3. The License Agreement's Confirmation window will appear. If you have read the Software License Agreement, click **Agree**.

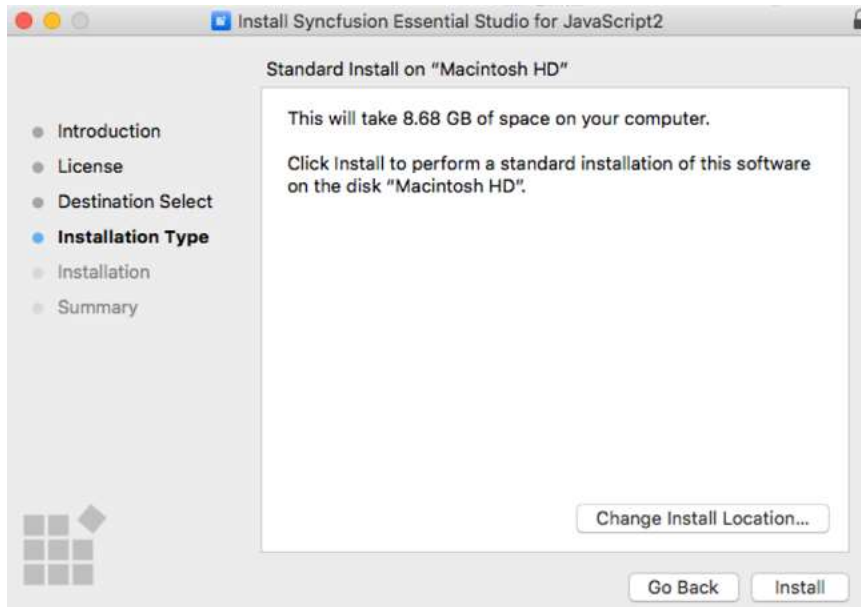


**Note:** The Unlock key is not required to install the Mac installer. The Syncfusion Essential Studio JavaScript - EJ2 Mac installer can be used for development purposes without registering the Unlock key.

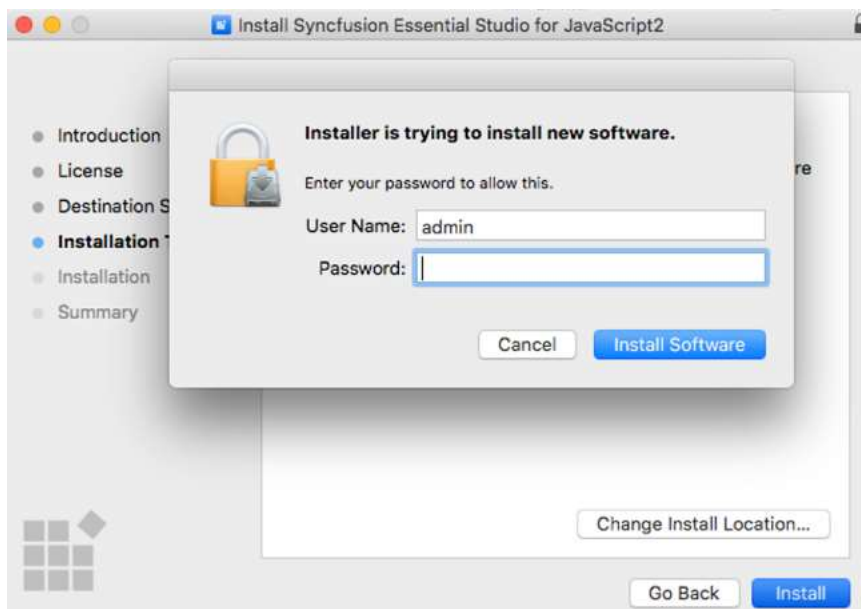
4. The Destination select wizard will appear. You can choose which disc to install the Syncfusion Essential Studio for JavaScript - EJ2 Mac installer on here.



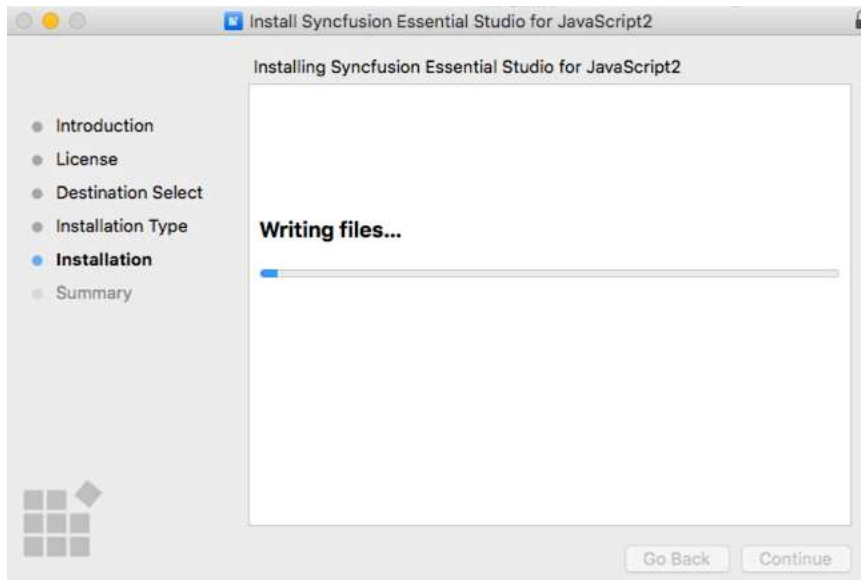
5. The Installation Type wizard will appear. Click Install to begin the standard installation of the Syncfusion Essential Studio JavaScript - EJ2 Mac installer.



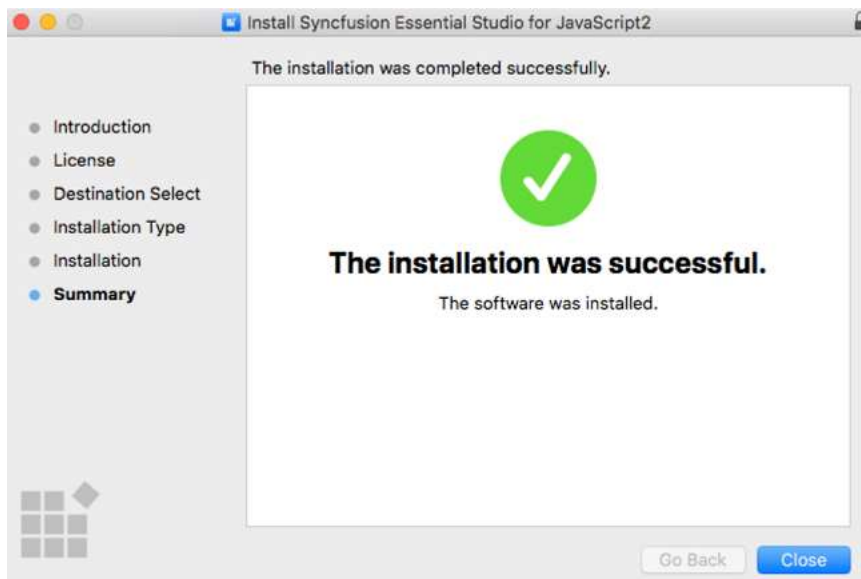
6. The Authentication window will appear. To begin the installation, enter the Mac machine's password and click **Install Software**.



7. The installation process will begin on your machine.



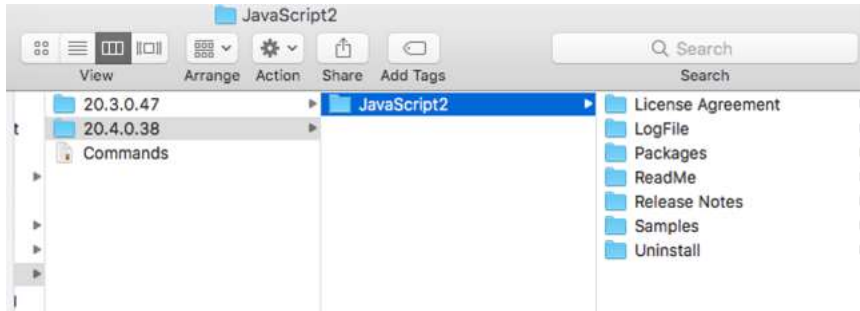
8. Once the installation is complete, the completed screen will be displayed. To exit the installation wizard, click Close.



By default, Mac installer will install the files in following location.

**Location:** {Documents}\Syncfusion\ {version}\ {platform}





### License key registration in samples

After the installation, the license key is required to register the demo source that is included in the Mac installer. To learn about the steps for license registration for the JavaScript - EJ2 Mac installer, please refer to this.

- [Register Syncfusion License key in the project](#)
- [Register Syncfusion license key in the Nuxt project](#)
- [Register the license key using the npx command](#)

### Linux Installer

#### Download Syncfusion JavaScript Linux Installer

The Syncfusion installer can be downloaded from the [Syncfusion](#) website. You can either download the licensed installer or try our trial installer depending on your license.

- Trial Installer - Licensed Installer

You can download the Syncfusion installer from [Syncfusion.com](#) website

#### *Download the Trial Version*

Our 30-day trial can be downloaded in two ways.

- Download Free Trial Setup
- Start Trials if using components through [NuGet.org](#)

#### Download Free Trial Setup

1. You can evaluate our 30-day free trial by visiting the [Download Free Trial](#) page and select the product
2. After completing the required form or logging in with your registered Syncfusion account, you can download the trial installer from the confirmation page. (as shown in below screenshot.)

## Thank you for choosing to evaluate Essential Studio for JavaScript!

Please choose your preferred evaluation experience to proceed.

Starting with v20.1.0.47 (2022 Vol. 1), we're providing licensing support for Essential Studio for JavaScript. Please refer to this [help topic](#) for more information.

### Full installation evaluation

Download and install the evaluation version of the full product along with all the required assemblies and demo code.

Choose Platform

Choose Download Format

[Download](#)

[More Download Options](#)

### NuGet evaluation

No installation required. Get started in 2 minutes

[Install assemblies from NuGet](#)

&

Download demo code from [GitHub](#) | Open demo code in [Visual Studio](#)

Version: 24.1.41, Released: November 28, 2023

[Release Notes](#) | [License Agreement](#) | [Installation Instructions](#) | [Security Management Report](#)

3. With a trial license, only the latest version's trial installer can be downloaded.
4. Unlock key is not required to install the Syncfusion JavaScript Linux trial installer.
5. Before the trial expires, you can download the trial installer at any time from your registered account's [Trials & Downloads](#) page (as shown in below screenshot.)

### Trial Downloads and Unlock Keys

Essential Studio [Trial SKU Name](#)

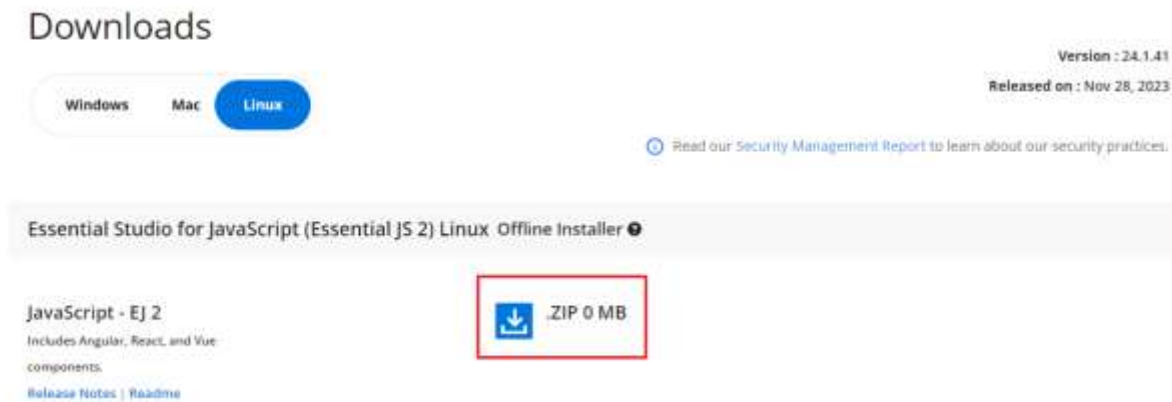
Trial Version: [Trial Version](#)

[What's New](#) | [Generate License File](#) | [Get Unlock Key](#)

[Download](#) **1**

[More Download Options](#) **2**

6. Click the More Download Options (element 2 in the above screenshot) button to get the JavaScript Product Offline trial installer which is available in ZIP format.

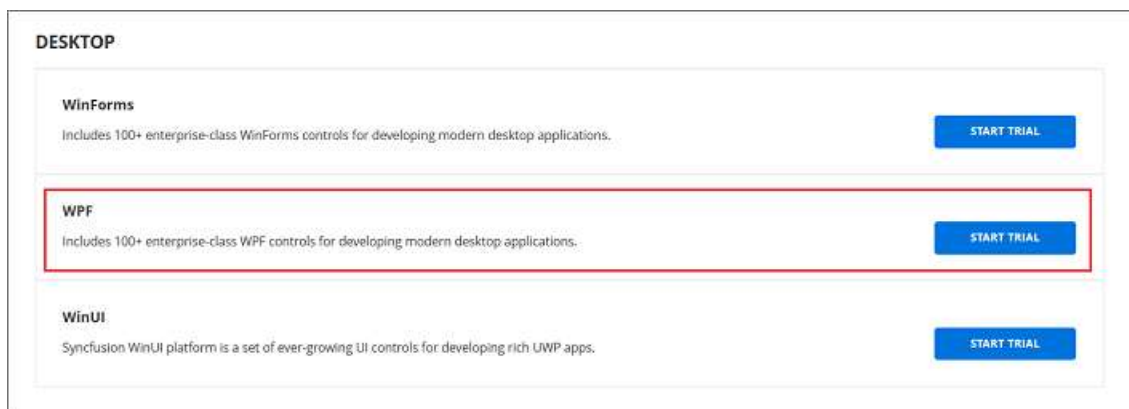


Start Trials if using components through [NuGet.org](https://www.nuget.org/packages?q=syncfusion)

You should initiate an evaluation if you have already obtained our components through [NuGet.org](https://www.nuget.org/packages?q=syncfusion)

1. You can start your 30-day free trial from the [Start Trial](#) page from your account.

**Note:** You can generate the license key for your active trial products from [Trials & Downloads](#) page. This license key will be mandatory to use our trial products in your application. To know more about License key, refer this [help topic](#).



2. To access this page, you must sign up\log in with your Syncfusion account.
3. Begin your trial by selecting the Syncfusion product.

**Note:** If you've already used the trial products and they haven't expired, you won't be able to start the trial for the same product again.

4. After you've started the trial, go to the [Trials & Downloads](#) page to get the latest version trial installer. You can generate the [unlock key](#) and [license key](#) here at any time before the trial period expires. (as shown in below screenshot.)



5. You can find your current active trial products on the [Trials & Downloads](#) page.

#### *Download the License Version*

1. Syncfusion licensed products will be available in the [License & Downloads](#) page under your registered Syncfusion account.
2. You can view all the licenses (both active and expired) associated with your account.
3. You can download JavaScript Linux licensed installer by going to More Downloads Options (element 3 in the screenshot below).



4. Unlock key is not required to install the Syncfusion JavaScript Linux trial installer.
5. For Linux OS, ZIP formats is available for download.

## Downloads

Windows

Mac

Linux

Version : 24.1.41  
Released on : Dec 18, 2023

[Read our Security Management Report](#) to learn about our security practices.

WEB - Essential JS 2

Offline Installer

Blazor

[Release Notes](#) | [Readme](#)

.ZIP 652 MB  
Checksum Value

ASP.NET Core - EJ 2

[Release Notes](#) | [Readme](#)

.ZIP 1092 MB  
Checksum Value

JavaScript - EJ 2

[Release Notes](#) | [Readme](#)

[Includes Angular, React, and Vue components.](#)

.ZIP 2066 MB  
Checksum Value

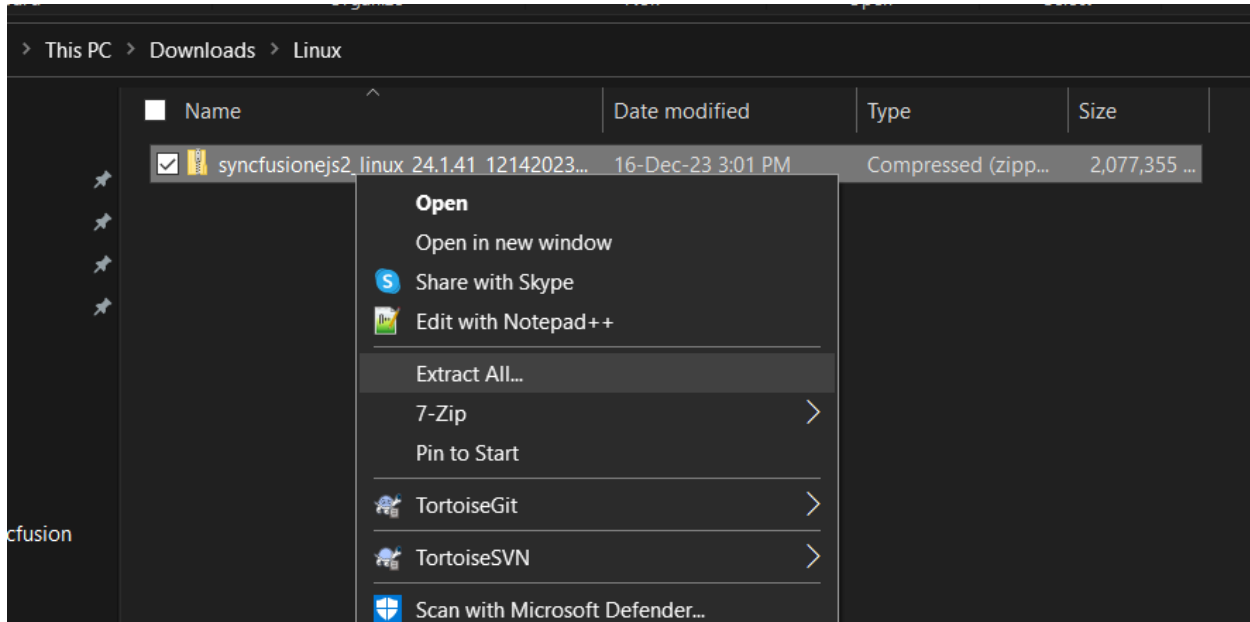
You can also refer to the [JavaScript Linux installer](#) links for step-by-step installation guidelines.

### Installing Syncfusion JavaScript Linux installer

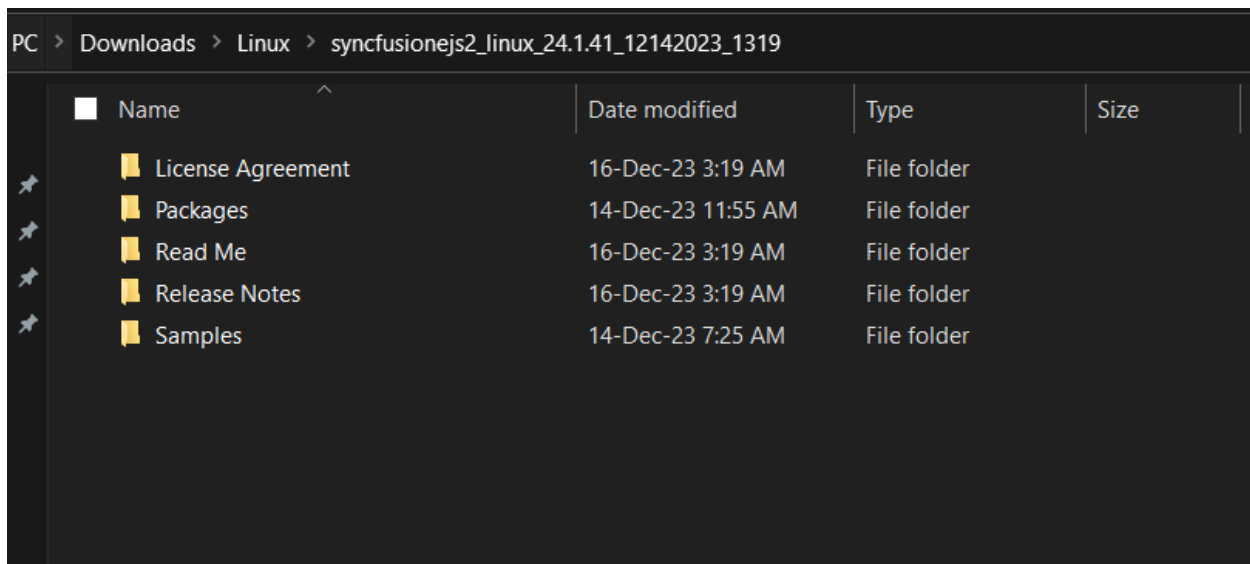
#### *Step-by-Step Installation*

The steps below show how to install JavaScript Linux installer.

1. Extract the Syncfusion JavaScript Linux installer(.zip) file. The files are extracted in your machine.



2. The Linux zip file contains the following folders.



**Note:** The Unlock key is not required to install the Linux installer.

4. You can launch the demo source and use the NuGet packages included in the Linux installer.

### [License key registration in samples](#)

After the installation, the license key is required to register the demo source that is included in the Linux installer. To learn about the steps for license registration for the JavaScript - EJ2 Linux installer, please refer to this.

- [Register Syncfusion License key in the project](#)
- [Register Syncfusion license key in the Nuxt project](#)
- [Register the license key using the npx command](#)

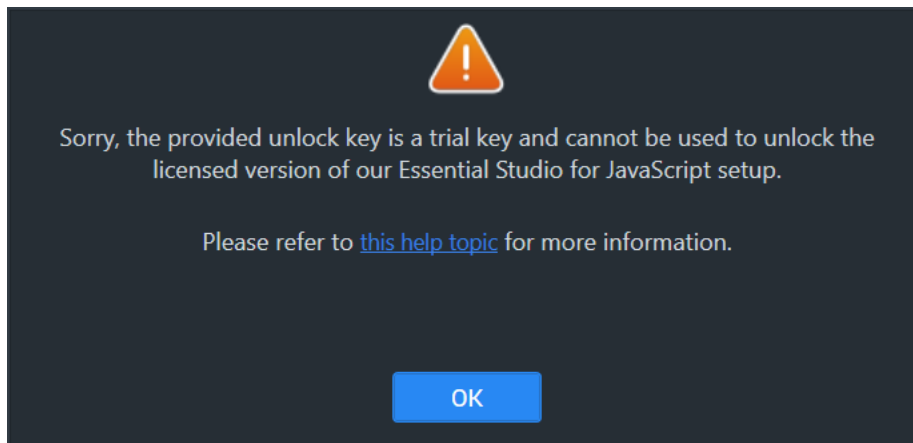
### Common Installation Errors

This article describes the most common installation errors, as well as the causes and solutions to those errors.

- Unlocking the license installer using the trial key
- License has expired
- Unable to find a valid license or trial
- Unable to install because of another installation
- Unable to install due to controlled folder access

#### Unlocking the license installer using the trial key

**Error message:** Sorry, the provided unlock key is a trial unlock key and cannot be used to unlock the licensed version of our Essential Studio for JavaScript installer.



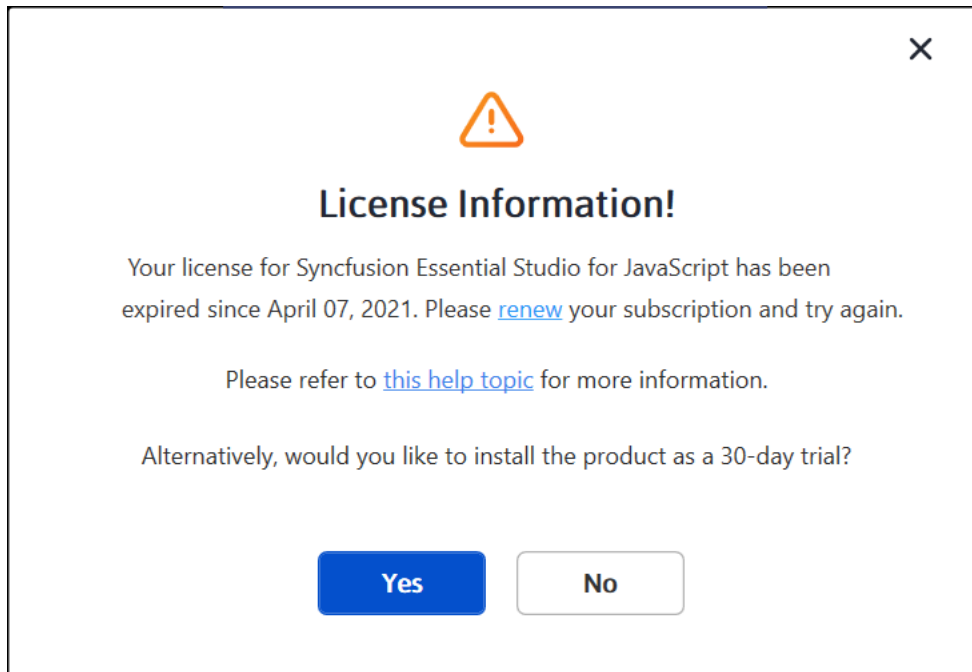
**Reason** <br /> You are attempting to use a Trial unlock key to unlock the licensed installer.

**Suggested solution** <br /> Only a licensed unlock key can unlock a licensed installer. So, to unlock the Licensed installer, use the Licensed unlock key. To generate the licensed unlock key, refer to [this](#) article.

#### License has expired

**Error message:** Your license for Syncfusion Essential Studio for JavaScript – EJ2 has been expired since {date}. Please renew your subscription and try again.

#### Online installer



**Reason** <br /> This error message will appear if your license has expired.

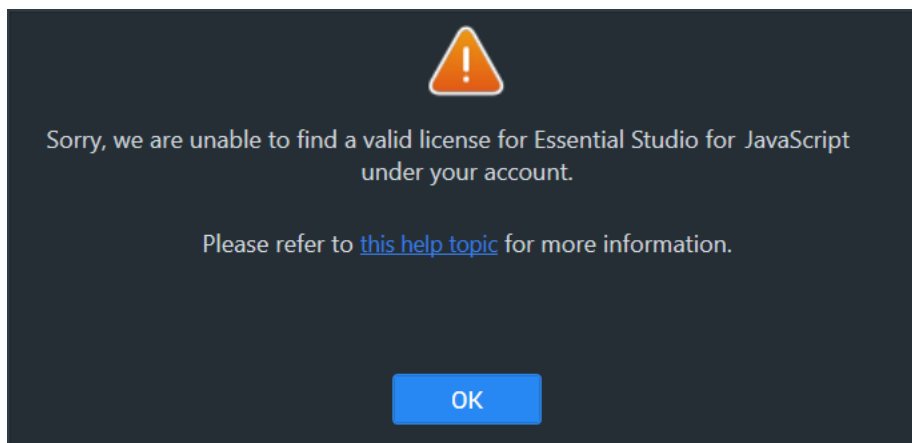
**Suggested solution** <br /> You can choose from the options below.

1. You can renew your subscription [here](#).
2. You can get a new license [here](#).
3. You can reach out to our sales team by emailing [sales@syncfusion.com](mailto:sales@syncfusion.com).
4. You can also extend the 30-day trial period after your trial license has expired.

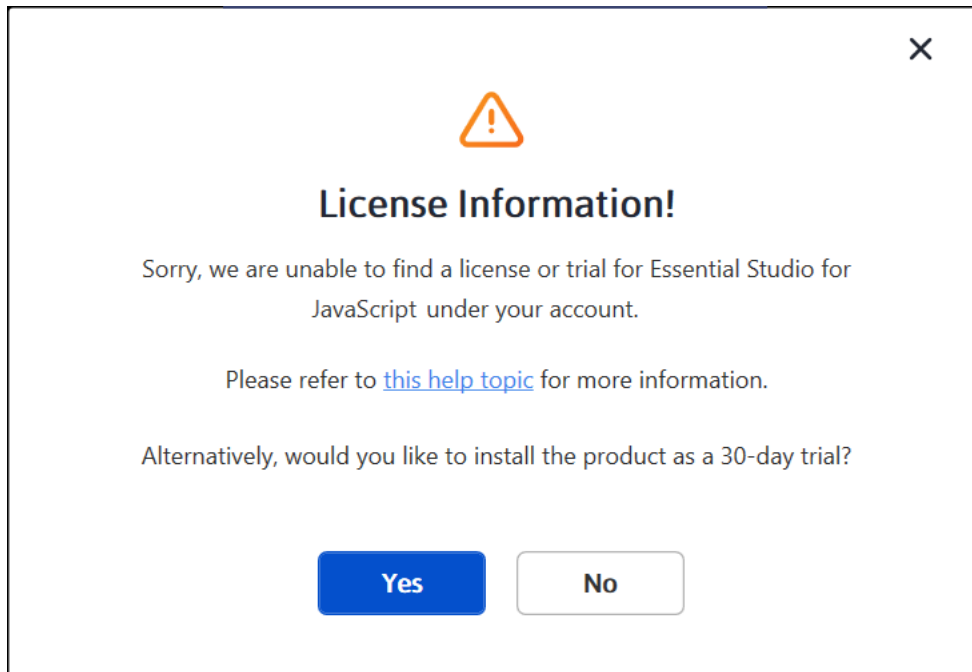
Unable to find a valid license or trial

**Error message:** Sorry, we are unable to find a valid license or trial for Essential Studio for JavaScript – EJ2 under your account.

**Offline installer**



**Online installer**



**Reason** <br /> The following are possible causes of this error:

The following are possible causes of this error:

- When your trial period expired
- When you don't have a license or an active trial
- You are not the license holder of your license
- Your account administrator has not yet assigned you a license.

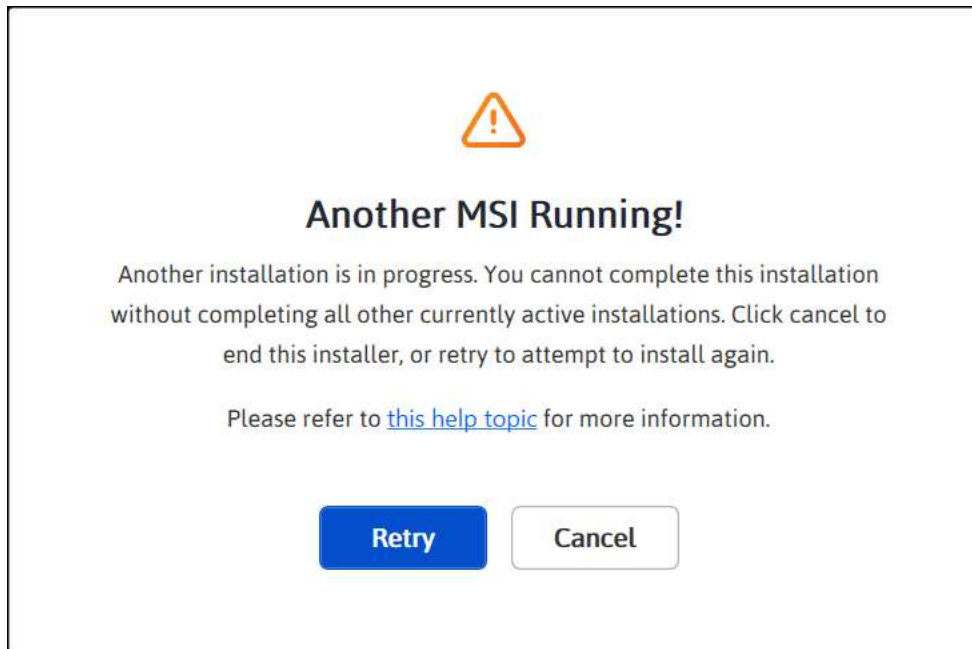
**Suggested solution** <br />

1. You can get a new license [here](#).
2. Contact your account administrator.
3. Send an email to [clientrelations@syncfusion.com](mailto:clientrelations@syncfusion.com) to request a license.
4. You can reach out to our sales team by emailing [sales@syncfusion.com](mailto:sales@syncfusion.com).

Unable to install because of another installation

**Error message:** Another installation is in progress. You cannot start this installation without completing all other currently active installations. Click cancel to end this installer or retry to attempt after currently active installation completed to install again.

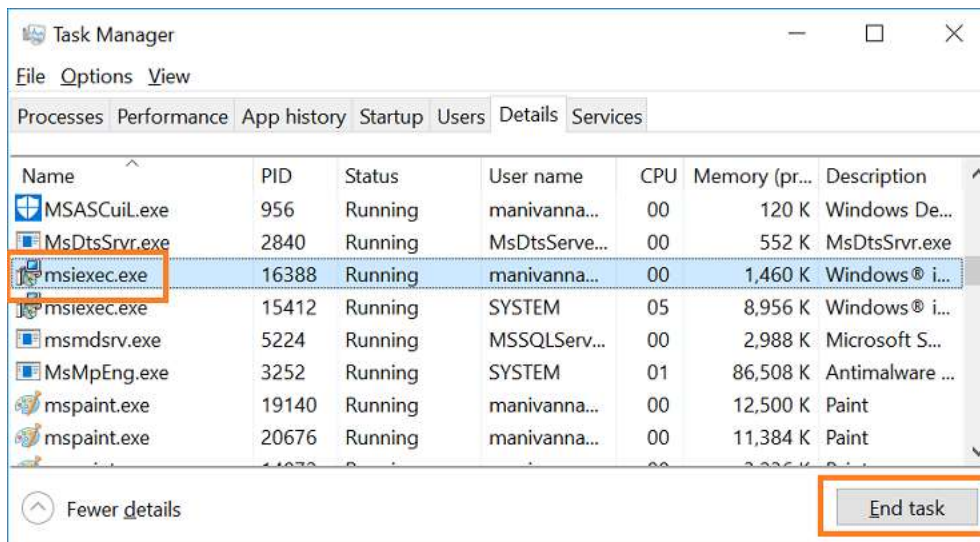




**Reason** <br /> You are trying to install when another installation is already running in your machine.

**Suggested solution** <br /> Open and kill the msixec process in the task manager and then continue to install Syncfusion. If the problem is still present, restart the computer and try Syncfusion installer.

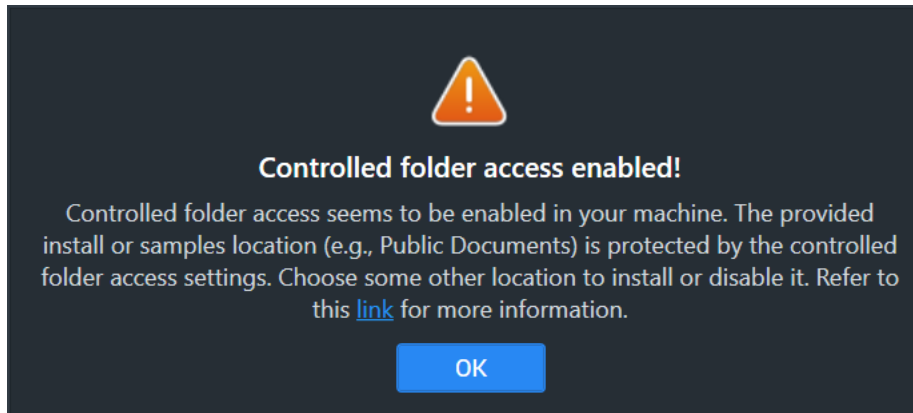
1. Open the Windows Task Manager.
2. Browse the Details tab.
3. Select the msixec.exe and click **End task**.



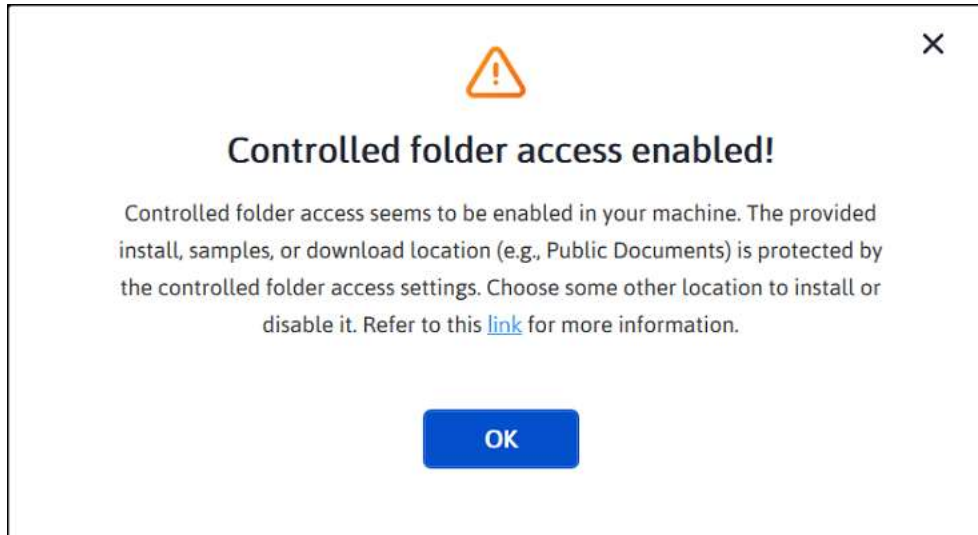
Unable to install due to controlled folder access

### Offline

**Error message:** Controlled folder access seems to be enabled in your machine. The provided install or samples location (e.g., Public Documents) is protected by the controlled folder access settings.

**Online**

**Error message:** Controlled folder access seems to be enabled in your machine. The provided install, samples, or download location (e.g., Public Documents) is protected by the controlled folder access settings.



**Reason** <br /> You have enabled controlled folder access settings on your computer.

**Suggested solution****Suggestion 1:** <br />

1. We will ship our demos in the public documents folder by default.
2. You have controlled folder access enabled on your machine, so our demos cannot be installed in the documents folder. If you need to install our demos in the Documents folder, follow the steps in this [link](#) and disable the controlled folder access.
3. You can enable this option after the installing our Syncfusion setup.

**Suggestion 2:** <br />

1. If you do not want to disable controlled folder access, you can install our demos in another directory.

## Upgrade

### Syncfusion Vue supported versions

#### Vue version compatibility

The following table represents the supported Vue versions by different Syncfusion Vue UI components releases.

Version	Syncfusion Vue components version
-----	-----
<a href="#">Vue v2.7</a>	20.3.47 and above
<a href="#">Vue v3.0</a>	19.2.44 and above

#### Syncfusion version information

Syncfusion follows a quarterly release schedule, introducing new volumes every three months. To track these releases and their associated changes, Syncfusion Vue components utilize a sequence-based identifier system, employing the format **Major.Minor.Revision**. This system enables developers to easily monitor modifications made in each release.

For example, if the release package version is **22.1.34**, the version number can be interpreted as follows:

- **22** represents the **major release** version. This number changes every three months and encompasses significant updates, new features, as well as bug fixes and breaking changes.
- **1** corresponds to the **minor release** version. This number signifies releases primarily focused on new features and addressing bugs, without introducing breaking changes.
- **34** denotes the **revision number**, also referred to as the **patch number**. This number increases for weekly patch releases, which predominantly consist of bug fixes and do not introduce new features or breaking changes.

See also

- [Syncfusion product release lifecycle](#)
- [Upgrade guide](#)

### Upgrading Syncfusion JavaScript (Essential JS2)

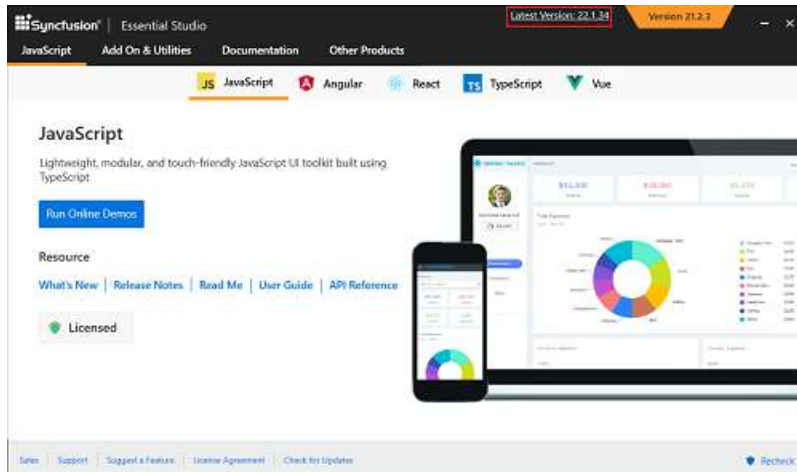
Syncfusion releases new volumes once every three months, with exciting new features. There will be one Service Pack release for these volume releases. Service Pack releases are provided to address major bug fixes in the volume releases.

You can upgrade to our latest version from any installed Syncfusion version.

See our "[Upgrade Guide](#)" for JavaScript – EJ2 to learn more about the “**Breaking Changes, Bug Fixes, Features and Known Issues**” between your current version and the latest version you are trying to upgrade.

#### Upgrading to the latest version

The most recent version of Syncfusion JavaScript – EJ2 can be downloaded and installed by clicking on the “Latest Version: {Version}” link at the top of the Syncfusion JavaScript – EJ2 Control Panel.



You can also upgrade to the latest version just by downloading and installing the products you require from [this](#) link. The existing installed versions are not required to be uninstalled.

It is not required to install the Volume release before installing the Service Pack release. As releases for Volume and Service Packs work independently, you can install the latest version with major bug fixes directly.

#### Upgrade from trial version to license version

Uninstall the trial version and install the fully licensed installer from the [License and Downloads](#) section of our website to upgrade from the trial version.

Note: License key registration is not required for JavaScript, if you are using scripts (.js) and css files.

## Licensing

### Syncfusion Licensing Overview

We have introduced license key validation for Essential JS2 platforms from the 2022 Volume 1 release. This licensing key validation will enforce the developer to register the valid licensing key in an application while referring to any of the latest JavaScript packages, either from npm or CDN or build.

License key can be obtained from the [My Account >> License and downloads](#) section of the Syncfusion website. To obtain a license key, you will need to have an active trial or license or community license.

Before using any JavaScript controls, you must register the obtained license key in the application code. Otherwise you will get license validation error message in application as shown in below

This application was built using a trial version of Syncfusion Essential Studio. Please include a valid license to permanently remove this license validation message. You can also obtain a free 30 day evaluation license to temporarily remove this message during the evaluation period. Please refer to this [help topic](#) for more information.

#### Difference between unlock key and license key

Please note that this license key is different from the installer unlock key that you might have used in the past and needs to be separately generated from Syncfusion website.

- **Unlock Key** - Syncfusion Unlock Key is used to unlock the Syncfusion installers alone.
- **License Key** - Syncfusion License Key is a string that needs to be registered in your script to avoid licensing warning.

Refer to [this](#) KB article to know more about difference between the Syncfusion Unlock Key and the Syncfusion License Key.

#### Registering Syncfusion license keys in Build server

| Source of Syncfusion assemblies | Details | License Key needs to be registered? | Where to get license key from |

| ----- | ----- | ----- | ----- |

| **NuGet package** | If the Syncfusion assemblies used in Build Server were from the Syncfusion NuGet packages, then no need to install any Syncfusion installer. We can directly use the required Syncfusion NuGet packages at [nuget.org](https://nuget.org). <br><br>But, if using NuGet packages from the [nuget.org](https://nuget.org), then we should register the Syncfusion license key in the application. | Yes | Use any developer license to [generate](#) keys for Build Environments as well. |

| **Trial installer** | If the Syncfusion assemblies used in Build Server were from Trial Installer, we should register the license key in the application for the corresponding version and platforms, to avoid trial license warning. | Yes | Use any developer trial license to [generate](#) keys for Build Environments as well. |

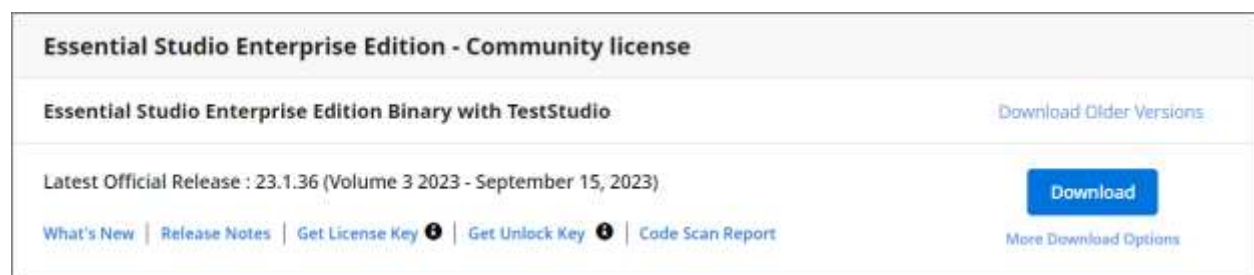
| **Licensed installer** | If the Syncfusion assemblies used in Build Server were from Licensed Installer, then there is no need to register the license keys.<br><br>You can [download](#) and [install](#) the licensed version of our installer. | No | Not applicable |

See also

- [How to generate Syncfusion Vue license key?](#)
- [How to register Syncfusion license key in Vue application?](#)
- [Licensing FAQ](#)

#### Generate Syncfusion Vue License key

License keys can be generated from the [License & Downloads](#) or [Trial & Downloads](#) section of the Syncfusion website.



\* Syncfusion license keys are **version and platform specific**. Refer to the [KB](#) to generate the license key for the required version and platform.

\* Refer to this [KB](#) to know which version of the Syncfusion license key should be used in the application.

#### Claim license key

Syncfusion License keys can also be generated from the “**Claim License Key**” page based on the trial or valid license associated with your Syncfusion account.

You can get the license key, based on license availability in your Syncfusion account.

### Active license

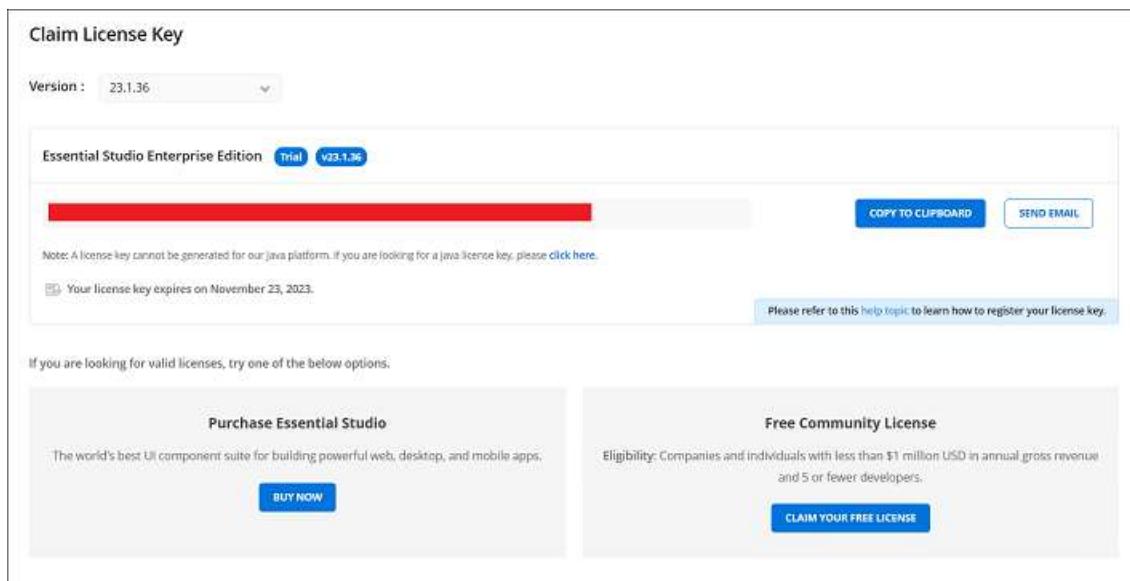
If you have a Syncfusion account associated with valid license, license key will be generated from claim license key page.



The screenshot shows the 'Claim License Key' interface. At the top, there's a 'Version' dropdown set to '23.1.36'. Below it, the product is identified as 'Essential Studio Enterprise Edition' with a 'v23.1.36' badge. A large red rectangular area represents the generated license key. To the right of this area are two buttons: 'COPY TO CLIPBOARD' and 'SEND EMAIL'. Below the key area, there is a note: 'For more details about this generated key, [click here](#).' and another note: 'Note: A license key cannot be generated for our Java platform. If you are looking for a Java license key, please [click here](#).' At the bottom right, a small blue box contains the text: 'Please refer to this [help topic](#) to learn how to register your license key.'

### Active trial

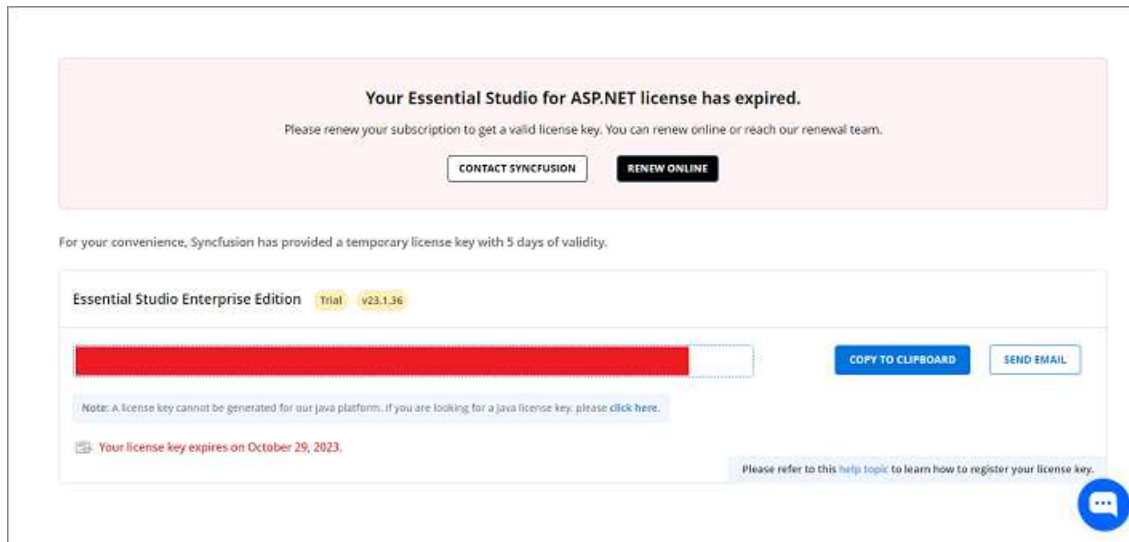
If you have a Syncfusion account associated with valid trial license, license key will be generated from claim license key page with expiry date.



This screenshot shows the 'Claim License Key' page for a trial license. The layout is similar to the active license page, but the product is 'Essential Studio Enterprise Edition' with a 'Trial' badge and 'v23.1.36' version. The license key area is represented by a red rectangle. The 'COPY TO CLIPBOARD' and 'SEND EMAIL' buttons are present. A note below the key states: 'Note: A license key cannot be generated for our Java platform. If you are looking for a Java license key, please [click here](#).' Below this, a message indicates: 'Your license key expires on November 23, 2023.' The same bottom-right note about the Java platform and help topic is present. At the bottom, there are two promotional boxes: 'Purchase Essential Studio' with a 'BUY NOW' button, and 'Free Community License' with a 'CLAIM YOUR FREE LICENSE' button. The 'Free Community License' box includes eligibility criteria: 'Eligibility: Companies and individuals with less than \$1 million USD in annual gross revenue and 5 or fewer developers.'

### Expired license

If you have a Syncfusion account with an expired license, your license subscription must be renewed in order to obtain a valid license key for the latest Essential Studio version. Meanwhile, a temporary license key with a five day validity period will be generated.



**Your Essential Studio for ASP.NET license has expired.**

Please renew your subscription to get a valid license key. You can renew online or reach our renewal team.

[CONTACT SYNCFUSION](#) [RENEW ONLINE](#)

For your convenience, Syncfusion has provided a temporary license key with 5 days of validity.

Essential Studio Enterprise Edition **Trial** v23.1.36

[Redacted License Key]

[COPY TO CLIPBOARD](#) [SEND EMAIL](#)

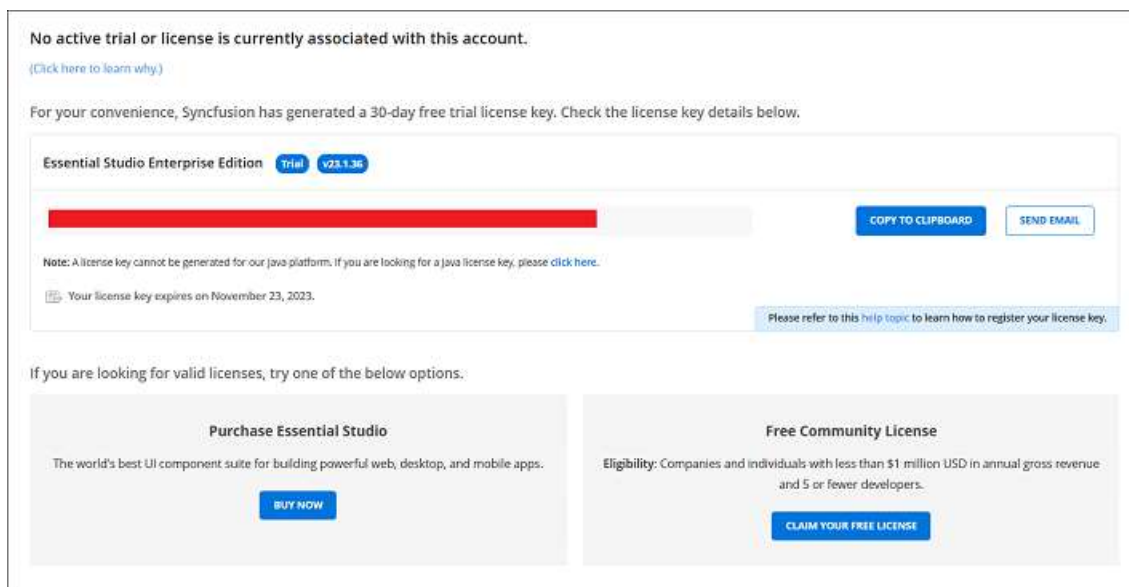
Note: A license key cannot be generated for our Java platform. If you are looking for a Java license key, please [click here](#).

Your license key expires on October 29, 2023.

Please refer to this [help topic](#) to learn how to register your license key.

### *No trial or no license or expired trial*

If the Syncfusion account is not associated with a trial, license, or expired trial, you can try to claim either a trial or a valid license from claim license page.



No active trial or license is currently associated with this account.

[\(Click here to learn why\)](#)

For your convenience, Syncfusion has generated a 30-day free trial license key. Check the license key details below.

Essential Studio Enterprise Edition **Trial** v23.1.36

[Redacted License Key]

[COPY TO CLIPBOARD](#) [SEND EMAIL](#)

Note: A license key cannot be generated for our Java platform. If you are looking for a Java license key, please [click here](#).

Your license key expires on November 23, 2023.

Please refer to this [help topic](#) to learn how to register your license key.

If you are looking for valid licenses, try one of the below options:

**Purchase Essential Studio**

The world's best UI component suite for building powerful web, desktop, and mobile apps.

[BUY NOW](#)

**Free Community License**

Eligibility: Companies and individuals with less than \$1 million USD in annual gross revenue and 5 or fewer developers.

[CLAIM YOUR FREE LICENSE](#)

See also

- [How to register Syncfusion license key in the application?](#)
- [Licensing FAQ](#)

### Register Syncfusion License key in Vue application

Syncfusion license key should be registered, if your project using Syncfusion Vue packages reference. The generated license key is a string that needs to be registered after any [Syncfusion Vue reference](#).

Note: Syncfusion license validation is done offline during application execution and does not require internet access. Apps registered with a Syncfusion license key can be deployed on any system that does not have an internet connection.



Generate the [Syncfusion license key](#) and register it in one of the following ways,

- [Register the license key in the project](#)
- [Register the license key in the Nuxt project](#)
- [Register the license key using the npx command](#)

#### Register Syncfusion license key in the project

Register the license key in the `main.js` file of the Vue project.

```
`ts
import { createApp } from 'vue'
import App from './App.vue'
import { registerLicense } from '@syncfusion/ej2-base';
// Registering Syncfusion license key
registerLicense('Replace your generated license key here');
createApp(App).mount('#app')
`
```

Only from 2022 Vol 1 v20.1.0.47, license key registration required for Essential JavaScript 2 products.

#### Register Syncfusion license key in the Nuxt project

Register the license key in the `index.js` file of the Nuxt project.

```
`ts
import Vue from 'vue'
import { registerLicense } from '@syncfusion/ej2-base';
// Registering Syncfusion license key.
registerLicense('Replace your generated license key here');
// Component: <ClientOnly>
Vue.component(ClientOnly.name, ClientOnly)
`
```

#### Register Syncfusion license key using the npx command

Register the Syncfusion license key through npx command in one of the following ways,

- [Register the license key with the license file](#)
- [Register the license key with the environment variable](#)

If both the license text file and the environment variable are used for license registration, priority is set to `syncfusion-license.txt` file. If you want to use the environment variable for license registration, then remove the license text file from the application.

#### *Register the license key with the license file*

The following steps show how to register the Syncfusion license key with the license text file.



- Create the `syncfusion-license.txt` file in the application root directory and paste the license key.
- Open the command prompt in the application root directory and activate the license key by using the below command,

```
npx syncfusion-license activate
```

- Once the Syncfusion license key is activated, the following console message will appear.

**License message:** `<br />` (INFO) Syncfusion License imported successfully.

- Remove the `.cache` folder from node modules in the application.
- Now run the application. If you are facing a license validation error, refer to this [link](#) to resolve it. Also, find the most frequent license registration questions from this [link](#).

If you don't want to use the license text file in the application, refer to this [link](#) to use an environment variable and register the Syncfusion license key.

*Register the license key with the environment variable*

You can set the environment variable as `SYNCFUSION_LICENSE` in the system and paste the license key as a value. It can be used in all applications on your machine.

The following steps show how to set environment variable in different operating systems and register the Syncfusion license key.

- Set the environment variable in different operating systems like below,

#### Windows

- Open the command prompt and use `setx` command to add the new environment variable.

```
setx SYNCFUSION_LICENSE "license key"
```

#### Mac

- Open the terminal and use the `env` command to view the variables list.
- You can set the environment variable by using below command,

```
echo 'export SYNCFUSIONLICENSE="license key"' >> ~/.bashprofile
```

- If you want to modify the environment variable in the bash profile. Use the below command,

```
nano .bash_profile
```

- Once modified the variable. Press `ctrl+x` to exit then `Y` and `Enter` button to save the changes.
- Close the terminal and open it again to see the environment variables changes using `env` command.

## Linux

- Open the terminal and use the `env` command to view the variables list.
- You can set or modify the [environment variable](#) by using below command,

```
export SYNCFUSION_LICENSE='license key'
```

- Once set the `SYNCFUSION_LICENSE` environment variable, restart the IDE or application terminal before using the license activation command.
- Open the command prompt in the application root directory and activate the license key by using the below command,

```
npx syncfusion-license activate
```

- Once the Syncfusion license key is activated, the following console message will appear.

**License message:** `<br />` (INFO) Syncfusion License imported successfully.

- Remove the `.cache` folder from node modules in the application.
- Now run the application. If you are facing a license validation error, refer to this [link](#) to resolve it. Also, find the most frequent license registration questions from this [link](#).

## Register Syncfusion license key in CI services

The following sections show how to use an environment variable in CI services.

### GitHub actions

- Create a [new Repository Secret](#) or an [Organization Secret](#). Set the name of the secret to `SYNCFUSION_LICENSE` and use the license key as a value.
- Add the Syncfusion license activation command after running npm install or yarn like below,

```
`bash
```

```
steps:
```

- name: Install node modules

```
run: npm install
```

- name: Activate Syncfusion License

```
run: npx syncfusion-license activate
```

```
env:
```

```
SYNCFUSIONLICENSE: ${{ secrets.SYNCFUSIONLICENSE }}
```

```
,
```

#### Azure Pipelines (YAML)

- Create a new [User-defined Variable](#) named `SYNCFUSION_LICENSE`. Use the license key as a value.
- Add the Syncfusion license activation command after running npm install or yarn like below,

The following example shows the syntax for Windows build agents.

```
`bash
```

```
pool:
```

```
vmImage: 'windows-latest'
```

```
steps:
```

- script: call npm install

```
displayName: 'Install node modules'
```

- script: call npx syncfusion-license activate

```
displayName: 'Activate Syncfusion License'
```

```
env:
```

```
SYNCFUSIONLICENSE: $(SYNCFUSIONLICENSE)
```

```
,
```

The following example shows the syntax for Linux build agents.

```
`bash
```

```
pool:
```

```
vmImage: 'ubuntu-latest'
```

```
steps:
```

- script: npm install

displayName: 'Install node modules'

- script: npx syncfusion-license activate

displayName: 'Activate Syncfusion License'

env:

SYNCFUSIONLICENSE: \$(SYNCFUSIONLICENSE)

,

#### Azure Pipelines (Classic)

- Create a new [User-defined Variable](#) named `SYNCFUSION_LICENSE`. Use the license key as a value.
- Add the Syncfusion license activation command after running npm install or yarn using bash task like below,

`bash

#### Activate the license

npx syncfusion-license activate

,

← Bash ⓘ

Type ⓘ

☐ File Path

☒ Inline

Script \*

```
# Activate the license  
npx syncfusion-license activate
```

Advanced ▼

About this task

Add

See also

- [Licensing FAQ](#)

### Syncfusion Licensing Errors

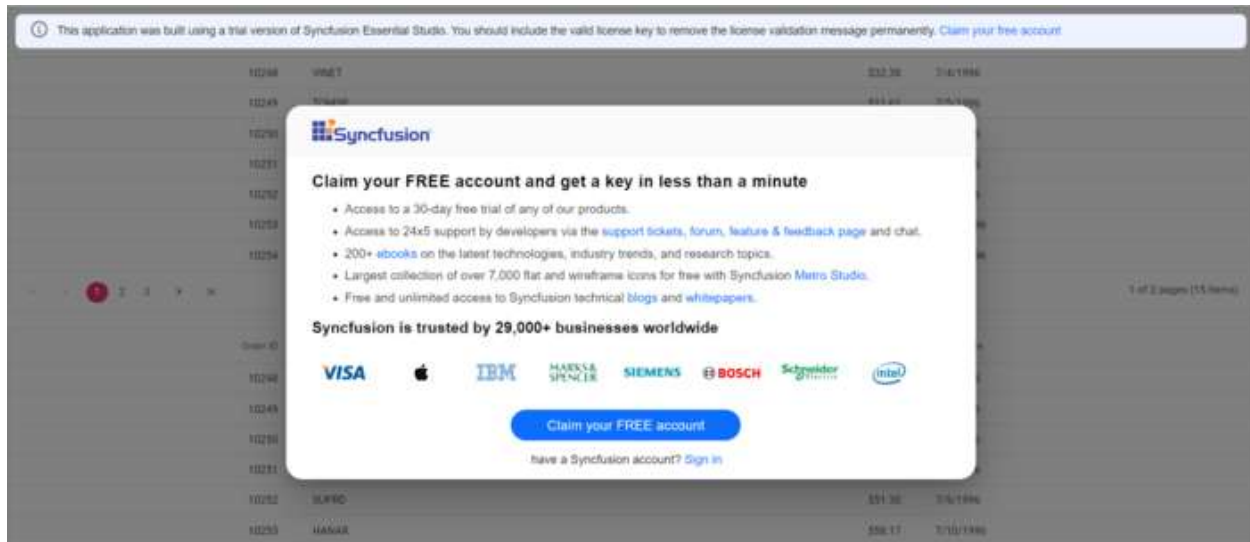
Licensing error popup is displayed with various messages under different circumstances. Here are some ways to resolve different issues.

#### Licensing errors

##### *License key not registered\trial expired*

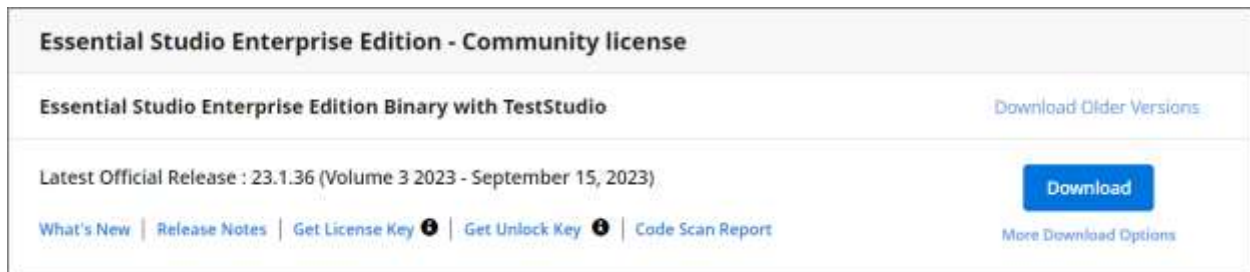
The following error message will be shown if a Syncfusion license key has not been registered in your application or if the trial key has expired after 30 days.

**Error message :** <br /> This application was built using a trial version of Syncfusion Essential Studio. You should include the valid license key to remove the license validation message permanently.



### Solution:

- If you use Vue components through syncfusion installer, you can choose from the options listed below
  1. If you **have a valid Syncfusion license**, you can **generate a license key for a specific version and product** from [this page](#).

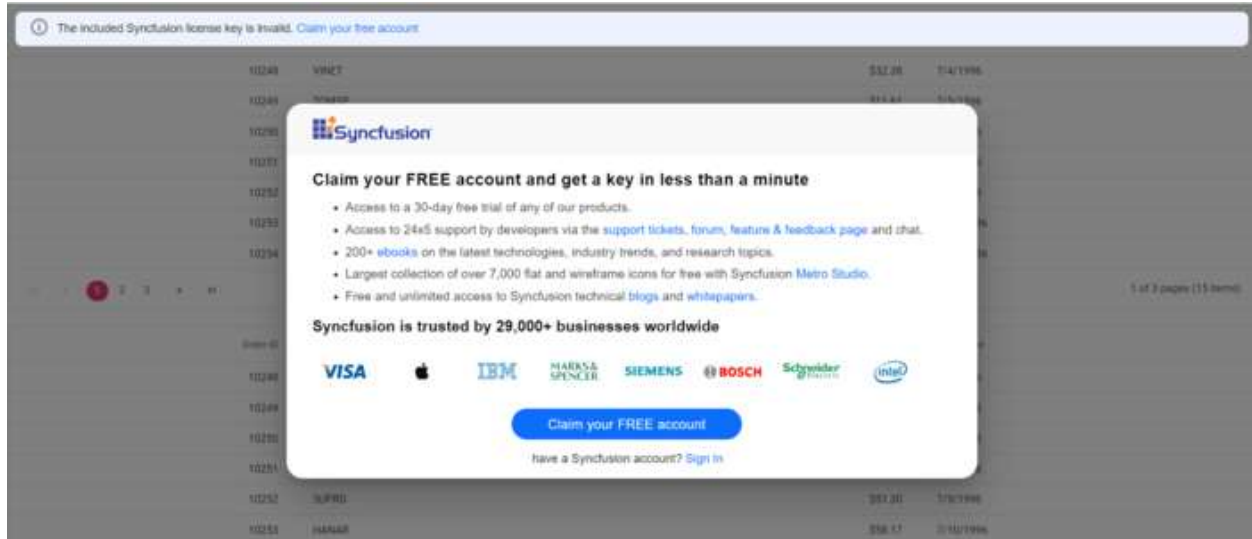


2. If you **have a Syncfusion account and an active trial**, you can **generate the trial license key for a specific version and platform** from [this page](#).
3. If you **have a Syncfusion account but no active trials**, [purchase a license](#) or [start your 30-day free trial](#). Then you can generate the trial license key for **a specific version and platform** from [this page](#).
4. If you **do not already have a Syncfusion account**, you can create one [here](#) and [purchase a license](#) or start your 30-day free trial. Then you can **generate the trial license key for a specific version and platform** from [this page](#).
5. Also, you can generate the license key from claim license key page by clicking the **“Claim your FREE account”** click from the licensing warning message. Refer to this [help topic](#) for more details.
  - In your application, register the generated license key. Please refer to this [help topic](#) for information on registering the license key.

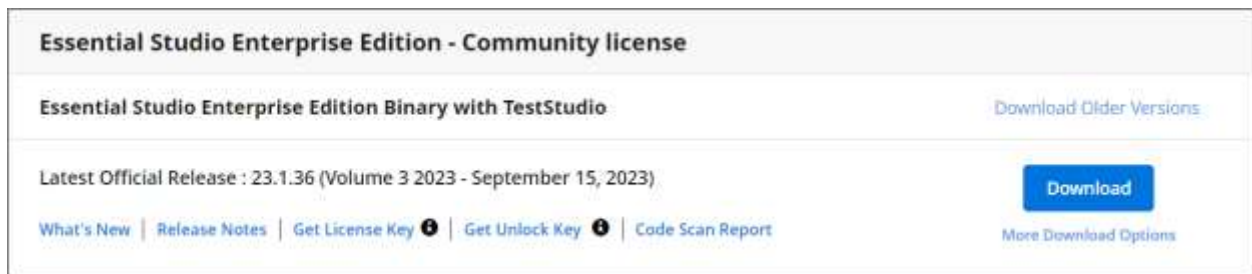
*Invalid key*

If the application is registered with an invalid key, another version of license key, or another platform's license key, the following error message will pop up when launching the application.

**Error Message:** The included Syncfusion license key is invalid.

**Solution:**

- If you use Vue components through syncfusion installer, you can choose from the options listed below
  1. If you have a valid Syncfusion license, you can **generate a license key for a specific version and product** from [this page](#).



2. If you have a Syncfusion account and an active trial, you can **generate the trial license key for a specific version and product** from [this page](#).
3. If you **have a Syncfusion account but no active trials**, [purchase a license](#) or [start your 30-day free trial](#). Then you can **generate the trial license key for a specific version and product** from [this page](#).
4. If you **do not already have a Syncfusion account**, you can create one here and [purchase a license](#) or [start your 30-day free trial](#). Then you can **generate the trial license key for a specific version and product** from [this page](#).
5. Also, you can generate the license key from claim license key page by clicking the **“Claim your FREE account”** click from the licensing warning message. Refer to this [help topic](#) for more details.

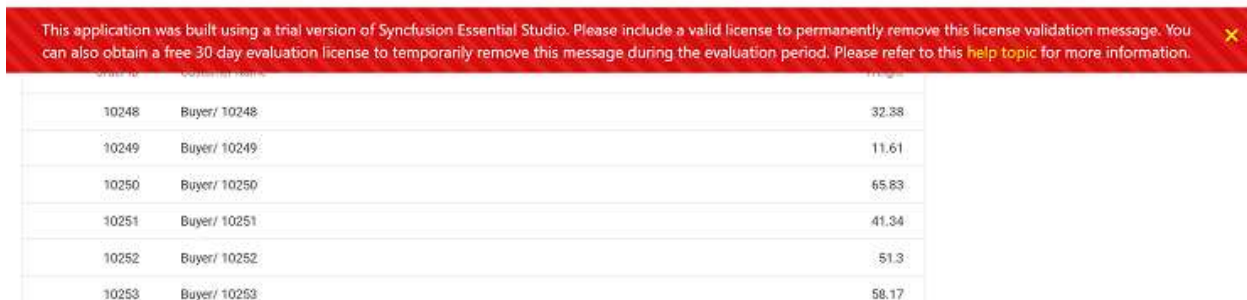
- In your application, register the generated license key. Please refer to this [help topic](#) for information on registering the license key.

Licensing errors from version 16.2.0\* to 20.3.0\*

#### *License key not registered*

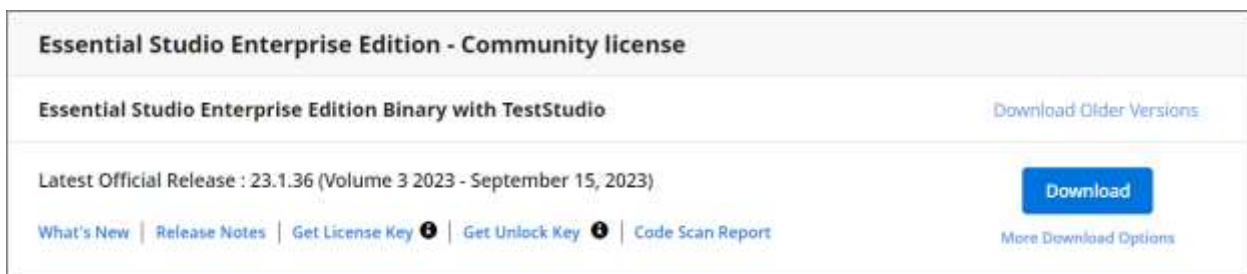
The following error message will be shown if a Syncfusion license key has not been registered in your application.

**Error message:** <br /> This application was built using a trial version of Syncfusion Essential Studio. Please include a valid license to permanently remove this license validation message. You can also obtain a free 30 day evaluation license to temporarily remove this message during the evaluation period. Please refer to this [help topic](#) for more information.



#### **Solution:**

- If you use Vue components through syncfusion installer, you can choose from the options listed below
  1. If you **have a valid Syncfusion license**, you can **generate a license key for a specific version and product** from [this page](#).



2. If you **have a Syncfusion account and an active trial**, you can **generate the trial license key for a specific version and platform** from [this page](#).
3. If you **have a Syncfusion account but no active trials**, [purchase a license](#) or [start your 30-day free trial](#). Then you can generate the trial license key for **a specific version and platform** from [this page](#).
4. If you **do not already have a Syncfusion account**, you can create one [here](#) and [purchase a license](#) or start your 30-day free trial. Then you can **generate the trial license key for a specific version and platform** from [this page](#).
  - In your application, register the generated license key. Please refer to this [help topic](#) for information on registering the license key.



*Invalid key*

If the application is registered with an invalid key, another version of license key, or another platform's license key, the following error message will pop up when launching the application.

**Error message:**   
The included Syncfusion license is invalid. Please refer to this [help topic](#) for more information.

The included Syncfusion license is invalid. Please refer to this <a href="#">help topic</a> for more information.		
Order ID	Customer Name	Freight
10248	Buyer/ 10248	32.38
10249	Buyer/ 10249	11.61
10250	Buyer/ 10250	65.83
10251	Buyer/ 10251	41.34
10252	Buyer/ 10252	51.3
10253	Buyer/ 10253	58.17

**Solution:**

- If you use Vue components through syncfusion installer, you can choose from the options listed below
  - If you have a valid Syncfusion license, you can **generate a license key for a specific version and product** from [this page](#).

### Essential Studio Enterprise Edition - Community license

Essential Studio Enterprise Edition Binary with TestStudio [Download Older Versions](#)

Latest Official Release : 23.1.36 (Volume 3 2023 - September 15, 2023)

[What's New](#) | 
 [Release Notes](#) | 
 [Get License Key](#) | 
 [Get Unlock Key](#) | 
 [Code Scan Report](#)

[Download](#)

[More Download Options](#)

- If you have a Syncfusion account and an active trial, you can **generate the trial license key for a specific version and product** from [this page](#).
- If you **have a Syncfusion account but no active trials**, [purchase a license](#) or [start your 30-day free trial](#). Then you can **generate the trial license key for a specific version and product** from [this page](#).
- If you **do not already have a Syncfusion account**, you can create one here and [purchase a license](#) or [start your 30-day free trial](#). Then you can **generate the trial license key for a specific version and product** from [this page](#).
  - In your application, register the generated license key. Please refer to this [help topic](#) for information on registering the license key.

*Trial expired*

The following error message will be shown if the trial key has expired after 30 days.

**Error message:**   
Your Syncfusion trial license has expired. Please refer to this [help topic](#) for more information.

Your Syncfusion trial license has expired. Please refer to this <a href="#">help topic</a> for more information.			✕
Order ID	Customer Name	Freight	
10248	Buyer/ 10248	32.38	
10249	Buyer/ 10249	11.61	
10250	Buyer/ 10250	65.83	
10251	Buyer/ 10251	41.34	
10252	Buyer/ 10252	51.3	
10253	Buyer/ 10253	58.17	

**Solution:** Purchase from [here](#) to get a valid Syncfusion license.

#### Platform Mismatch

If the application is registered with another platform's license key, the following error message will pop up when launching the application.

**Error message:** The included Syncfusion license is invalid (Platform mismatch). Please refer to this [help topic](#) for more information.

The included Syncfusion license is invalid (Platform mismatch). Please refer to this <a href="#">help topic</a> for more information.
---

#### Solution:

- License keys are version and product specific. So, if you use Vue components through syncfusion installer, you can choose from the options listed below
  - If you have a valid Syncfusion license, you can **generate a license key for a specific version and product** from [this page](#).

### Essential Studio Enterprise Edition - Community license

Essential Studio Enterprise Edition Binary with TestStudio [Download Older Versions](#)

Latest Official Release : 23.1.36 (Volume 3 2023 - September 15, 2023)

[What's New](#) | 
 [Release Notes](#) | 
 [Get License Key](#) | 
 [Get Unlock Key](#) | 
 [Code Scan Report](#)

[Download](#)

[More Download Options](#)

- If you have a Syncfusion account and an active trial, you can **generate the trial license key for a specific version and product** from [this page](#).
- If you **have a Syncfusion account but no active trials**, [purchase a license](#) or [start your 30-day free trial](#). Then you can **generate the trial license key for a specific version and product** from [this page](#).
  - In your application, register the generated license key. Please refer to this [help topic](#) for information on registering the license key.

#### Version Mismatch

If the application is registered with another version's license key, the following error message will pop up when launching the application.

**Error message:** <br /> The included Syncfusion license ({Registered Version}) is invalid for version {Required version}. Please refer to this [help topic](#) for more information.

The included Syncfusion license (v19.1.0.44) is invalid for version 20.1.x. Please refer to this <a href="#">help topic</a> for more information.		
Order ID	Customer Name	Freight
10248	Buyer/ 10248	32.38
10249	Buyer/ 10249	11.61
10250	Buyer/ 10250	65.82
10251	Buyer/ 10251	41.34
10252	Buyer/ 10252	51.3

**Solution:**

- License keys are version and product specific. So, if you use Vue components through syncfusion installer, you can choose from the options listed below
  - If you have a valid Syncfusion license, you can **generate a license key for a specific version and product** from [this page](#).

### Essential Studio Enterprise Edition - Community license

Essential Studio Enterprise Edition Binary with TestStudio [Download Older Versions](#)

Latest Official Release : 23.1.36 (Volume 3 2023 - September 15, 2023)

[What's New](#) | 
 [Release Notes](#) | 
 [Get License Key](#) | 
 [Get Unlock Key](#) | 
 [Code Scan Report](#)

[Download](#)

[More Download Options](#)

- If you have a Syncfusion account and an active trial, you can **generate the trial license key for a specific version and product** from [this page](#).
- If you **have a Syncfusion account but no active trials**, [purchase a license](#) or [start your 30-day free trial](#). Then you can **generate the trial license key for a specific version and product** from [this page](#).
  - In your application, register the generated license key. Please refer to this [help topic](#) for information on registering the license key.

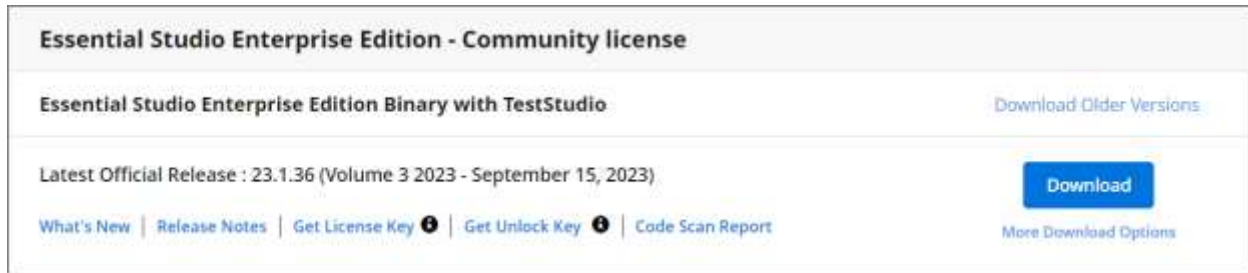
*Invalid license key structure using the npx command*

If you are using `npx syncfusion-license activate` command with an invalid license key structure, the following console error message will appear.

**Error message:** <br /> (Error) License key is not valid. Please refer to this [help topic](#) for more information.

**Solution:**

- If you use Vue components through syncfusion installer, you can choose from the options listed below
  - If you have a valid Syncfusion license, you can **generate a license key for a specific version and product** from [this page](#).



2. If you have a Syncfusion account and an active trial, you can **generate the trial license key for a specific version and product** from [this page](#).
3. If you **have a Syncfusion account but no active trials**, [purchase a license](#) or [start your 30-day free trial](#). Then you can **generate the trial license key for a specific version and product** from [this page](#).
4. If you **do not already have a Syncfusion account**, you can create one here and [purchase a license](#) or [start your 30-day free trial](#). Then you can **generate the trial license key for a specific version and product** from [this page](#).
  - In your application, register the generated license key using the npx command. Please refer to this [help topic](#) for information on registering the license key.

## Licensing FAQ

### Is an internet connection required for license validation

No, Internet connection is not required for the Syncfusion Essential Studio license validation. The Syncfusion license validation is done offline during application execution. Apps registered with a Syncfusion license key can be deployed on any system that does not have an internet connection.

### Upgrade from the trial version after purchasing a license

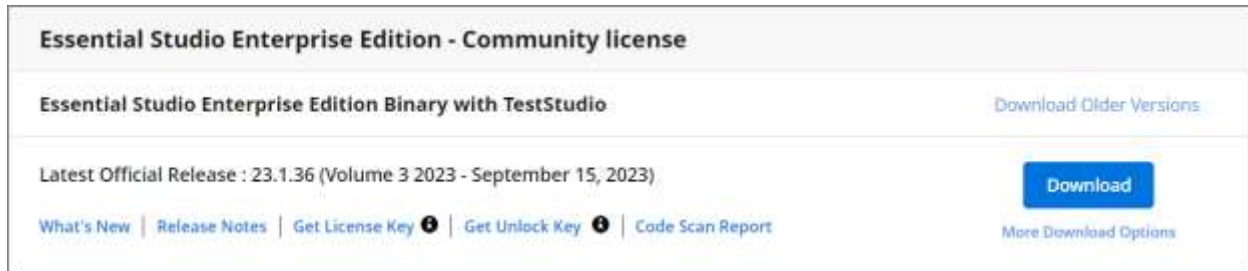
To upgrade from the trial version, there are two possible solutions:

- Uninstall the trial version and install the fully licensed build from the [License & Downloads](#) section of the Syncfusion website.
- If you are using Syncfusion controls from the [npm](#), replace the currently used trial license key with a paid license key that can be generated from the [License & Downloads](#) section of Syncfusion website. Refer to [this](#) topic for more information regarding registering the license in the application.

The license registration is not required if you reference Syncfusion scripts from the Licensed installer. These licensing changes apply to all evaluators who refer to the Syncfusion scripts from the evaluation installer and those who use the Syncfusion NuGet packages from [nuget.org](#).

### Where can I get a license key

License keys can be generated from the [License & Downloads](#) or the [Trial & Downloads](#) section of the Syncfusion website.



The Syncfusion license keys are the **version and platform-specific**, refer to the [KB](#) to generate the license key for the required version and platform. Also, refer to this [KB](#) to know which version of the Syncfusion license key should be used in the application.

While using the ASP.NET Core controls with the Javascript(ES5) components, you need to register the license key in both the Javascript(ES5) and the [ASP.NET core](#). Since the license is validated at the client side for Javascript(ES5) components and server-side for the ASP.NET core components.

#### [Will the registered license key expire](#)

No, the Syncfusion license keys won't expire for a particular version and you can continue to use it. So, you won't face any problems on the live site. If you have used the trial key, it will expire in 30 days and we don't recommend using it in production.

If you upgrade to newer versions of the Syncfusion packages, you have to generate new license keys and use them.

#### [When to generate new license key while upgrading](#)

You don't have to generate and change license keys for minor version upgrades. If you upgrade from one major version and another major version, you have to generate new license keys and register in the application.

For example,

- If you upgrade to weekly releases or the SP release in the same major version, you don't have to change license such as if you upgrade from the 20.1.47 version to 20.1.\* you don't have to change the license keys.
- If you upgrade from one major version to another major version, you have to generate new license keys for the latest version and change in the application, such as if you upgrade from the 20.1. version to 20.2., you have to generate new license keys for the latest version and change in the application.

#### [License registration for multiple developers on your project](#)

Syncfusion license key is a version based and it's not based on the developer. You don't have to register different keys for each developer. Just register one valid license key when developing and publishing the software.

#### [Can I use the same key for all the web apps under the project](#)

Yes, you can use the same license key for all the web apps.

#### [Does the license registration access any resources or data](#)

No, the license registration doesn't access any data or resources.

License & Downloads shows the "Essential Studio Enterprise Edition Binary with Test Studio" and the "Project License". Which license to use

Use any licenses shown on the [accounts & downloads](#) page. It shows two licenses because if you are part of your company's enterprise portal Global license and an individual license is also assigned to your account, on your account & downloads page, the individual license and your enterprise portal Global license are shown.

The image displays two screenshots of the Syncfusion licensing interface. Both screenshots show the 'Essential Studio Enterprise Edition Binary with TestStudio' section. The top screenshot includes a 'Download' button and a license expiration notice: 'Your license expires on June 30, 2023.' The bottom screenshot shows the 'Project License' section with the same 'Download' button and license expiration notice. Both screenshots also include links for 'What's New', 'Release Notes', 'Get License Key', 'Get Unlock Key', and 'Code Scan Report'.

Refer to the [KB](#) article which explains the Licenses offered by Syncfusion.

If I registered the license key in both the application and the license text file  
The application registered license key is set priority and used for license validation.

Potential causes of licensing errors in applications.

Below are the possible reasons that could lead to a license error within the application:

- The application may have a license issue due to duplicate Syncfusion packages.
- An invalid license issue may occur because of Syncfusion packages being referred with multiple versions.
- Registering the license key of a different version than the referred Syncfusion package version in the application can also cause licensing errors.

#### *License issue due to duplicate Syncfusion packages in the application*

One of the possible cases on experiencing license issues in your application is due to duplicate packages exists after upgrading packages to next or latest version. To remove the duplicate packages follow the below steps.

- Delete the `@Syncfusion` folder from `node_modules` and `package-lock.json` file from app root folder.
- Clear the npm `.cache` by running the command `npm cache clean --force` or you can directly delete the file present in the application.



- It is recommended to update all Syncfusion components in the package.json file to the **same major version**. This ensures consistency and compatibility across the project. For instance, if the updated version being utilized is **v20.4.XX**, it is advised to upgrade all components to the **same version**.
- Run **npm install** Command.

#### *Invalid license issue because of Syncfusion packages referred with multiple version*

It is essential to ensure that all the components used in a project are compatible and work seamlessly together. One common issue that arises in such scenarios is **version mismatch**. Version mismatch occurs when different components have different major versions, leading to compatibility issues and difficulties in license registration.

For example, consider a situation where one component in the project has a version of **v20.1.XX**, while another component has a version of **v20.2.XX**. When such components are used together, a **version mismatch** occurs, leading to license errors. To avoid version mismatch and ensure smooth functioning of the project, it is crucial to use the **same major version** for all the Syncfusion components. This will ensure compatibility and prevent any licensing issues that may arise due to version incompatibility.

#### *Registering the license key of a different version than the referred Syncfusion package version in the application*

When developing an application with Syncfusion packages, it is important to register the appropriate license key that matches the version of the package installed. Failure to do so may result in license errors within the application.

For instance, if you are using a component version labeled as **(v20.4.XX)**, it is essential to register the license key generated **specifically** for that version. By doing so, it ensures the smooth functioning of the controls and provides access to all features and functionality without encountering any license validation errors.

## Appearance

### Theme in Vue Appearance component

The Syncfusion Vue library has provided the below list of in-built themes,

Theme	Style Sheet Name
-----	-----
Material 3	material3.css
Material 3 Dark	material3-dark.css
Bootstrap 5	bootstrap5.css
Bootstrap 5 Dark	bootstrap5-dark.css
Fluent	fluent.css
Fluent Dark	fluent-dark.css
Google's Material	material.css
Google's Material-Dark	material-dark.css
Tailwind CSS	tailwind.css

|TailwindDark CSS | tailwind-dark.css |

|Microsoft Office Fabric | fabric.css |

|Microsoft Office Fabric Dark | fabric-dark.css |

|High Contrast | highcontrast.css |

The Syncfusion Bootstrap theme is designed based on **Bootstrap v3**, however, it can be compatible with **Bootstrap v4** applications. In addition to these four built-in themes, [ThemeStudio](#) provides support for the Fusion Theme that can only be downloaded from [ThemeStudio](#).

Themes are shipped as individual and combined CSS files. Combined CSS file can be referred from the npm package `@syncfusion/ej2` and individual CSS files are available within same component repository's `style` folder. In ej2 npm packages, we have shipped both CSS and SCSS files for all components.

Referring All components CSS

`

```
@import "../nodemodules/@syncfusion/ej2/<themenam>.css";
```

`

Referring All components SCSS

```
`scss
```

```
@import "ej2/<theme_name>.scss";
```

`

[Referring individual control theme](#)

We can get the individual theme from the individual npm package or from an overall ej2 npm package.

Referring individual control from individual package

```
`scss
```

```
@import "<dependent-package>/<dependent-control>/<theme_name>.scss";
```

```
@import "ej2-buttons/styles/button/<theme_name>.scss";
```

`

**Example:**

```
`scss
```

```
@import "ej2-base/styles/material.scss";
```

```
@import "ej2-buttons/styles/button/material.scss";
```

`

`ej2-base` is common dependent package for all Syncfusion JavaScript control styles. So, it needs first to be added in the import statement.

Referring individual control from ej2 package

```
`scss
```



```
@import "ej2/<dependent-control>/<theme_name>.scss";
@import "ej2/button/<theme_name>.scss";
`
```

**Example:**

```
`scss
@import "ej2/base/material.scss";
@import "ej2/button/material.scss";
`
```

The individual control theme will not include its dependent control styles. I.e. The dependent controls theme should be added before adding the individual control themes.

*Advantages of using the individual components theme*

- Reducing the page load time of the application.
- Reducing the bundle size of the application.
- Avoids using the unused components CSS.

*Common variables*

The below list of common variables used in the Syncfusion Vue library themes for all components. You can change these variables to customize the corresponding theme.

*Syncfusion Material 3 theme*

Name	Value (Default Theme)	Value (Dark Theme)
--color-sf-black	rgb(0,0,0)	rgb(0,0,0)
--color-sf-white	rgb(255,255,255)	rgb(255,255,255)
--color-sf-primary	rgb(103, 80, 164)	rgb(208, 188, 255)
--color-sf-primary-container	rgb(234, 221, 255)	rgb(79, 55, 139)
--color-sf-on-primary	rgb(255, 255, 255)	rgb(55, 30, 115)
--color-sf-on-primary-container	rgb(33, 0, 94)	rgb(234, 221, 255)
--color-sf-surface	rgb(255, 255, 255)	rgb(28, 27, 31)
--color-sf-surface-variant	rgb(231, 224, 236)	rgb(73, 69, 79)
--color-sf-on-surface	rgb(28, 27, 31)	rgb(230, 225, 229)
--color-sf-on-surface-variant	rgb(73, 69, 78)	rgb(202, 196, 208)

Name	Value (Default Theme)	Value (Dark Theme)
--color-sf-secondary	rgb(98, 91, 113)	rgb(204, 194, 220)
--color-sf-secondary-container	rgb(232, 222, 248)	rgb(74, 68, 88)
--color-sf-on-secondary	rgb(255, 255, 255)	rgb(51, 45, 65)
--color-sf-on-secondary-container	rgb(30, 25, 43)	rgb(232, 222, 248)
--color-sf-tertiary	rgb(125, 82, 96)	rgb(239, 184, 200)
--color-sf-tertiary-container	rgb(255, 216, 228)	rgb(99, 59, 72)
--color-sf-on-tertiary	rgb(255, 255, 255)	rgb(73, 37, 50)
--color-sf-on-tertiary-containe	rgb(55, 11, 30)	rgb(255, 216, 228)
--color-sf-background	rgb(255, 255, 255)	rgb(28, 27, 31)
--color-sf-on-background	rgb(28, 27, 31)	rgb(230, 225, 229)
--color-sf-outline	rgb(121, 116, 126)	rgb(147, 143, 153)
--color-sf-outline-variant	rgb(196, 199, 197)	rgb(68, 71, 70)
--color-sf-shadow	rgb(0, 0, 0)	rgb(0, 0, 0)
--color-sf-surface-tint-color	rgb(103, 80, 164)	rgb(208, 188, 255)
--color-sf-inverse-surface	rgb(49, 48, 51)	rgb(230, 225, 229)
--color-sf-inverse-on-surface	rgb(244, 239, 244)	rgb(49, 48, 51)
--color-sf-inverse-primary	rgb(208, 188, 255)	rgb(103, 80, 164)
--color-sf-scrim	rgb(0, 0, 0)	rgb(0, 0, 0)
--color-sf-error	rgb(179, 38, 30)	rgb(242, 184, 181)
--color-sf-error-container	rgb(249, 222, 220)	rgb(140, 29, 24)
--color-sf-on-error	rgb(255, 250, 250)	rgb(96, 20, 16)
--color-sf-on-error-container	rgb(65, 14, 11)	rgb(249, 222, 220)

Name	Value (Default Theme)	Value (Dark Theme)
--color-sf-success	rgb(32, 81, 7)	rgb(83, 202, 23)
--color-sf-success-container	rgb(209, 255, 186)	rgb(22, 62, 2)
--color-sf-on-success	rgb(244, 255, 239)	rgb(13, 39, 0)
--color-sf-on-success-container	rgb(13, 39, 0)	rgb(183, 250, 150)
--color-sf-info	rgb(1, 87, 155)	rgb(71, 172, 251)
--color-sf-info-container	rgb(233, 245, 255)	rgb(0, 67, 120)
--color-sf-on-info	rgb(250, 253, 255)	rgb(0, 51, 91)
--color-sf-on-info-container	rgb(0, 51, 91)	rgb(173, 219, 255)
--color-sf-warning	rgb(145, 76, 0)	rgb(245, 180, 130)
--color-sf-warning-container	rgb(254, 236, 222)	rgb(123, 65, 0)
--color-sf-on-warning	rgb(255, 255, 255)	rgb(99, 52, 0)
--color-sf-on-warning-container	rgb(47, 21, 0)	rgb(255, 220, 193)

*Bootstrap 5 theme*

Name	Value (Default Theme)	Value (Dark Theme)
\$black	#000	#000
\$white	#fff	#fff
\$gray-100	#f8f9fa	#f8f9fa
\$gray-200	#e9ecef	#e9ecef
\$gray-300	#dee2e6	#dee2e6
\$gray-400	#ced4da	#ced4da
\$gray-500	#adb5bd	#adb5bd
\$gray-600	#6c757d	#6c757d

Name	Value (Default Theme)	Value (Dark Theme)
\$gray-700	#495057	#495057
\$gray-800	#343a40	#343a40
\$gray-900	#212529	#212529
\$blue	#0d6efd	#0d6efd
\$indigo	#6610f2	#6610f2
\$purple	#6f42c1	#6f42c1
\$pink	#d63384	#d63384
\$red	#dc3545	#dc3545
\$orange	#fd7e14	#fd7e14
\$yellow	#ffc107	#ffc107
\$green	#198754	#198754
\$teal	#20c997	#20c997
\$cyan	#0dcaf0	#0dcaf0

*Fluent theme*

Name	Value (Default Theme)	Value (Dark Theme)
\$black	#000	#000
\$white	#fff	#fff
\$gray220	#11100f	#11100f
\$gray210	#161514	#161514
\$gray200	#1b1a19	#1b1a19
\$gray190	#201f1e	#201f1e
\$gray180	#252423	#252423

Name	Value (Default Theme)	Value (Dark Theme)
\$gray170	#292827	#292827
\$gray160	#323130	#323130
\$gray150	#3b3a39	#3b3a39
\$gray140	#484644	#484644
\$gray130	#605e5c	#605e5c
\$gray120	#797775	#797775
\$gray110	#8a8886	#8a8886
\$gray100	#979593	#979593
\$gray90	#a19f9d	#a19f9d
\$gray80	#b3b0ad	#b3b0ad
\$gray70	#bebbb8	#bebbb8
\$gray60	#c8c6c4	#c8c6c4
\$gray50	#d2d0ce	#d2d0ce
\$gray40	#e1dfdd	#e1dfdd
\$gray30	#edebe9	#edebe9
\$gray20	#f3f2f1	#f3f2f1
\$gray10	#faf9f8	#faf9f8
\$cyanblue10	#0078d4	#0078d4
\$red10	#d13438	#d13438
\$orange20	#ca5010	#ca5010
\$green20	#0b6a0b	#0b6a0b
\$cyan20	#038387	#038387

*Material theme*

Name	Value (Default Theme)	Value (Dark Theme)
\$accent	#e3165b	#ff80ab
\$accent-font	#fff	#000
\$primary	#3f51b5	#3f51b5
\$primary-50	#e8eaf6	#e8eaf6
\$primary-100	#c5cae9	#c5cae9
\$primary-200	#9fa8da	#9fa8da
\$primary-300	#7986cb	#7986cb
\$primary-font	#fff	#fff
\$primary-50-font	#000	#000
\$primary-100-font	#000	#000
\$primary-200-font	#000	#000
\$primary-300-font	#fff	#fff
\$grey-white	#fff	#fff
\$grey-black	#000	#000
\$grey-50	#fafafa	#fafafa
\$grey-100	#f5f5f5	#f5f5f5
\$grey-200	#eee	#eee
\$grey-300	#e0e0e0	#e0e0e0
\$grey-400	#bdbdbd	#bdbdbd
\$grey-500	#9e9e9e	#9e9e9e
\$grey-600	#757575	#757575

Name	Value (Default Theme)	Value (Dark Theme)
\$grey-700	#616161	#616161
\$grey-800	#424242	#424242
\$grey-900	#212121	#212121
\$grey-dark	#303030	#303030
\$grey-light-font	#000	#000
\$grey-dark-font	#fff	#fff
\$base-font	#000	#000
\$error-font	#f44336	#ff6652
\$success-bg		#4caf50
\$error-bg		#ff6652
\$warning-bg		#ff9800
\$info-bg		#03a9f4
\$message-font		#fff
\$success-font		#4caf50
\$warning-font		#ff9800
\$info-font		#03a9f4

*Tailwind CSS theme*

Name	Value (Default Theme)	Value (Dark Theme)
\$black	#000	#000
\$white	#fff	#fff
\$transparent	transparent	transparent
\$cool-gray-50	#f9fafb	#f9fafb

Name	Value (Default Theme)	Value (Dark Theme)
\$cool-gray-100	#f3f4f6	#f3f4f6
\$cool-gray-200	#e5e7eb	#e5e7eb
\$cool-gray-300	#d1d5db	#d1d5db
\$cool-gray-400	#9ca3af	#9ca3af
\$cool-gray-500	#6b7280	#6b7280
\$cool-gray-600	#4b5563	#4b5563
\$cool-gray-700	#374151	#374151
\$cool-gray-800	#1f2937	#1f2937
\$cool-gray-900	#111827	#111827
\$red-100	#fee2e2	#fee2e2
\$red-400	#f87171	#f87171
\$red-500	#ef4444	#ef4444
\$red-600	#dc2626	#dc2626
\$red-800	#991b1b	#991b1b
\$green-100	#dcfce7	#dcfce7
\$green-500	#22c55e	#22c55e
\$green-600	#16a34a	#16a34a
\$green-700	#15803d	#15803d
\$orange-100	#ffedd5	#ffedd5
\$orange-500	#f97316	#f97316
\$orange-600	#ea580c	#ea580c
\$orange-700	#c2410c	#c2410c



Name	Value (Default Theme)	Value (Dark Theme)
\$orange-800	#9a3412	#9a3412
\$cyan-300	#67e8f9	#67e8f9
\$cyan-400	#22d3ee	#22d3ee
\$cyan-500	#06b6d4	#06b6d4
\$cyan-600	#0891b2	#0891b2
\$cyan-800	#155e75	#155e75
\$indigo-50	#eef2ff	
\$indigo-100	#e0e7ff	
\$indigo-200	#c7d2fe	
\$indigo-300	#a5b4fc	
\$indigo-400	#818cf8	
\$indigo-500	#6366f1	
\$indigo-600	#4f46e5	
\$indigo-700	#4338ca	
\$indigo-800	#3730a3	
\$indigo-900	#312e81	
\$green-400		#4ade80
\$light-blue-50		#f0f9ff
\$light-blue-100		#e0f2fe
\$light-blue-400		#38bdf8
\$light-blue-500		#0ea5e9
\$light-blue-600		#0284c7

Name	Value (Default Theme)	Value (Dark Theme)
\$light-blue-700		#0369a1
\$light-blue-800		#075985

*Microsoft Office Fabric theme*

Name	Value (Default Theme)	Value (Dark Theme)
\$theme-primary	#0078d6	#0074cc
\$theme-dark-alt	darken(\$theme-primary, 3%)	darken(\$theme-primary, 3%)
\$theme-dark	darken(\$theme-primary, 10%)	darken(\$theme-primary, 6%)
\$theme-darker	darken(\$theme-primary, 18%)	darken(\$theme-primary, 10%)
\$theme-secondary	lighten(\$theme-primary, 3%)	lighten(\$theme-primary, 3%)
\$theme-tertiary	lighten(\$theme-primary, 21%)	lighten(\$theme-primary, 21%)
\$theme-light	lighten(\$theme-primary, 44%)	lighten(\$theme-primary, 44%)
\$theme-lighter	lighten(\$theme-primary, 49%)	lighten(\$theme-primary, 49%)
\$theme-lighter-alt	lighten(\$theme-primary, 55%)	lighten(\$theme-primary, 55%)
\$neutral-white	#fff	#201f1f
\$neutral-lighter-alt	#f8f8f8	#282727
\$neutral-lighter	#f4f4f4	#333232
\$neutral-light	#eaeaea	#414040
\$neutral-quintenaryalt	#dadada	#4a4848
\$neutral-quintenary	#d0d0d0	#514f4f
\$neutral-tertiary-alt	#c8c8c8	#6f6c6c
\$neutral-tertiary	#a6a6a6	#9a9a9a
\$neutral-secondary-alt	#767676	#c8c8c8

Name	Value (Default Theme)	Value (Dark Theme)
\$neutral-secondary	#666	#dadada
\$neutral-primary	#333	#fff
\$neutral-dark	#212121	#f4f4f4
\$neutral-black	#000	#f8f8f8
\$alert-bg	#deecf9	#bf7500
\$error-bg	#fde7e9	#cd2a19
\$success-bg	#dff6dd	#37844d
\$theme-dark-font	#fff	#fff
\$theme-primary-font	#fff	#fff
\$theme-light-font	#333	#000
\$neutral-light-font	#333	#dadada
\$neutral-light-fontalt	#000	#fff
\$grey-dark-font	#fff	#000
\$base-font	#333	#dadada
\$message-font	#333	#fff
\$alert-font	#d83b01	#ff9d48
\$error-font	#a80000	#ff5f5f
\$success-font	#107c10	#8eff8d
\$info-bg		#1e79cb
\$info-font		#62cfff

*High Contrast theme*

Name	Value
\$selection-bg	#ffd939
\$selection-font	#000
\$selection-border	#ffd939
\$hover-bg	#685708
\$hover-font	#fff
\$hover-border	#fff
\$border-default	#969696
\$border-alt	#757575
\$border-fg	#fff
\$border-fg-alt	#ffd939
\$bg-base-0	#000
\$bg-base-5	#0d0d0d
\$bg-base-10	#1a1a1a
\$bg-base-15	#262626
\$bg-base-20	#333
\$bg-base-75	#bfbfbf
\$bg-base-100	#fff
\$header-font	#ffd939
\$header-font-alt	#fff
\$content-font	#fff
\$content-font-alt	#969696

Name	Value
\$link	#8a8aff
\$invert-font	#000
\$success-bg	#166600
\$error-bg	#b30900
\$message-font	#fff
\$alert-bg	#944000
\$info-bg	#0056b3
\$success-alt	#2bc700
\$error-alt	#ff6161
\$alert-alt	#ff7d1a
\$info-alt	#66b0ff
\$disable	#757575

### Size modes in Vue Appearance component

An application that is designed to be accessed through a web browser on various devices, including desktop computers and mobile devices, may have a distinct layout or user interface on a mobile device compared to a desktop computer to better suit the smaller screen size.

Syncfusion Vue components support both touch (bigger) and normal size modes. Touch mode creates a responsive design for mobile devices by adding the `e-bigger` class, which enhances interactions, visibility, and the overall experience.

#### Size mode for application

The user can enable touch mode (bigger) for the entire application by adding the `e-bigger` class to the `body` element in the `index.html` file as follows:

```
`  
<body className="e-bigger">  
...  
</body>  
`
```

### Size mode for a component

The user can enable touch mode (bigger) for a component by adding the `e-bigger` class to the `div` element that contains the component. Another way of enabling touch mode is by adding the `e-bigger` class using the available `cssClass` property of the component.

#### APP.VUE

```
<template>
  <div id='container' class="e-bigger">
    <ejs-button>Button</ejs-button>
  </div>
</template>
<script>
import Vue from "vue";
import { ButtonPlugin } from '@syncfusion/ej2-vue-buttons';
Vue.use(ButtonPlugin);
export default {
}
</script>
<style>
@import '../node_modules/@syncfusion/ej2-base/styles/material.css';
@import '../node_modules/@syncfusion/ej2-buttons/styles/material.css';
</style>
```

{% previewsample "page.domainurl/code-snippet/common/size-modes-cs1" %}

### Change size mode for application at runtime

The user can change the size mode of the application between touch and normal (mouse) mode at runtime by adding and removing the `e-bigger` class. The following steps explain how to change the size mode of an application at runtime:

#### APP.VUE

```
<template>
  <div id='container'>
    <div>
      <ejs-button content="Touch Mode" v-
on:click.native='touchClick'>Button</ejs-button>
      <ejs-button content="Mouse Mode" v-
on:click.native='mouseClick'>Button</ejs-button>
    </div>
    <div class="control">
      <ejs-calendar ></ejs-calendar>
    </div>
  </div>
</template>
<script>
import Vue from "vue";
import { ButtonPlugin } from '@syncfusion/ej2-vue-buttons';
import { CalendarPlugin } from '@syncfusion/ej2-vue-calendars';
Vue.use(ButtonPlugin);
Vue.use(CalendarPlugin);
export default {
  methods : {
    touchClick: function(event) {
```

```

        document.body.classList.add('e-bigger');
    },
    mouseClick: function(event) {
        document.body.classList.remove('e-bigger');
    }
}
}
</script>
<style>
@import '../node_modules/@syncfusion/ej2-base/styles/material.css';
@import '../node_modules/@syncfusion/ej2-buttons/styles/material.css';
@import "../node_modules/@syncfusion/ej2-vue-calendars/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/common/size-modes-cs2" %}

### Change size mode for a component at runtime

The user can change the size mode of a component between touch and normal (mouse) mode at runtime by setting the `e-bigger` CSS class.

#### APP.VUE

```

<template>
  <div id='container'>
    <div>
      <ejs-button content="Touch Mode" v-
on:click.native='touchClick'>Button</ejs-button>
      <ejs-button content="Mouse Mode" v-
on:click.native='mouseClick'>Button</ejs-button>
    </div>
    <div class="control">
      <ejs-calendar ></ejs-calendar>
    </div>
  </div>
</template>
<script>
import Vue from "vue";
import { ButtonPlugin } from '@syncfusion/ej2-vue-buttons';
import { CalendarPlugin } from '@syncfusion/ej2-vue-calendars';
Vue.use(ButtonPlugin);
Vue.use(CalendarPlugin);
export default {
  methods : {
    touchClick: function(event) {
      var controls = document.querySelectorAll('.control');
      for (var index = 0; index < controls.length; index++) {
        controls[index].classList.add('e-bigger');
      }
    },
    mouseClick: function(event) {
      var controls = document.querySelectorAll('.control');
      for (var index = 0; index < controls.length; index++) {
        controls[index].classList.remove('e-bigger');
      }
    }
  }
}

```

```

    }
  }
</script>
<style>
@import '../node_modules/@syncfusion/ej2-base/styles/material.css';
@import '../node_modules/@syncfusion/ej2-buttons/styles/material.css';
@import "../node_modules/@syncfusion/ej2-vue-calendars/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/common/size-modes-cs3" %}

See also

- [Sidebar responsiveness](#)
- [DataGrid responsiveness](#)
- [TreeGrid responsiveness](#)
- [Dashboard Layout responsiveness](#)
- [Kanban responsiveness](#)
- [Toolbar responsiveness](#)
- [Tab responsiveness](#)

## Icons in Vue Appearance component

Syncfusion's icon library is a collection of pre-designed icons that can be used to enhance the user interface of an application. This pre-designed icons are set of `base64` formatted font icons. Utilizing this icon library can make it simpler to create a cohesive, visually pleasing design for an application.

### Referring icons in the Vue application

Using the below approaches, the icons can be referenced in the Vue application.

- [npm package](#) - Use the npm package to access icons.
- [CDN reference](#) - Use the static web asset to access icons.

#### The npm package

All Syncfusion theme icons are shipped in the [ej2-icons](#) package, which is published on the [npmjs.com](#) public registry. This package contains both CSS and SCSS theme files for all themes.

Icons can be used from the npm package `ej2-icons`. To use the icons, install the npm package using the following command:

```
`bash
```

```
npm install @syncfusion/ej2-icons
```

```
,
```

Refer to the following syntax to use icons in a Vue application:

```
,
```

```
@import "../nodemodules/@syncfusion/ej2-icons/<themename>.css";
```

```
,
```

#### Example:



```
@import "../node_modules/@syncfusion/ej2-icons/material.css";
```

### CDN reference

All Syncfusion theme icons are available on the CDN. Instead of using a local resource on the server, use a cloud CDN to refer to the icons.

Make sure that the version of the icons in the URL matches the version of the Syncfusion Vue package. This will prevent compatibility issues and ensure that the correct version of the icons is loaded.

To use the icons from the CDN, refer to the icons by URLs in the application. This can be done by linking the icons in the HTML file by adding a link tag to the head section.

```
// Bootstrap5
<head>
<link href="https://cdn.syncfusion.com/ej2/ej2-icons/styles/bootstrap5.css" rel="stylesheet"/>
</head>

//Material
<head>
<link href="https://cdn.syncfusion.com/ej2/ej2-icons/styles/material.css" rel="stylesheet"/>
</head>
```

### Steps to use icons library

Let's create a Vue application using the following command:

For an introduction and configuration of the common specifications, see [getting started with the Syncfusion Vue application](#).

### Using icons directly in HTML element

The built-in Syncfusion icons can be rendered directly in the HTML element by defining the `e-icons` class, which contains the font-family and common properties of font icons, and defining the available icon's class with the `e-` prefix.

The following steps explain the direct rendering of the Syncfusion icon in the HTML element.

1. Add the class name `e-icons` to the HTML element that needs to render the icon.
2. Add the icon class with corresponding icon content from the [available icons](#). For example, the below code snippet represents the paste icon class.

```
.e-paste:before {
content:'\e355';
```

```
}
,
```

3. Add `e-icons` and `e-paste` classes to the HTML element.

```
,
<span class="e-icons e-paste"></span>
,
```

4. Add the CDN link reference of icons library in the `~index.html` file.

```
,
<link href="https://cdn.syncfusion.com/ej2/ej2-icons/styles/bootstrap5.css" rel="stylesheet" />
,
```

The below code snippet represents a complete example of icon usage.

#### **APP.VUE**

```
<template>
  <div class="icon-bar">
    <span class="e-icons e-cut"></span>
    <span class="e-icons e-copy"></span>
    <span class="e-icons e-paste"></span>
  </div>
</template>
<script>
  import Vue from "vue";
  export default {
  }
</script>
<style>

.icon-bar {
  width: 20%;
  background-color: lightgrey;
  overflow: auto;
  margin-top: 20px;
  margin-left: 240px;
}
.icon-bar span {
  float: left;
  width: 33%;
  text-align: center;
  padding: 12px 0;
  transition: all 0.3s ease;
}
.icon-bar span:hover {
  background-color: #e2e2e2;
}
</style>
```

```
{% previewsample "page.domainurl/code-snippet/common/animation-multiple-cs1" %}
```

### Icon size

The `ej2-icons` package offers options to display icons in different size modes. A user can use different icon sizes in their application based on touch or mouse mode. If the user is using touch mode, add `e-large` class to the element to make the icon easily interactable, or add the `e-small` or `e-medium` class in mouse mode.

The pre-defined icon size is present in the available classes listed below.

- `e-small` - Sets the icon size as 8px.
- `e-medium` - Sets the icon size to 16px.
- `e-large` - Sets the icon size to 24px.

### Example:

,

```
<span class="e-icons e-small e-search"></span>
```

```
<span class="e-icons e-medium e-search"></span>
```

```
<span class="e-icons e-large e-search"></span>
```

,

### APP.VUE

```
<template>
  <div>
    <p><b>Smaller Icons</b></p>
    <div class="small-icon-bar">
      <span class="e-icons e-small e-cut"></span>
      <span class="e-icons e-small e-copy"></span>
      <span class="e-icons e-small e-paste"></span>
    </div>
    <p><b>Medium Icons</b></p>
    <div class="medium-icon-bar">
      <span class="e-icons e-medium e-cut"></span>
      <span class="e-icons e-medium e-copy"></span>
      <span class="e-icons e-medium e-paste"></span>
    </div>
    <p><b>Larger Icons</b></p>
    <div class="large-icon-bar">
      <span class="e-icons e-large e-cut"></span>
      <span class="e-icons e-large e-copy"></span>
      <span class="e-icons e-large e-paste"></span>
    </div>
  </div>
</template>
<script>
  import Vue from "vue";
  export default {
  }
</script>
<style>
```

```

.small-icon-bar {
  width: 20%;
  background-color: lightgrey;
  overflow: auto;
}
.small-icon-bar span:hover {
  background-color: #e2e2e2;
}
.small-icon-bar span {
  float: left;
  width: 33%;
  text-align: center;
  padding: 12px 0;
  transition: all 0.3s ease;
  font-size: 36px;
}
.medium-icon-bar {
  width: 20%;
  background-color: lightgrey;
  overflow: auto;
}
.medium-icon-bar span:hover {
  background-color: #e2e2e2;
}
.medium-icon-bar span {
  float: left;
  width: 33%;
  text-align: center;
  padding: 12px 0;
  transition: all 0.3s ease;
  font-size: 36px;
}
.large-icon-bar {
  width: 20%;
  background-color: lightgrey;
  overflow: auto;
}
.large-icon-bar span:hover {
  background-color: #e2e2e2;
}
.large-icon-bar span {
  float: left;
  width: 33%;
  text-align: center;
  padding: 12px 0;
  transition: all 0.3s ease;
  font-size: 36px;
}
</style>

```

{% previewsample "page.domainurl/code-snippet/common/animation-multiple-cs2" %}

#### *Icon appearance customization*

The Syncfusion Vue icons can be customized with custom color and size by overriding the `e-icons` class. Customizing the icons in the library can be useful for making the icons more visually appealing and

fitting to the design of the application. For example, a user can change the color of an icon to match the color scheme of their application, or increase the size of an icon to make it more visible on smaller screens. It may also be useful for creating a consistent look and feel across different parts of the application. Overall, customizing the icons in the library can improve the overall user experience of the application.

### APP.VUE

```
<template>
  <div class="icon-bar">
    <span class="e-icons e-cut"></span>
    <span class="e-icons e-copy"></span>
    <span class="e-icons e-paste"></span>
  </div>
</template>
<script>
  import Vue from "vue";
  export default {
  }
</script>
<style>

.e-icons {
  color: #ffffff;
}
.icon-bar {
  width: 20%;
  background-color: #555;
  overflow: auto;
  margin-top: 20px;
  margin-left: 240px;
}
.icon-bar span {
  float: left;
  width: 33%;
  text-align: center;
  padding: 12px 0;
  transition: all 0.3s ease;
}
.icon-bar span:hover {
  background-color: #000;
}
</style>
```

{% previewsample "page.domainurl/code-snippet/common/animation-multiple-cs3" %}

### Available icons

The complete package of Essential JS 2 icons is listed below. The corresponding icon content can be referred in the content section.

<!-- markdownlint-disable MD033 -->

### Material

```
<iframe class="doc-sample-frame"
src="https://ej2.syncfusion.com/products/icons/material/demo.html"
style="height:1000px;width:100%;"></iframe>
```

### Fabric

```
<iframe class="doc-sample-frame" src="https://ej2.syncfusion.com/products/icons/fabric/demo.html"
style="height:1000px;width:100%;"></iframe>
```

### Bootstrap

```
<iframe class="doc-sample-frame"
src="https://ej2.syncfusion.com/products/icons/bootstrap/demo.html"
style="height:1000px;width:100%;"></iframe>
```

### Bootstrap 4

```
<iframe class="doc-sample-frame"
src="https://ej2.syncfusion.com/products/icons/bootstrap4/demo.html"
style="height:1000px;width:100%;"></iframe>
```

### Bootstrap 5

```
<iframe class="doc-sample-frame"
src="https://ej2.syncfusion.com/products/icons/bootstrap5/demo.html"
style="height:1000px;width:100%;"></iframe>
```

### High Contrast

```
<iframe class="doc-sample-frame"
src="https://ej2.syncfusion.com/products/icons/highcontrast/demo.html"
style="height:1000px;width:100%;"></iframe>
```

### Tailwind CSS

```
<iframe class="doc-sample-frame"
src="https://ej2.syncfusion.com/products/icons/tailwind/demo.html"
style="height:1000px;width:100%;"></iframe>
```

### Fluent

```
<iframe class="doc-sample-frame" src="https://ej2.syncfusion.com/products/icons/fluent/demo.html"
style="height:1000px;width:100%;"></iframe>
```

## Theme studio in Vue Appearance component

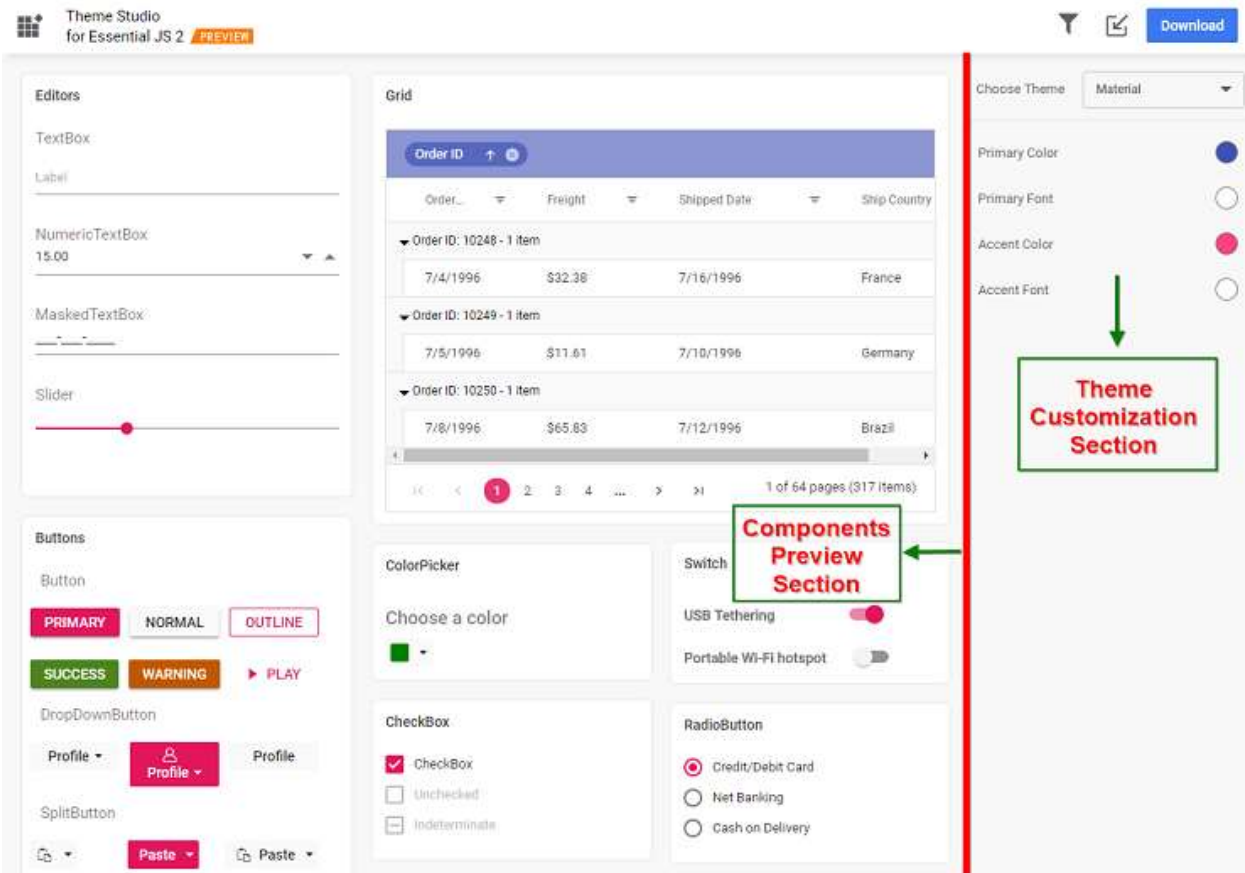
Theme Studio for Essential JS 2 can be used to customize a new theme from an existing theme. It doesn't support with Data visualization components like Chart, Diagram, Gauge, Range Navigator, Maps.

### Customizing theme color from theme studio

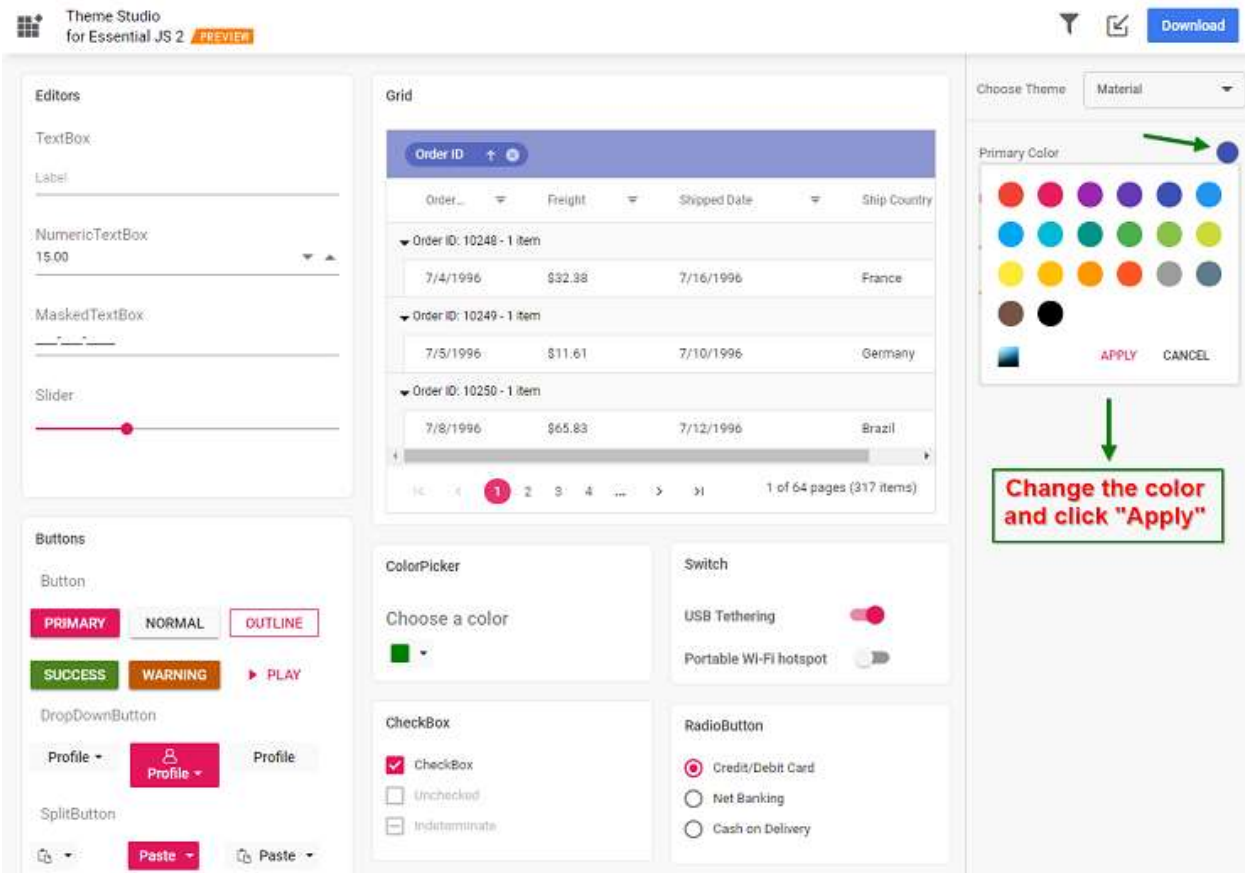
The Syncfusion Vue (Essential JS 2) themes are developed under the SCSS environment. Each theme has a unique common variable list. When you change the common variable color code value, it will reflect in all Syncfusion Vue UI components. All Syncfusion Vue component styles are derived from these [theme-based common variables](#). This common variable list is handled inside the theme studio application for customizing theme-based colors.

**Step 1:** Navigate to the theme studio application at <https://ej2.syncfusion.com/themestudio/>.

**Step 2:** The theme studio application page can be divided into two sections: the components preview section on the left, and the theme customization section on the right.

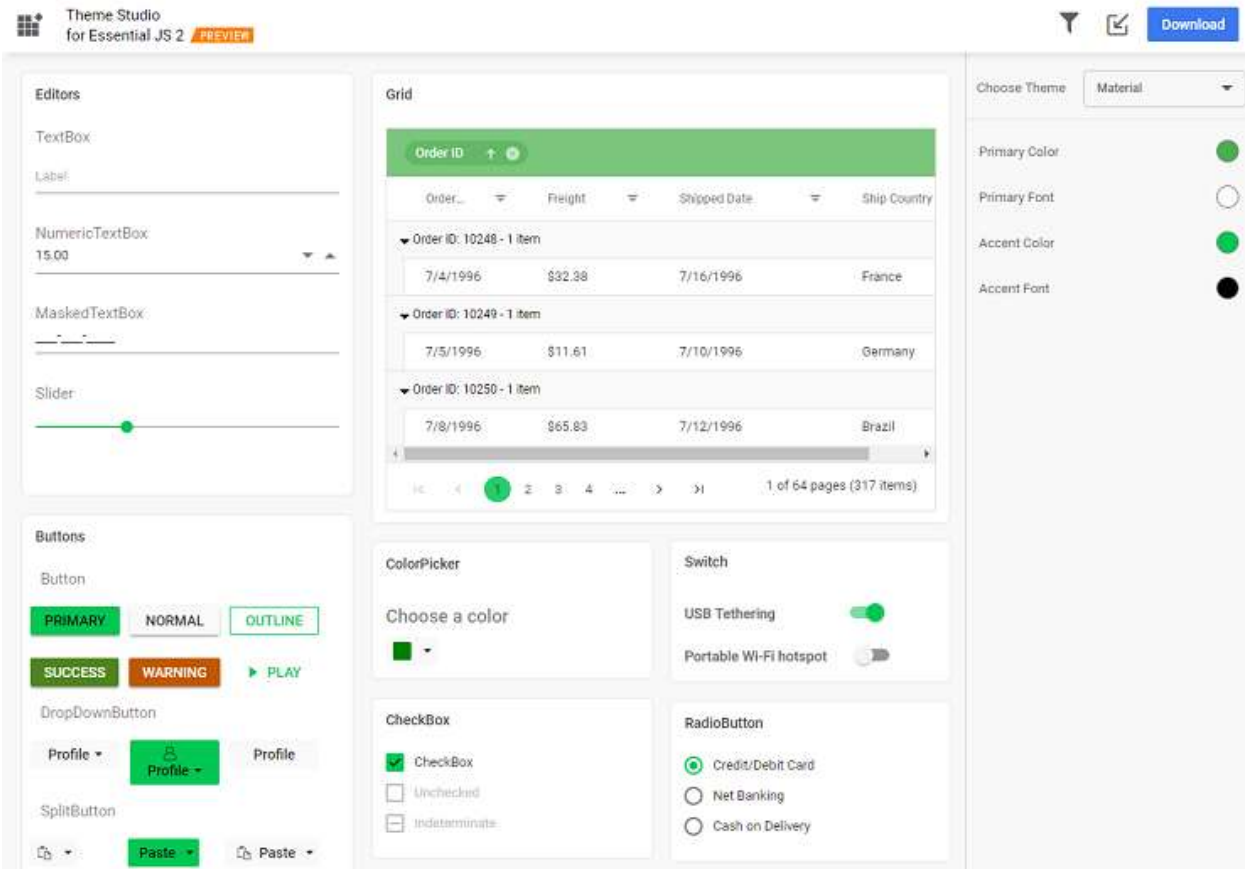


**Step 3:** Click the color pickers in the theme customization section to select your desired color.



**Step 4:** After selecting colors with the color pickers, the Syncfusion Vue components will have the newly selected colors applied to them in the components preview section.

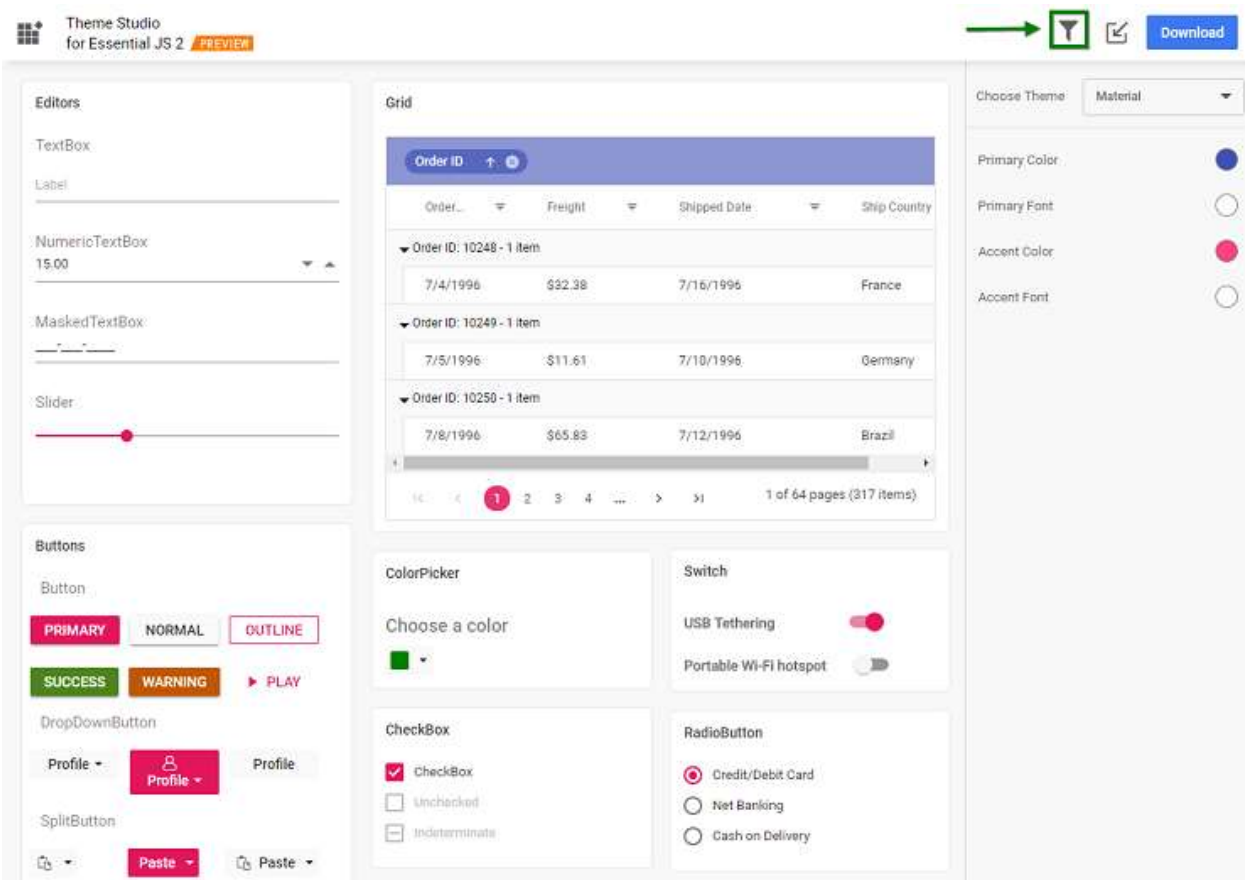




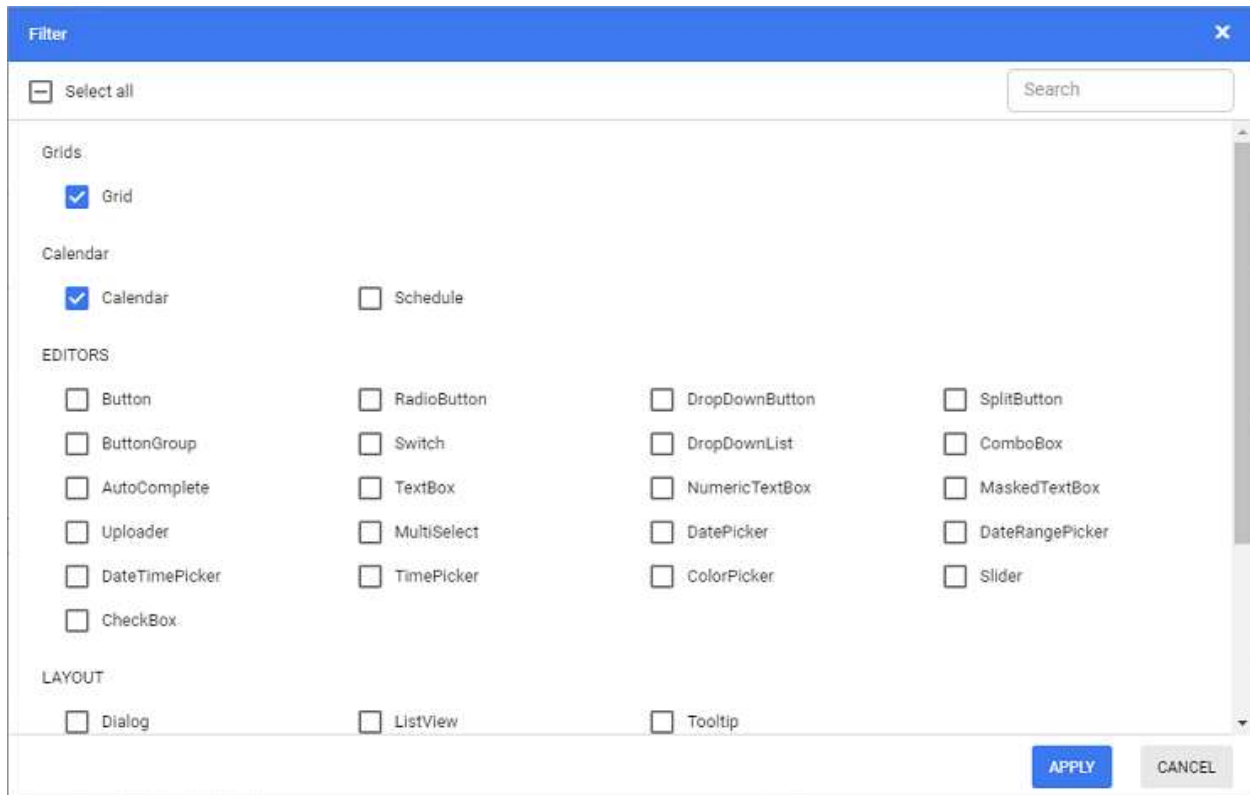
### *Filtering a specific list of components*

Using the theme studio, you can apply custom themes to a list of specific components. This option is useful when you have integrated a selective list of Syncfusion Vue components in your application. The theme studio will filter the selected components and customize the final output for those component's styles alone, reducing the final output file size.

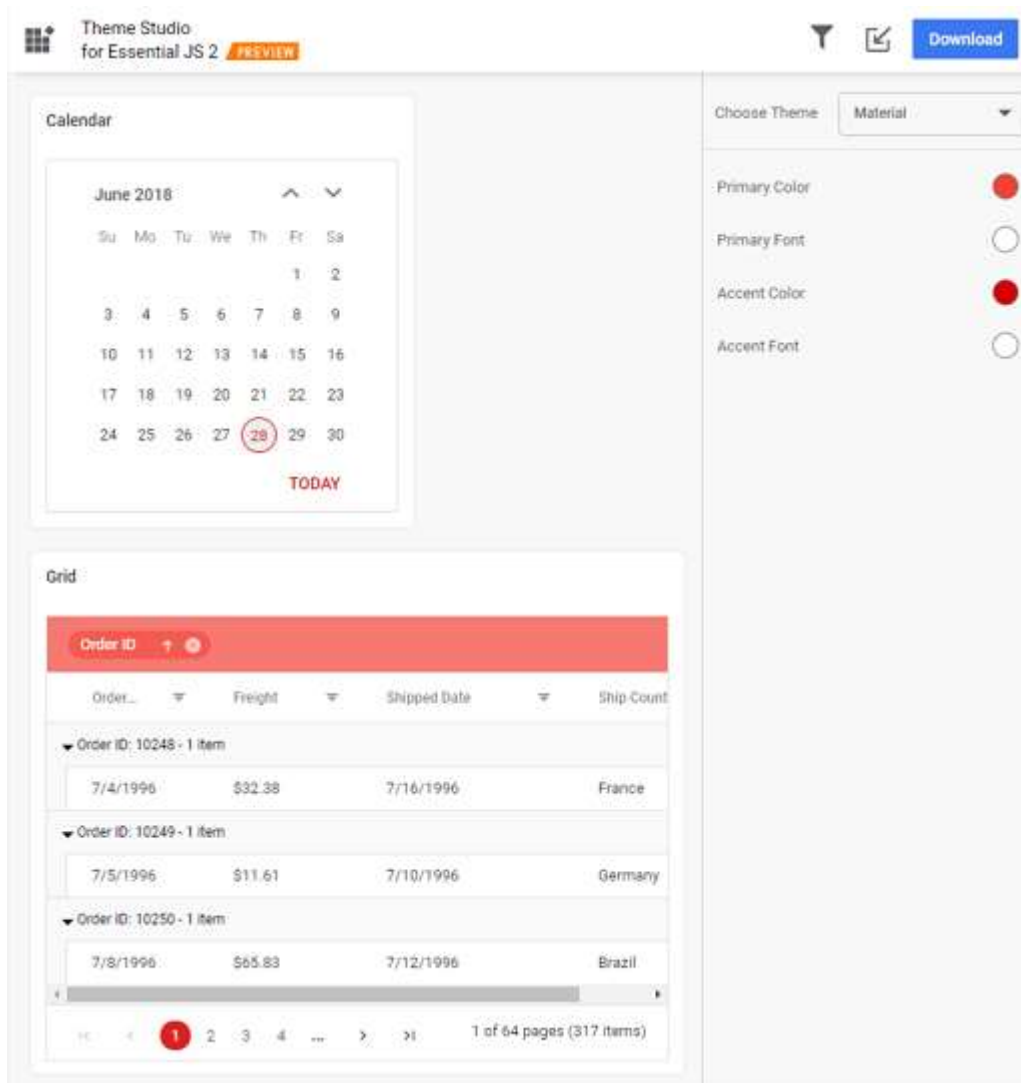
**Step 1:** Click the Filter icon in the top right corner and select the components whose theme you want to customize.



**Step 2:** Click the Apply button in the Filter dialog. Now, only the selected components will be rendered in the components preview section.



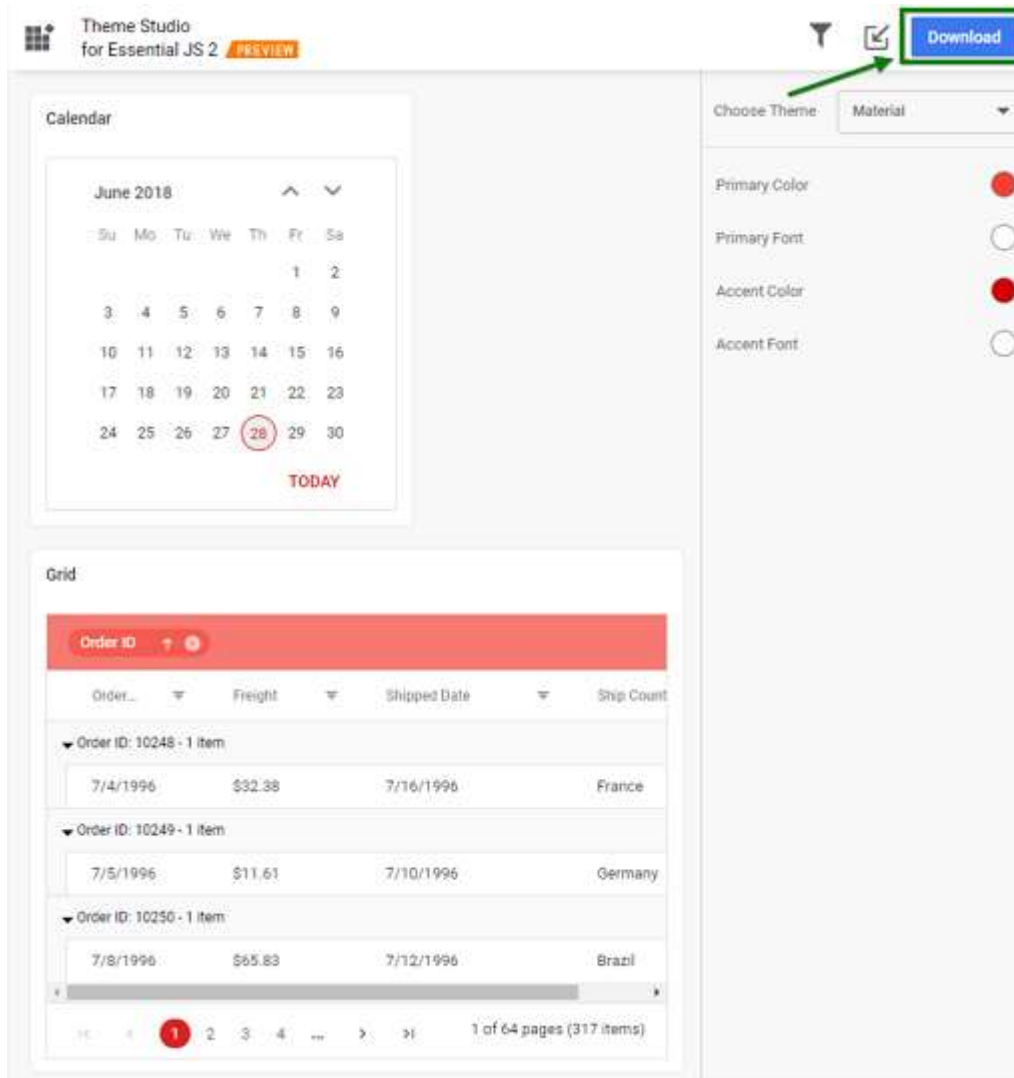
**Step 3:** Now you can customize the colors in the theme customization section for the components you selected.



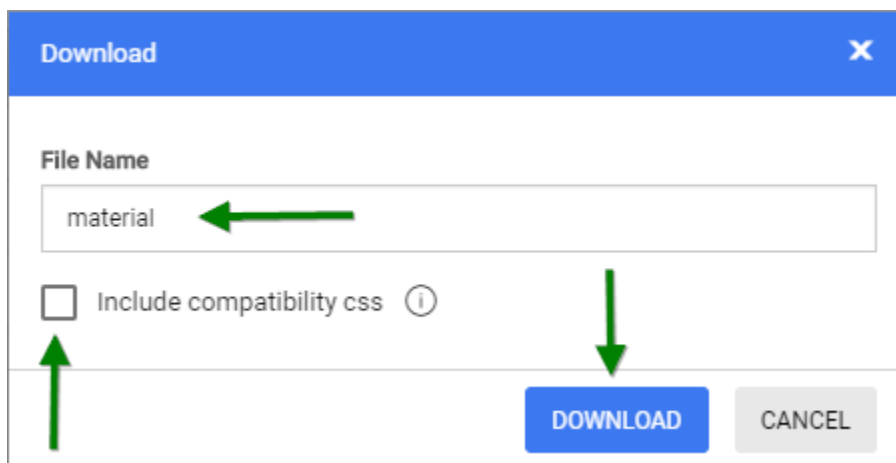
### [Download the customized theme](#)

You can download the custom styles after customizing the theme colors.

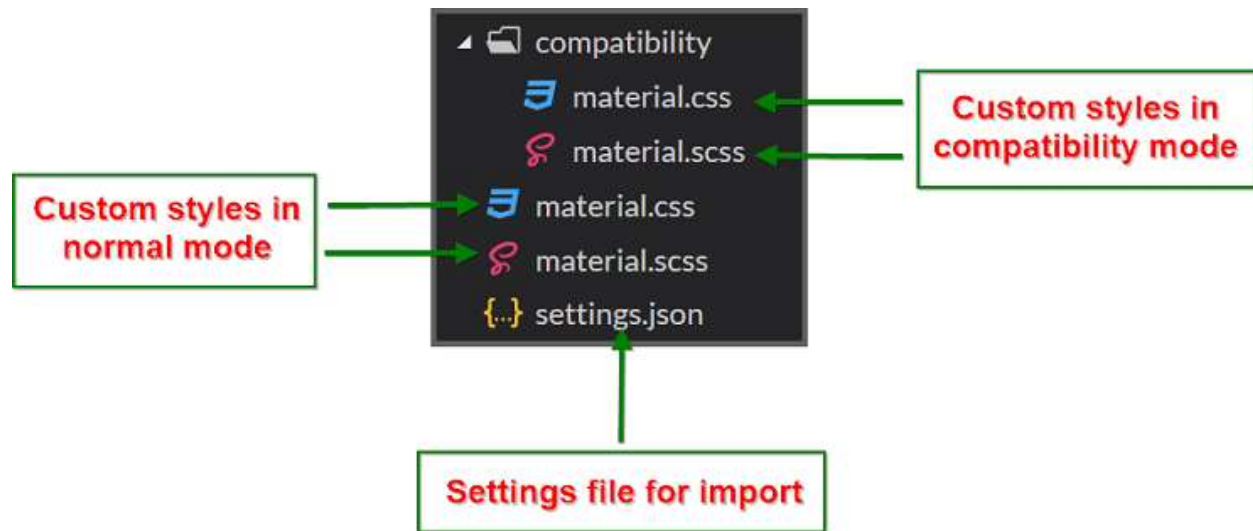
**Step 1:** Click the Download button in the top right corner. The Download dialog will open.



**Step 2:** Assign a theme name in the File Name field and click the Download button.



**Step 3:** The download styles will come as a zip file that contains SCSS and CSS files for the selected Syncfusion Vue components. The current settings are stored in the `settings.json` file.



#### Using a customized theme in a web application

You can directly use the customized CSS file in the web application.

**Step 1:** Copy/paste the customized CSS file from the download folder into your application at any folder.  
Example: `styles/{file-name}.css`.

**Step 2:** Refer the customized CSS file reference in the `index.html` or `shared/_layout.cshtml` main page head section.

,

```
<head>
```

```
<link href="styles/{file-name}.css" rel="stylesheet"/>
```

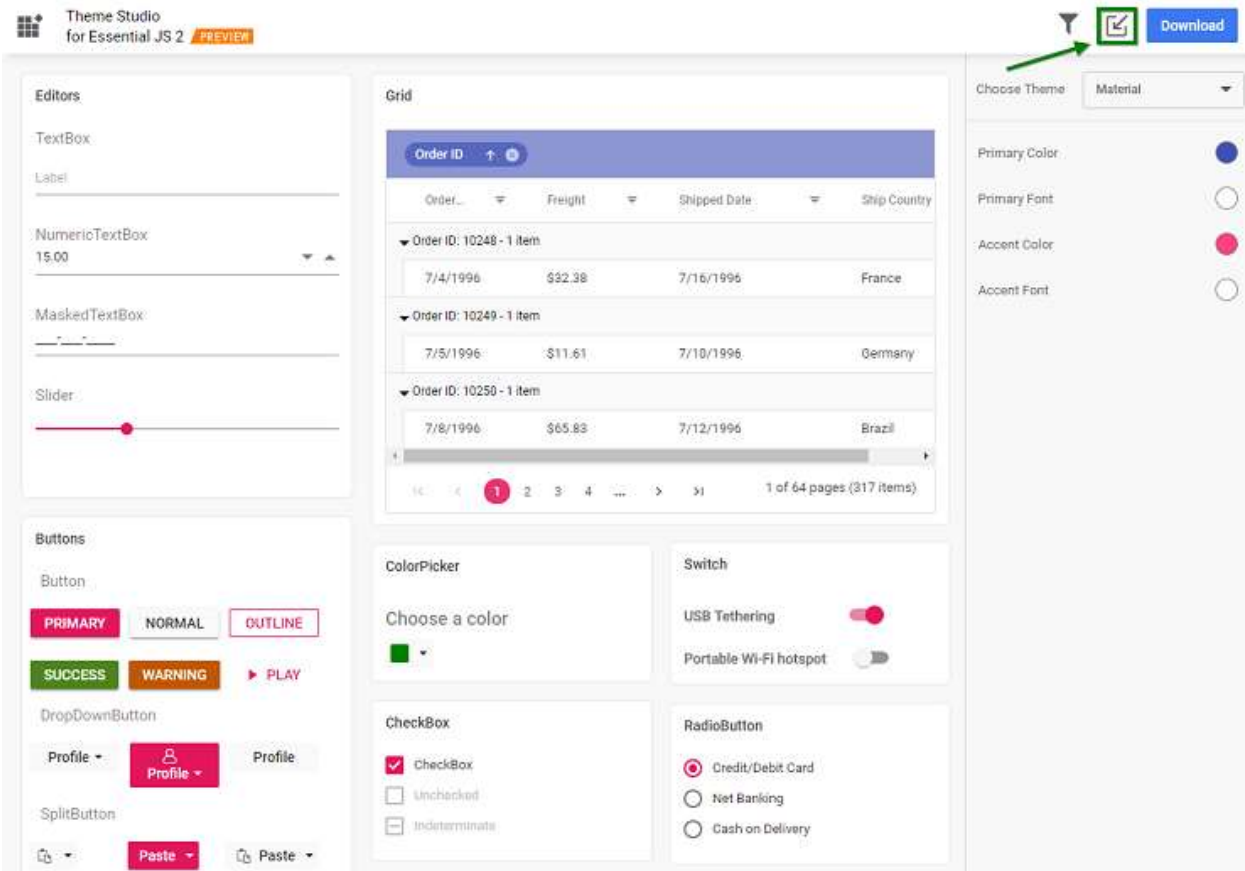
```
</head>
```

,

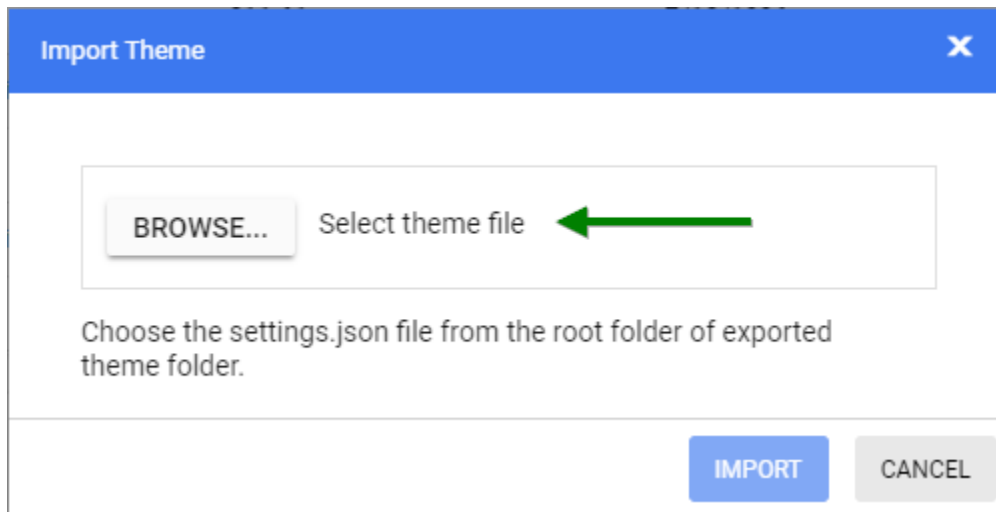
#### Import previously changed settings into the theme studio

When you want to change your application theme and UI design in the future, you won't need to customize the Syncfusion Vue components from scratch in the theme studio. Just import the old `settings.json` file to review and update your stored settings in the theme studio application.

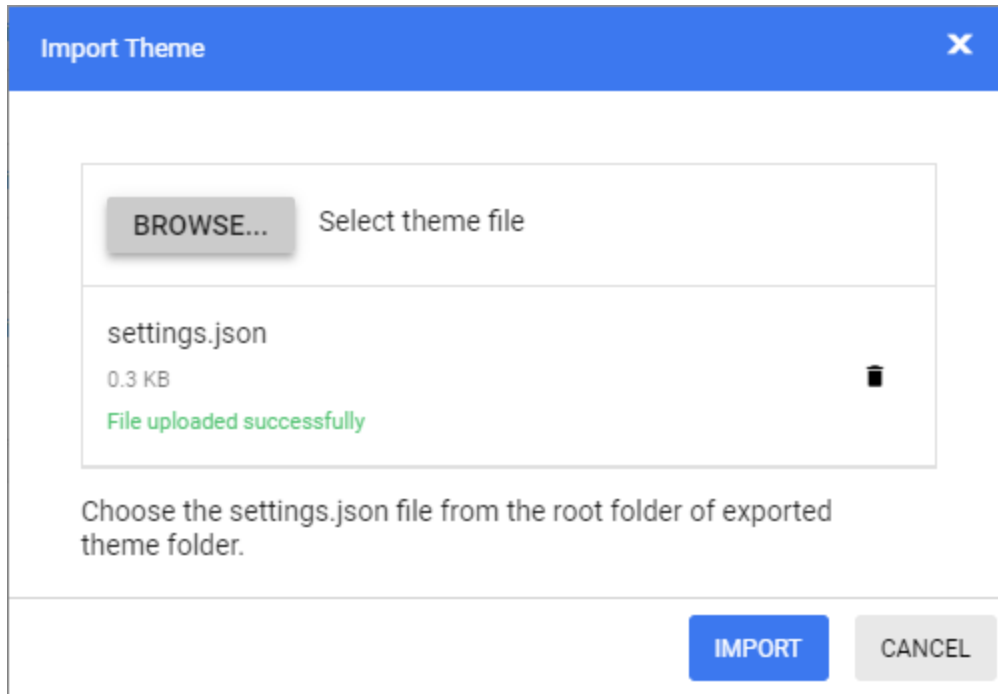
**Step 1:** Click the Import icon in the top right corner.



**Step 2:** The Import Theme dialog will open. Click the Browse button and select a `settings.json` file you exported previously.

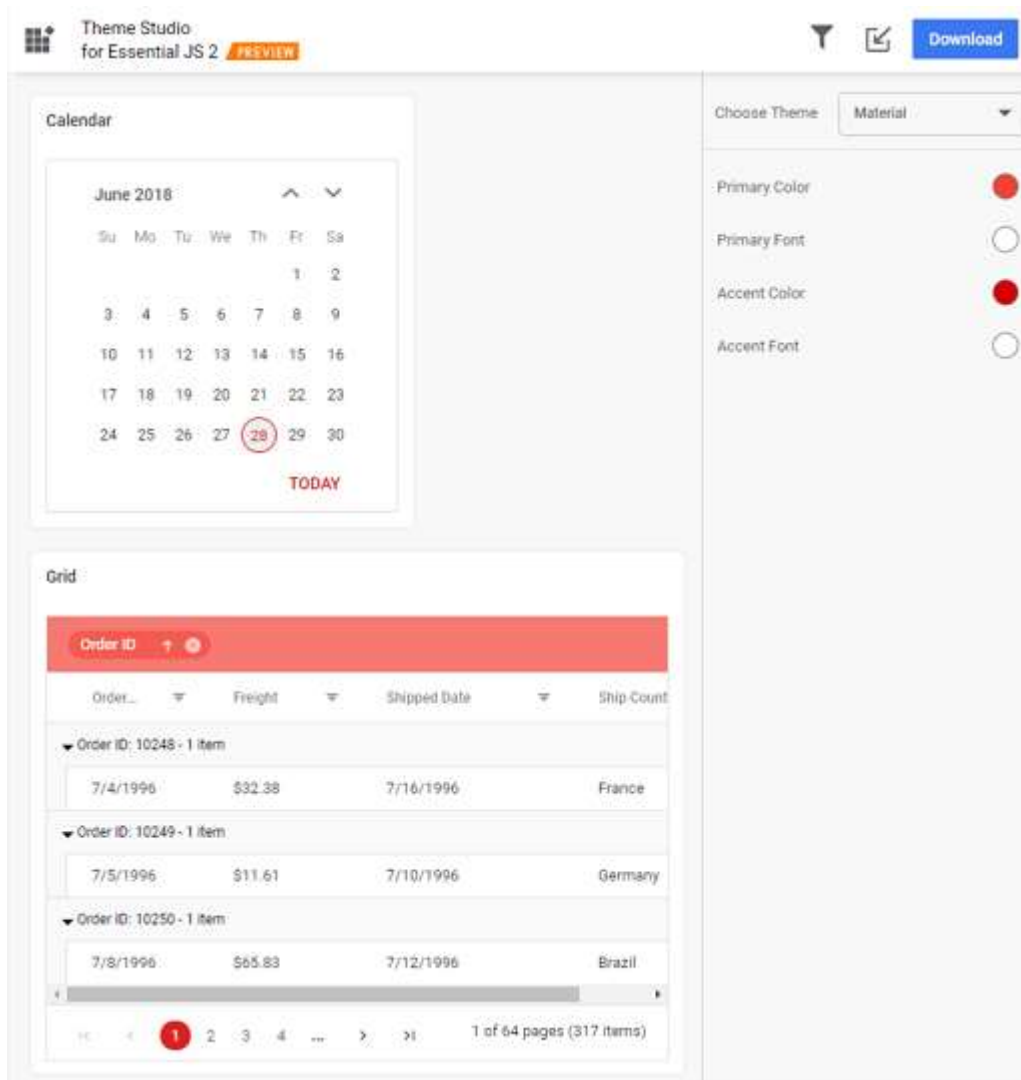


**Step 3:** Click the Import button.



**Step 4:** The stored data will be reflected in the theme studio application. Now you can change the theme colors based on your latest design and export the theme again.





**Step 5:** The exported file will contain your latest changes. You can just replace the older custom style with the newer one to refresh your application.

### Material 3 Theme

Material 3 is an updated theme that encompasses enhanced theming, components, and personalization features, including dynamic color capabilities. It has been specifically designed to align seamlessly with the new visual style and system UI introduced in Android 12 and above. For more detailed information, please refer to the following link: [Link to Material 3 Information](#).

### Syncfusion Material 3 Theme

Syncfusion has introduced the Material theme across all EJ2 Controls. The theme includes light and dark variants and utilizes CSS variables for easy customization of control colors in CSS format. This implementation enables seamless switching between light and dark color schemes, offering users a flexible solution to suit their preferences and application needs.

**Note:** Please be aware that in the Material 3 theme, we utilize CSS variables with `rgb()` values for our color variables. Using hex values in this context may lead to improper functionality. For example, in previous versions of our Material theme or other themes, the primary color variable was defined as

follows: `$primary: #6200ee;`. In the Material 3 theme, the primary color variable is defined as follows: `--color-sf-primary: 98, 0, 238;`.

### What are CSS Variables?

CSS variables, also known as custom properties, are a powerful feature in CSS that allows you to define reusable values and apply them throughout your stylesheets. Prefixed with "--", CSS variables can be utilized in any property value within a CSS rule. To retrieve the value of a CSS variable, the `var()` function is used. CSS variables can access the Document Object Model (DOM), enabling the creation of variables with either local or global scope. For further details, please consult the following link: [Link to CSS Variables Information](#).

### How does Syncfusion Theming utilize CSS Variables?

The Material 3 theme in our system incorporates CSS variable support, utilizing CSS variables with `rgb()` values for color customization.

```
`CSS
:root {
--color-sf-black: 0, 0, 0;
--color-sf-white: 255, 255, 255;
--color-sf-primary: 103, 80, 164;
--color-sf-primary-container: 234, 221, 255;
--color-sf-secondary: 98, 91, 113;
--color-sf-secondary-container: 232, 222, 248;
--color-sf-tertiary: 125, 82, 96;
--color-sf-tertiary-container: 255, 216, 228;
--color-sf-surface: 255, 255, 255;
--color-sf-surface-variant: 231, 224, 236;
--color-sf-background: var(--color-sf-surface);
--color-sf-on-primary: 255, 255, 255;
--color-sf-on-primary-container: 33, 0, 94;
--color-sf-on-secondary: 255, 255, 255;
--color-sf-on-secondary-container: 30, 25, 43;
--color-sf-on-tertiary: 255, 255, 255;
}
```

### Exploring Color Customization

Through the utilization of these CSS variables, customization of the color variables becomes remarkably straightforward. For example, to customize the primary color variable in this theme, simply modify its value in the root values.

### Default primary value

```

--color-sf-black: 0, 0, 0;
--color-sf-white: 255, 255, 255;
--color-sf-primary: 103, 80, 164;
--color-sf-primary-container: 234, 221, 255;
--color-sf-secondary: 98, 91, 113;
--color-sf-secondary-container: 232, 222, 248;
--color-sf-tertiary: 125, 82, 96;
--color-sf-tertiary-container: 255, 216, 228;
--color-sf-surface: 255, 255, 255;
--color-sf-surface-variant: 231, 224, 236;

```

### Customized primary value

```

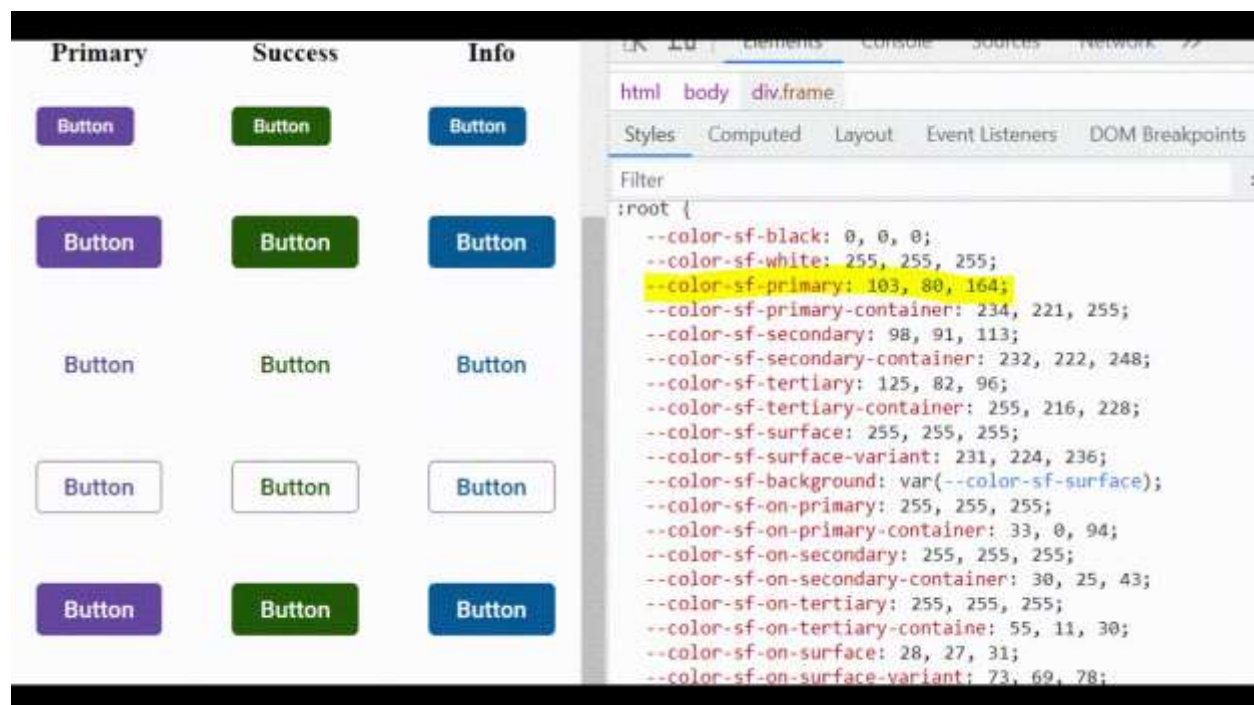
--color-sf-black: 0, 0, 0;
--color-sf-white: 255, 255, 255;
--color-sf-primary: 236, 80, 164;
--color-sf-primary-container: 234, 221, 255;
--color-sf-secondary: 98, 91, 113;
--color-sf-secondary-container: 232, 222, 248;
--color-sf-tertiary: 125, 82, 96;
--color-sf-tertiary-container: 255, 216, 228;
--color-sf-surface: 255, 255, 255;
--color-sf-surface-variant: 231, 224, 236;

```

With this CSS variable support, you can effortlessly customize the color variable values for our EJ2 controls.

### Dark mode support

The controls in our system now seamlessly transition between light and dark modes without any noticeable delay. This achievement was made by consolidating the configurations of the light theme and dark theme into [material 3 light theme](#) file.



### APP.VUE

```

<template>
<div>
  <ejs-checkbox label='Default'></ejs-checkbox>
  <ejs-button cssClass='e-primary'>Button</ejs-button>
  <ejs-button cssClass='e-success'>Button</ejs-button>
  <ejs-button cssClass='e-info'>Button</ejs-button>
  <ejs-button cssClass='e-warning'>Button</ejs-button>
  <ejs-button cssClass='e-danger'>Button</ejs-button>
</div>
</template>
<script>
import Vue from 'vue';
import { ButtonPlugin } from '@syncfusion/ej2-vue-buttons';
import { enableRipple } from '@syncfusion/ej2-base';
enableRipple(true);
Vue.use(CheckBoxPlugin);
Vue.use(ButtonPlugin);
export default {}
</script>
<style>
@import '../node_modules/@syncfusion/ej2-base/styles/material3.css';
@import '../node_modules/@syncfusion/ej2-buttons/styles/material3.css';
.e-checkbox-wrapper {
  margin-top: 18px;
}
button {
  margin: 25px 5px 20px 20px;
}
.dark
{
  background-color:black;
}
</style>

```

{% previewsample "page.domainurl/code-snippet/common/material3-cs1" %}

The above example demonstrates the appearance of an application in dark mode. By using CSS variable support, the transition between light and dark mode is smooth and visually pleasing.

`CSS

```

.e-dark-mode {
--color-sf-black: 0, 0, 0;
--color-sf-white: 255, 255, 255;
--color-sf-primary: 208, 188, 255;
--color-sf-primary-container: 79, 55, 139;
--color-sf-secondary: 204, 194, 220;
--color-sf-secondary-container: 74, 68, 88;
--color-sf-tertiary: 239, 184, 200;
--color-sf-tertiary-container: 99, 59, 72;

```

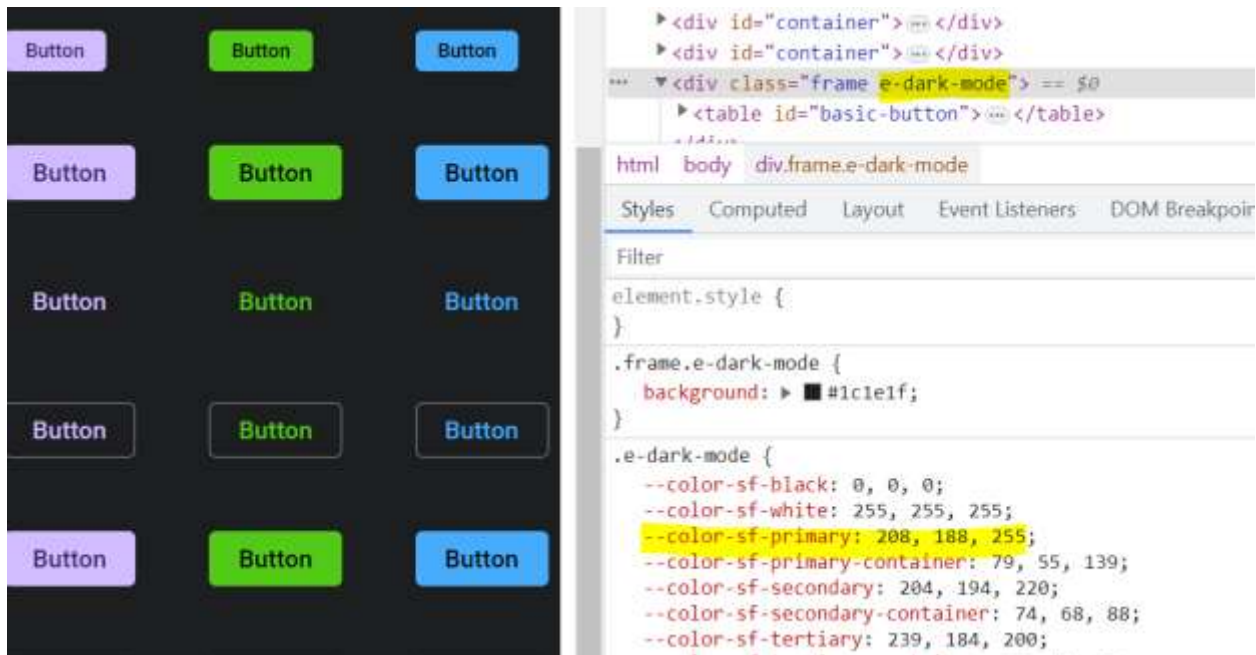
```

--color-sf-surface: 28, 27, 31;
--color-sf-surface-variant: 28, 27, 31;
--color-sf-background: var(--color-sf-surface);
--color-sf-on-primary: 55, 30, 115;
--color-sf-on-primary-container: 234, 221, 255;
--color-sf-on-secondary: 51, 45, 65;
--color-sf-on-secondary-container: 232, 222, 248;
--color-sf-on-tertiary: 73, 37, 50;
}

```

#### [How to switch to dark mode](#)

With this CSS variable support, transitioning between light and dark theme modes has become effortless. To switch to dark mode, simply add the `e-dark-mode` class to the body section of your application. Upon adding this class, the theme will seamlessly switch to dark mode. Please refer to the example image below for guidance.



#### [ThemeStudio application](#)

The ThemeStudio application now includes seamless integration with the Material 3 theme, offering a comprehensive solution for customization requirements. This enhancement enables users to effortlessly customize and personalize their themes.

Access the Syncfusion ThemeStudio application, featuring the Material 3 theme, via the following link:  
[Link to Syncfusion ThemeStudio with Material 3 Theme](#)

## Common

### Accessibility in Syncfusion Vue Components

#### Accessibility overview

Accessibility in components refers to the practice of designing and building user interface elements in a way that makes them accessible to users with disabilities. This can include a variety of things, such as making sure that text is high-contrast and easy to read, providing alternative text for images, and designing controls and interactions that can be used with a keyboard or assistive technology.

#### Accessibility standards

The component is said to be accessible when it is constructed in accordance with certain standards that are required to make it accessible.

The accessibility of the components consists of the following standards and aspects:

- [ADA](#) - A law to ensure that people with disabilities have the same opportunities and access as people without disabilities.
- [WCAG 2.2](#) - The Web Content Accessibility Guidelines (WCAG) provide guidelines developed by the World Wide Web Consortium (W3C) to ensure web content is accessible to people with disabilities. [WCAG 2.2](#) establishes a framework of accessibility principles and their associated success criteria. The level of accessibility conformance achieved by a web application is determined by the extent to which it meets these success criteria, categorized into three levels: A, AA, and AAA.
- [Section 508](#) - It is a set of guidelines for making electronic and information technology (EIT) accessible to people with disabilities. These standards apply to federal agencies in the United States, and they are based on the Web Content Accessibility Guidelines (WCAG).
- [WAI-ARIA](#) - WAI-ARIA stands for "Web Accessibility Initiative - Accessible Rich Internet Applications." It is a set of technical specifications and guidelines developed by the World Wide Web Consortium (W3C) as part of the Web Accessibility Initiative (WAI). WAI-ARIA is designed to enhance the accessibility of dynamic web content, particularly web applications and rich internet applications (RIAs), for people with disabilities. WAI-ARIA provides a set of roles, states, and properties that can be added to HTML elements to provide additional context and information about the purpose and behavior of those elements. This can help assistive technologies better understand and interpret web content and interact with web applications.
- [Keyboard navigation](#) - It refers to the ability to use a keyboard to interact with and navigate through a user interface. It is an important aspect of web accessibility, as it allows people who cannot use a mouse or other pointing device to access and use web content and applications.

Syncfusion Vue components adhere to these established standards.

#### Accessibility compliance

The accessibility support provided by Syncfusion Vue components is based on a collection of methodologies for adhering to and applying [recognized standards and technical specifications](#) to ensure an intuitive experience for people with disabilities.

There are several methodologies of accessibility validation that can be performed on the Syncfusion Vue components, such as:

- The [WAI-ARIA patterns](#) are followed by the Syncfusion Vue components to enable appreciable accessibility.

- Each Vue component is subjected to manual testing with a screen reader and also automated test cases to ensure the component's required attributes.
- Attributes are allocated and updated correctly during interaction as well. Each component has been assigned a distinct **Role** attribute and its own set of ARIA attributes defined by the [WCAG 2.2](#) specification.

In addition to the methodologies mentioned above, Syncfusion Vue components are constructed to support the following accessibility aspects.

#### *Screen reader support*

A screen reader allows people who are blind or visually impaired to use a computer by reading aloud the text that is displayed on the screen. Syncfusion Vue components followed the [WAI-ARIA](#) standards to work properly in the screen readers such as [Narrator](#) for Windows and [Embedded VoiceOver](#) for MAC.

#### *Right-To-Left support*

Right-to-Left (RTL) support allows applications to effectively communicate with users who use languages that are written from right to left, such as Arabic, Hebrew, etc. Syncfusion Vue components support the Right-to-Left feature. Refer to the [Right-to-Left documentation](#) to learn more about this support.

#### *Color contrast*

Syncfusion Vue components come equipped with [predefined themes](#) that guarantee the presence of satisfactory color contrast, benefiting individuals with visual impairments.

#### *Mobile device support*

Syncfusion Vue components are more user-friendly and accessible to individuals using mobile devices, including those with disabilities. These are designed to be responsive, adaptable to various screen sizes and orientations, and touch-friendly.

#### *Keyboard navigation support*

Syncfusion Vue components support keyboard navigation, allowing users who rely on alternate methods to effortlessly navigate and interact with the component.

#### *Ensuring accessibility*

Ensuring the accessibility of Syncfusion Vue components is crucial for making the product inclusive and user-friendly for individuals with disabilities. This process involves various types of accessibility testing, including:

- **Automated testing:** The Syncfusion Vue component's accessibility levels are ensured through an [accessibility-checker](#) and [axe-core](#) software tools.
- **Manual testing:** This type of testing involves manually evaluating the Syncfusion Vue components. During manual accessibility testing, testers will ensure accessibility using the screen readers such as [Narrator](#) for Windows and [Embedded VoiceOver](#) for MAC.

Syncfusion Vue components will keep improving when there is anything required. It also involves client feedback to make the component more accessible.

#### *Accessibility support for specific components*

Consult the component-specific documents below for detailed information about how Syncfusion Vue components ensure compliance with accessibility standards, encompassing Section 508, WAI-ARIA, WCAG 2.2, keyboard navigation, and more.

<style>

table

```
{
border:0 !important;
line-height: 2!important;
}
tr
{
border:0 !important;
}
td
{
border:0 !important;
vertical-align: top;
}
.controlanchorlink
{
text-decoration: none !important;
font-size: 14px !important;
text-align: left !important;
padding: 5px 0px;
letter-spacing: 1px;
}
.controlcategory
{
font-size: 14px !important;
text-align: left !important;
font-weight: bold !important;
letter-spacing: 0.7px;
}
}
```

</style>

GRIDS	DATA VISUALIZATION	CALENDARS	DROPDOWNS
-------	-----------------------	-----------	-----------



<a href="#">DataGrid</a>	<a href="#">Accumulation Chart</a>	<a href="#">Scheduler</a>	<a href="#">AutoComplete</a>
<a href="#">Pivot Table</a>	<a href="#">Charts</a>	<a href="#">Gantt Chart</a>	<a href="#">ListBox</a>
<a href="#">TreeGrid</a>	<a href="#">Stock Chart</a>	<a href="#">Calendar</a>	<a href="#">ComboBox</a>
<a href="#">Spreadsheet</a>	<a href="#">Circular Gauge</a>	<a href="#">DatePicker</a>	<a href="#">Dropdown List</a>
FILE VIEWERS & EDITORS	<a href="#">Linear Gauge</a>	<a href="#">DateRangePicker</a>	<a href="#">Multiselect DropDown</a>
	<a href="#">Maps</a>	<a href="#">DateTime Picker</a>	<a href="#">DropDown Tree</a>
	<a href="#">Diagram</a>	<a href="#">TimePicker</a>	<a href="#">Mention</a>
	<a href="#">Range Selector</a>	INPUTS	NAVIGATION
<a href="#">In-place Editor</a>	<a href="#">Smith Chart</a>		
<a href="#">RichTextEditor</a>	<a href="#">Sparkline Charts</a>		
<a href="#">PDF Viewer</a>	<a href="#">TreeMap</a>		
LAYOUT	<a href="#">Bullet Chart</a>	<a href="#">Input Mask</a>	<a href="#">Accordion</a>
	BUTTONS	<a href="#">Numeric TextBox</a>	<a href="#">Carousel</a>
<a href="#">Dialog</a>		<a href="#">Masked TextBox</a>	<a href="#">Context Menu</a>
<a href="#">ListView</a>		<a href="#">RadioButton</a>	<a href="#">Menu Bar</a>
<a href="#">Tooltip</a>		<a href="#">CheckBox</a>	<a href="#">Tabs</a>
<a href="#">Splitter</a>	<a href="#">ButtonGroup</a>	<a href="#">Color Picker</a>	<a href="#">Toolbar</a>
Dashboard	<a href="#">Dropdown Menu</a>	<a href="#">File Upload</a>	<a href="#">TreeView</a>
	<a href="#">Progress Button</a>	<a href="#">Range Slider</a>	<a href="#">File Manager</a>
	<a href="#">SplitButton</a>	<a href="#">Toggle Switch Button</a>	<a href="#">Breadcrumb</a>
	<a href="#">Chips</a>	<a href="#">Signature</a>	<a href="#">Pager</a>
	<a href="#">Speed Dial</a>	<a href="#">Rating</a>	NOTIFICATION
			<a href="#">Toast</a>
			<a href="#">Skeleton</a>
			<a href="#">Message</a>

## Right-To-Left support in Syncfusion Vue Components

Right-to-Left (RTL) support allows applications to effectively communicate with users who use languages that are written from right to left, such as Arabic, Hebrew, etc.

Syncfusion Vue UI components support for right-to-left (RTL) by setting the `enableRtl` property to `true`. This adds the class name `e-rtl` to the component element and renders all Syncfusion Vue components in a right-to-left direction.

### Enable RTL for all components

To enable Right-To-Left (RTL) support for all components, users can set the `enableRtl` property directly in their application. Here is an example code snippet using the Grid component:

#### **APP.VUE**

```

<template>
  <div id="app">
    <ejs-grid :dataSource='data' locale='ar-AE' :allowPaging='true'
:pageSettings='pageOptions'>
      <e-columns>
        <e-column field='OrderID' headerText='Order ID'
textAlign='Right' width=120></e-column>
        <e-column field='CustomerID' headerText='Customer ID'
width=150></e-column>
        <e-column field='ShipCity' headerText='Ship City'
width=150></e-column>
        <e-column field='ShipName' headerText='Ship Name'
width=150></e-column>
      </e-columns>
    </ejs-grid>
  </div>
</template>
<script>
import Vue from "vue";
import { L10n, setCulture, enableRtl } from '@syncfusion/ej2-base';
import { GridPlugin, Page } from "@syncfusion/ej2-vue-grids";
import { data } from './datasource.js';
// Enables Right to left alignment for all components
enableRtl(true);
setCulture('ar-AE');
L10n.load({
  'ar-AE': {
    'grid': {
      'EmptyRecord': 'لا سجلات لعرضها',
      'EmptyDataSourceError': 'يجب أن يكون مصدر البيانات فارغة في  
التحميل الأولي منذ يتم إنشاء الأعمدة من مصدر البيانات في أوتوجينيراتد عمود  
الشبكة'
    },
    'pager': {
      'currentPageInfo': '{0} صفحة 1 {من}',
      'totalItemsInfo': '({0} العناصر)',
      'firstPageTooltip': 'انتقل إلى الصفحة الأولى',
      'lastPageTooltip': 'انتقل إلى الصفحة الأخيرة',
      'nextPageTooltip': 'انتقل إلى الصفحة التالية',
      'previousPageTooltip': 'انتقل إلى الصفحة السابقة',
      'nextPagerTooltip': 'الذهاب إلى بيجر المقبل',
      'previousPagerTooltip': 'الذهاب إلى بيجر السابقة'
    }
  }
});
Vue.use(GridPlugin);
export default {
  data() {
    return {
      data: data,
      pageOptions: { pageSize: 7 }
    };
  },
  provide: {
    grid: [Page]
  }
}

```

```

</script>
<style>
  @import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/common/right-to-left-cs1" %}

### Enable RTL for an individual component

To enable Right-To-Left (RTL) support for an individual component, users can set the `enableRtl` property in the component's options. Here is an example code snippet using the Grid component:

#### APP.VUE

```

<template>
  <div id="app">
    <ejs-grid :dataSource='data' locale='ar-AE' :enableRtl='true'
    :allowPaging='true' :pageSettings='pageOptions'>
      <e-columns>
        <e-column field='OrderID' headerText='Order ID'
        textAlign='Right' width=120></e-column>
        <e-column field='CustomerID' headerText='Customer ID'
        width=150></e-column>
        <e-column field='ShipCity' headerText='Ship City'
        width=150></e-column>
        <e-column field='ShipName' headerText='Ship Name'
        width=150></e-column>
      </e-columns>
    </ejs-grid>
  </div>
</template>
<script>
import Vue from "vue";
import { L10n, setCulture } from '@syncfusion/ej2-base';
import { GridPlugin, Page } from "@syncfusion/ej2-vue-grids";
import { data } from './datasource.js';
setCulture('ar-AE');
// Enables Right to left alignment for all components
L10n.load({
  'ar-AE': {
    'grid': {
      'EmptyRecord': 'لا سجلات لعرضها',
      'EmptyDataSourceError': 'يجب أن يكون مصدر البيانات فارغة في  
التحميل الأولي منذ يتم إنشاء الأعمدة من مصدر البيانات في أوتوجينيراتد عمود  
الشبكة',
    },
    'pager': {
      'currentPageInfo': '{0} صفحة 1 {من}',
      'totalItemsInfo': '({0} العناصر)',
      'firstPageTooltip': 'انتقل إلى الصفحة الأولى',
      'lastPageTooltip': 'انتقل إلى الصفحة الأخيرة',
      'nextPageTooltip': 'انتقل إلى الصفحة التالية',
      'previousPageTooltip': 'انتقل إلى الصفحة السابقة',
      'nextPagerTooltip': 'الذهاب إلى بيجر المقبل',
      'previousPagerTooltip': 'الذهاب إلى بيجر السابقة'
    }
  }
});

```

```

});
Vue.use(GridPlugin);
export default {
  data() {
    return {
      data: data,
      pageOptions: { pageSize: 7 }
    };
  },
  provide: {
    grid: [Page]
  }
}
</script>
<style>
  @import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/common/right-to-left-cs2" %}

### State Persistence in Syncfusion Vue components

Syncfusion Vue UI components support persisting their state across page refreshes or navigation. To enable this feature, set the `enablePersistence` property to `true` for the desired component. This stores the component's state in the browser's `localStorage` object on the `unload` event of the page. For example, the `enablePersistence` property can be set for the Grid component, as shown in the following code snippet.

#### APP.VUE

```

<template>
  <div id="app">
    <ejs-grid :dataSource='data' :allowSorting='true'
      :enablePersistence='true' :allowPaging='true' :allowFiltering='true'
      height='210px'>
      <e-columns>
        <e-column field='OrderID' headerText='Order ID'
          textAlign='Right' width=120></e-column>
        <e-column field='CustomerID' headerText='Customer ID'
          width=150></e-column>
        <e-column field='ShipCity' headerText='Ship City'
          width=150></e-column>
        <e-column field='ShipName' headerText='Ship Name'
          width=150></e-column>
      </e-columns>
    </ejs-grid>
  </div>
</template>
<script>
import Vue from "vue";
import { GridPlugin, Sort, Page, Filter } from "@syncfusion/ej2-vue-grids";
import { data } from './datasource.js';
Vue.use(GridPlugin);
export default {
  data() {
    return {

```

```

        data: data
      };
    },
    provide: {
      grid: [Sort, Page, Filter]
    }
  }
</script>
<style>
  @import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/common/right-to-left-cs3" %}

### State Persistence supported components and properties

The following table illustrates the list of Syncfusion Vue components that are supported with state persistence and the properties that are stored in the `localStorage`.

<!-- markdownlint-disable MD033 -->

component Name	Properties
Grid	<ul style="list-style-type: none"> <li>Columns</li> <li>filterSettings</li> <li>searchSettings</li> <li>groupSettings</li> <li>pageSettings</li> <li>selectedRowIndex</li> <li>scrollPosition</li> </ul>
Accordion	<ul style="list-style-type: none"> <li>expandedIndices</li> </ul>
Tabs	<ul style="list-style-type: none"> <li>selectedItem</li> </ul>
Schedule	<ul style="list-style-type: none"> <li>currentView</li> <li>selectedDate</li> <li>scrollLeft</li> <li>scrollTop</li> </ul>
Kanban	<ul style="list-style-type: none"> <li>columns</li> <li>dataSource</li> <li>swimlaneToggleArray</li> </ul>
Chart	<ul style="list-style-type: none"> <li>zoomFactor</li> <li>zoomPosition</li> </ul>
Maps	<ul style="list-style-type: none"> <li>zoomSettings</li> </ul>
Pivot Table	<ul style="list-style-type: none"> <li>dataSourceSettings</li> </ul>

	<ul style="list-style-type: none"> <li>• pivotValues</li> <li>• gridSettings</li> <li>• chartSettings</li> <li>• displayOption</li> </ul>
TreeGrid	<ul style="list-style-type: none"> <li>• columns</li> <li>• pageSettings</li> <li>• searchSettings</li> <li>• filterSettings</li> <li>• selectedRowIndex</li> <li>• sortSettings</li> </ul>
Switch, Checkbox	<ul style="list-style-type: none"> <li>• checked</li> </ul>
RadioButton	<ul style="list-style-type: none"> <li>• checked</li> <li>• value</li> </ul>
ColorPicker, ListBox, In-placeEditor, RichTextEditor, Autocomplete, Calendar, ComboBox, DatePicker, DropDownList, MaskedTextBox, NumericTextBox, Textbox, TimePicker, Multiselect, DateTimePicker, Slider, Dropdown Tree	<ul style="list-style-type: none"> <li>• value</li> </ul>
QueryBuilder	<ul style="list-style-type: none"> <li>• rule</li> </ul>
Splitter	<ul style="list-style-type: none"> <li>• paneSettings</li> </ul>
DateRangePicker	<ul style="list-style-type: none"> <li>• startDate</li> <li>• endDate</li> <li>• value</li> </ul>
Uploader	<ul style="list-style-type: none"> <li>• filesData</li> </ul>
ListView	<ul style="list-style-type: none"> <li>• cssClass</li> <li>• enableRtl</li> <li>• htmlAttributes</li> <li>• enable</li> <li>• fields</li> <li>• animation</li> <li>• headerTitle</li> <li>• sortOrder</li> <li>• showIcon</li> <li>• height</li> <li>• width</li> <li>• showCheckBox</li> <li>• checkBoxPosition</li> </ul>

TreeView	<ul style="list-style-type: none"> <li>selectedNodes</li> <li>checkedNodes</li> <li>expandedNodes</li> </ul>
Dashboard Layout	<ul style="list-style-type: none"> <li>panels</li> </ul>
File Manager	<ul style="list-style-type: none"> <li>view</li> <li>path</li> <li>selectedItems</li> </ul>
Sidebar	<ul style="list-style-type: none"> <li>type</li> <li>position</li> <li>isOpen</li> </ul>

<!-- markdownlint-enable MD033 -->

Check out the following component's document to learn more about the state persistence.

- [Grid](#)
- [TreeGrid](#)
- [Pivot Table](#)
- [Gantt](#)
- [Kanban](#)
- [Schedule](#)
- [QueryBuilder](#)
- [Tabs](#)

## Event Binding

Syncfusion Vue UI components support Binding of both Custom and Native Events. For more information, refer official [Vue Documentation](#)

### Custom events

Custom events are the **Component** specific events provided by Syncfusion Vue components supported through **v-on** directive. The Syntax for Binding Custom Event is **v-on:event-name="function"**.

Refer the below code snippet for Binding of Custom Events.

,

```
// Custom Event Binding
```

```
<template>
```

```
<div id="calendar">
```

```
<ejs-calendar v-on:created="display" />
```

```
</div>
```

```
</template>
```

```

<script>
import Vue from 'vue';
import { CalendarPlugin } from '@syncfusion/ej2-vue-calendars';
Vue.use(CalendarPlugin);
export default {
  methods: {
    display: function() {
      window.alert('Calendar Created');
    }
  }
}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-vue-calendars/styles/material.css";
calendar {
  max-width: 250px;
  margin: 0 auto;
}
</style>
`

```

### Native events

Native events are the DOM events of Syncfusion Vue component's root element. `.native` modifier for `v-on` directive is used for binding these events. The Syntax for Binding Native Event is `v-on:event-name.native="function"`.

Refer the below code snippet for Binding Native Events.

```

//Native Event Binding
<template>
<div id="button">
  <ejs-button :content="name" v-on:click.native="clicked"></ejs-button>
</div>
</template>
<script>
import Vue from 'vue';

```



```

import { ButtonPlugin } from '@syncfusion/ej2-vue-buttons';
Vue.use(ButtonPlugin);
export default {
  data () {
    return {
      name: "Button"
    };
  },
  methods: {
    clicked: function() {
      window.alert('Button Clicked');
    }
  }
}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-vue-buttons/styles/material.css";
</style>

```

## Model Binding

### Integrating Vue model binding in Syncfusion Vue UI components

Syncfusion Vue UI components support model binding through the `v-model` directive. The model binding support in Syncfusion Vue components uses custom 'modelchanged' event, which is used to notify Vue that a model is changed.

- Syncfusion Vue UI components that are initialized from form elements support model binding.

Refer to the following code snippet.

#### **APP.VUE**

```

<template>
  <div id="app">
    <ejs-checkbox v-model="checked" label="EJ2 Vue Checkbox"
    :checked="checked" />
    <p>
      Checkbox State: <span id="checked-state" v-text="checked" ></span>
    </p>
  </div>
</template>
<script>

```

```

import Vue from 'vue';
import { CheckBoxPlugin } from '@syncfusion/ej2-vue-buttons';
Vue.use(CheckBoxPlugin);
export default {
  data () {
    return {
      checked: true
    }
  }
}
</script>
<style>
  @import "../node_modules/@syncfusion/ej2-vue-buttons/styles/material.css";
  #app {
    color: #008cff;
    height: 40px;
    left: 45%;
    position: absolute;
    top: 45%;
    width: 30%;
  }
</style>

```

{% previewsample "page.domainurl/code-snippet/common/model-cs1" %}

## Templates in Syncfusion Vue Components

Syncfusion Vue components are rendered with a pre-defined layout or structure that is used to define how the component should be rendered on the user interface. The user wants to customise the appearance of the component and add functionality that is specific to the needs of the application. Syncfusion Vue components have the option to achieve this using template support.

### Types of templates

Syncfusion Vue components have three types of templates, such as:

- [Slot template](#)
- [Inline template](#)
- [External template](#)

### Slot template

The Syncfusion Vue components do support [slots](#), which can help reduce the number of properties that need to be defined and increase the readability of the component. This is because using slots allows defining the content or behaviour of the component in the parent component rather than in the component's own code. This can make it easier to understand the purpose and functionality of the component at a glance and make the component more modular and flexible.

In the Vue component, the `v-slot` directive is used to define a slot template in the component's template where users can insert custom content. Refer to the following code sample.

### ~/SRC/APP.VUE

```

<ejs-grid ref="grid" :dataSource="ds">
  <e-columns>

```

```

<e-column field="OrderID" headerText="Order ID" width=120 textAlign="Right"
/>
<e-column field="CustomerName" headerText="Customer Name" width=150 />
<e-column field="ShipCountry" headerText="Ship Country" width=150
:template="'cTemplate'">
<template v-slot:cTemplate>
<ejs-button :content="ShipCountry"></ejs-button>
</template>
</e-column>
</e-columns>
</ejs-grid>

```

### Render scope

In a single-page application, there may be a need to access the parent component scope in the template. The slot content has access to the data scope of the parent component. To access the component's data source value inside the template, pass the props ({data}) to the `v-slot` directive. Expressions within the slot can access the component's data source.

```

`html
<template v-slot:templateName="{data}">
<ejs-button :content="data.ShipCountry"></ejs-button>
</template>
`

```

### Named slot

The Syncfusion Vue components support multiple templates. Each template is differentiated by its name. To render the slot content to the corresponding slot outlet, the name of each slot must map to the name of the corresponding property.

```

`html
<template v-slot:templateName></template>
`

```

When passing a slot to a component, ensure that the component's property value is of the "string" type.

An example of a Grid component sample with a named slot (cTemplate) template is shown below.

### ~/SRC/APP.VUE

```

<ejs-grid ref="grid" :dataSource="ds">
<e-columns>
<e-column field="OrderID" headerText="Order ID" width=120 textAlign="Right"
/>
<e-column field="CustomerName" headerText="Customer Name" width=150 />
<e-column field="ShipCountry" headerText="Ship Country" width=150
:template="'cTemplate'"/>
</e-columns>
<template v-slot:cTemplate="{data}">
<ejs-button :content="data.ShipCountry"></ejs-button>
</template>
</ejs-grid>

```

The slot template can also be used to insert content into nested tags within a component. In the below code example, `cTemplate` is rendered in the nested tag `<e-column>`.

#### ~/SRC/APP.VUE

```
<template>
  <div id="grid">
    <ejs-grid ref="grid" :dataSource="ds">
      <e-columns>
        <e-column field="OrderID" headerText="Order ID" width=120
textAlign="Right" />
        <e-column field="CustomerName" headerText="Customer Name" width=150
/>
        <e-column field="ShipCountry" headerText="Ship Country" width=150
:template="'cTemplate'">
          <template v-slot:cTemplate="{data}">
            <ejs-button :content="data.ShipCountry"></ejs-button>
          </template>
        </e-column>
      </e-columns>
    </ejs-grid>
  </div>
</template>
<script>
  import Vue from 'vue';
  import { ButtonPlugin } from '@syncfusion/ej2-vue-buttons';
  import { GridPlugin } from '@syncfusion/ej2-vue-grids';
  Vue.use(GridPlugin);
  Vue.use(ButtonPlugin);
  var empData = [
    { OrderID: 10248, ShipCountry: "France", CustomerName: "Paul Henriot" },
    { OrderID: 10249, ShipCountry: "Germany", CustomerName: "Karin Josephs"
  },
    { OrderID: 10250, ShipCountry: "Brazil", CustomerName: "Mario Pontes" },
    { OrderID: 10251, ShipCountry: "France", CustomerName: "Mary Saveley" }
  ];
  export default {
    data () { return { ds: empData } }
  }
</script>
<style>
  @import "../node_modules/@syncfusion/ej2-base/styles/material.css";
  @import "../node_modules/@syncfusion/ej2-buttons/styles/material.css";
  @import "../node_modules/@syncfusion/ej2-calendars/styles/material.css";
  @import "../node_modules/@syncfusion/ej2-dropdowns/styles/material.css";
  @import "../node_modules/@syncfusion/ej2-inputs/styles/material.css";
  @import "../node_modules/@syncfusion/ej2-navigations/styles/material.css";
  @import "../node_modules/@syncfusion/ej2-popups/styles/material.css";
  @import "../node_modules/@syncfusion/ej2-
splitbuttons/styles/material.css";
  @import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
</style>
```

{% previewsample "page.domainurl/code-snippet/common/slot-cs1" %}

### Inline template

The user can use the `Vue.component` method to add custom content to the template that can be used in the Syncfusion Vue components. The template elements can be added to `template` attribute of the `Vue.component` method. Refer to the below code snippet to create the template element using `Vue.component` method.

```
`js
var inlineTemplate = Vue.component("inlineTemplate", {
  template: '<ejs-button :content="$${data.ShipCountry}"></ejs-button>',
  data() { return { data: {} }; }
});
`
```

Create a template function that returns an object { key: 'template', value: 'importedTemplate' } to map this template to the Grid component.

```
`js
cTemplate: function() {
  return { template: inlineTemplate };
}
`
```

Now, the template function is assigned to the `template` property of the Grid component. Refer to the below example for the inline template.

### ~/SRC/APP.VUE

```
<template>
  <div id="grid">
    <ejs-grid ref="grid" :dataSource="ds">
      <e-columns>
        <e-column field="OrderID" headerText="Order ID" width=120
textAlign="Right" />
        <e-column field="CustomerName" headerText="Customer Name" width=150
/>
        <e-column field="ShipCountry" headerText="Ship Country" width=150
:template='cTemplate' />
      </e-columns>
    </ejs-grid>
  </div>
</template>
<script>
  import Vue from 'vue';
  import { ButtonPlugin } from '@syncfusion/ej2-vue-buttons';
  import { GridPlugin } from "@syncfusion/ej2-vue-grids";
  Vue.use(GridPlugin);
  Vue.use(ButtonPlugin);
  var inlineTemplate = Vue.component("inlineTemplate", {
    template: '<ejs-button :content="$${data.ShipCountry}"></ejs-button>',
    data() { return { data: {} }; }
  });
</script>
```

```

    });
    var empData = [
      { OrderID: 10248, ShipCountry: "France", CustomerName: "Paul Henriot" },
      { OrderID: 10249, ShipCountry: "Germany", CustomerName: "Karin Josephs"
    },
      { OrderID: 10250, ShipCountry: "Brazil", CustomerName: "Mario Pontes" },
      { OrderID: 10251, ShipCountry: "France", CustomerName: "Mary Saveley" }
    ];
    export default {
      data () {
        return {
          ds: empData,
          cTemplate: function () {
            return { template: inlineTemplate };
          }
        }
      }
    }
  }
</script>
<style>
  @import "../node_modules/@syncfusion/ej2-base/styles/material.css";
  @import "../node_modules/@syncfusion/ej2-buttons/styles/material.css";
  @import "../node_modules/@syncfusion/ej2-calendars/styles/material.css";
  @import "../node_modules/@syncfusion/ej2-dropdowns/styles/material.css";
  @import "../node_modules/@syncfusion/ej2-inputs/styles/material.css";
  @import "../node_modules/@syncfusion/ej2-navigations/styles/material.css";
  @import "../node_modules/@syncfusion/ej2-popups/styles/material.css";
  @import "../node_modules/@syncfusion/ej2-splitbuttons/styles/material.css";
  @import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/common/template-cs1" %}

### External template

The template elements can be defined in an external file (single-file component) and used in Syncfusion Vue components. Refer to the below code snippet to define template elements in `template.vue` file.

### ~/SRC/TEMPLATE.VUE

```

<template>
  <div class="button">
    <ejs-button :content="'${data.ShipCountry}'"> </ejs-button>
  </div>
</template>
<script>
import Vue from "vue";
import { ButtonPlugin } from '@syncfusion/ej2-vue-buttons';
Vue.use(ButtonPlugin);
export default {
  data () { return { data: {} } }
}
</script>

```

Import the `template.vue` file into the corresponding `App.vue` file as specified in the following code snippet.

```
`js
import externalTemplate from './template.vue'
`
```

Create a template function that returns an object { key: 'template', value: 'importedTemplate' } to map this template to the Grid component.

```
`js
cTemplate: function() {
return { template: externalTemplate };
}
`
```

Now, the template function is assigned to the `template` property of the Grid component. Refer to the below code snippet from `App.vue` file for the external template.

#### ~/SRC/APP.VUE

```
<template>
<div id="app">
<ejs-grid ref="grid" :dataSource="ds">
<e-columns>
<e-column headerText='Ship Country' width='150' textAlign='Center'
:template='cTemplate' />
<e-column field="OrderID" headerText="Order ID" width=120 textAlign="Right"
/>
<e-column field="CustomerName" headerText="Customer Name" width=150 />
</e-columns>
</ejs-grid>
</div>
</template>
<script>
import Vue from "vue";
import { GridPlugin } from "@syncfusion/ej2-vue-grids";
import externalTemplate from "./template.vue";
Vue.use(GridPlugin);
var empData = [
{ OrderID: 10248, ShipCountry: "France", CustomerName: "Paul Henriot" },
{ OrderID: 10249, ShipCountry: "Germany", CustomerName: "Karin Josephs" },
{ OrderID: 10250, ShipCountry: "Brazil", CustomerName: "Mario Pontes" },
{ OrderID: 10251, ShipCountry: "France", CustomerName: "Mary Saveley" }
];
export default {
name: 'App',
data() {
return {
ds: empData,
cTemplate: function() {
return { template: externalTemplate };
}
}
}
```

```

};
}
};
</script>
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-buttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-calendars/styles/material.css";
@import "../node_modules/@syncfusion/ej2-dropdowns/styles/material.css";
@import "../node_modules/@syncfusion/ej2-inputs/styles/material.css";
@import "../node_modules/@syncfusion/ej2-navigations/styles/material.css";
@import "../node_modules/@syncfusion/ej2-popups/styles/material.css";
@import "../node_modules/@syncfusion/ej2-splitbuttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
</style>

```

### External modules in templates

Syncfusion provides the option to use external modules in template content. To use the external modules in the template, add those modules to the `plugins` property of the Vue component. For example, the "i18n" module is added to the `plugins` property of the Grid component. Refer to the below code snippet.

#### ~/SRC/APP.VUE

```

<template>
<h3>Grid component</h3>
<ejs-grid height='210px' :plugins="modules"></ejs-grid>
</template>
<script>
import { GridPlugin } from "@syncfusion/ej2-vue-grids";
export default {
  components: {
    "ejs-grid": GridComponent,
  },
  data() {
    return { modules: [i18n] };
  }
};
</script>

```

Below is the example code to define `i18n` external module in the Vue 3 application.

#### ~/MAIN.JS

```

import { createApp } from 'vue'
import App from './App.vue'
import { createI18n } from "vue-i18n";
const messages = {
  en: {
    message: {
      customer: "Customer Name",
    },
  },
  ja: {
    message: {

```



```

customer: "顧客名",
},
},
};
export const i18n = createI18n({
  legacy: false,
  locale: "ja",
  fallbackLocale: "en",
  messages,
});
createApp(App).use(i18n).mount('#app')

```

Below is the example code to use `i18n` external module in the Grid component template using `plugins` property.

### ~/SRC/APP.VUE

```

{% raw %}
<template>
<ejs-grid ref="grid" :dataSource="ds" :plugins="modules">
<e-columns>
<e-column field="OrderID" headerText="Order ID" width=120 textAlign="Right"
/>
<e-column headerText='Customer Name' width=150 :template="'cTemplate'">
<template v-slot:cTemplate={data}>
<div>{{ $t("message.customer") }} - {{data.CustomerName}}</div>
</template>
</e-column>
</e-columns>
</ejs-grid>
</template>
<script>
import { i18n } from "./main";
import { GridComponent, ColumnsDirective, ColumnDirective } from
"@syncfusion/ej2-vue-grids";
var empData = [
{ OrderID: 10248, ShipCountry: "France", CustomerName: "Paul Henriot" },
{ OrderID: 10249, ShipCountry: "Germany", CustomerName: "Karin Josephs" },
{ OrderID: 10250, ShipCountry: "Brazil", CustomerName: "Mario Pontes" },
{ OrderID: 10251, ShipCountry: "France", CustomerName: "Mary Saveley" }
];
export default {
  name: "App",
  components: {
    "ejs-grid": GridComponent,
    "e-columns": ColumnsDirective,
    "e-column": ColumnDirective,
  },
  data() {
    return {
      modules: [i18n],
      ds: empData,
    };
  },
};
</script>

```

```

<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-buttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-calendars/styles/material.css";
@import "../node_modules/@syncfusion/ej2-dropdowns/styles/material.css";
@import "../node_modules/@syncfusion/ej2-inputs/styles/material.css";
@import "../node_modules/@syncfusion/ej2-navigations/styles/material.css";
@import "../node_modules/@syncfusion/ej2-popups/styles/material.css";
@import "../node_modules/@syncfusion/ej2-splitbuttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
</style>
{% endraw %}

```

### Provide/Inject in templates

In Vue, **provide** and **inject** options are used to share data between components that are not directly related through a parent-child relationship.

Syncfusion components can use these **provide** and **inject** options in templates. It allows to pass data from a parent component to its template components without having to pass props down the component tree. Instead, the parent component provides the data, and the child components inject it.

To provide data from a parent component to its template, use the **provide** option. The provide option is an object that contains the data to provide. The keys in the object are the names of the properties, and the values are the data to provide.

In this below example, the parent component provides the content property with the value of **Update** in **App.vue** file.

#### ~/SRC/APP.VUE

```

<template>
<div id="grid">
<ejs-grid ref="grid" :dataSource="ds">
<e-columns>
<e-column field="OrderID" headerText="Order ID" width=120 textAlign="Right"
/>
<e-column field="CustomerName" headerText="Customer Name" width=150 />
<e-column field="ShipCountry" headerText="Ship Country" width=150
:template="'cTemplate'">
<template v-slot:cTemplate={data}>
<div>{{data.ShipCountry}} <MyTemplate /></div>
</template>
</e-column>
</e-columns>
</ejs-grid>
</div>
</template>
<script>
import Vue from 'vue';
import { GridPlugin } from "@syncfusion/ej2-vue-grids";
import MyTemplate from "./MyTemplate.vue";
Vue.use(GridPlugin);
var empData = [
{ OrderID: 10248, ShipCountry: "France", CustomerName: "Paul Henriot" },
{ OrderID: 10249, ShipCountry: "Germany", CustomerName: "Karin Josephs" },

```

```

{ OrderID: 10250, ShipCountry: "Brazil", CustomerName: "Mario Pontes" },
{ OrderID: 10251, ShipCountry: "France", CustomerName: "Mary Saveley" }
];
export default {
  components: {
    MyTemplate
  },
  data () {
    return {
      ds: empData
    }
  },
  provide: { content: 'Update' }
}
</script>

```

To inject data provided by a parent component, use the `inject` option. The `inject` option is an array or an object that contains the names of the properties to inject.

In this below example, the child template component injects content property using the `inject` option, and displays its value using an interpolation directive (`{{ raw }}{{ content }}{% endraw %}`) in **MyTemplate.vue** file.

#### ~/SRC/MYTEMPLATE.VUE

```

{% raw %}
<template>
<ejs-button>{{ content }}</ejs-button>
</template>
<script>
import Vue from 'vue';
import { ButtonPlugin } from '@syncfusion/ej2-vue-buttons';
Vue.use(ButtonPlugin);
export default {
  name: 'MyTemplate',
  data () {
    return {}
  },
  inject: ['content']
}
</script>
{% endraw %}

```

## Animation in Vue

The **Animation** is used to perform animation effects on HTML elements by running a sequence of frames. It can be used to enhance the user experience.

Syncfusion [Animation](#) library supports animating the HTML elements using the [animate](#) method. This method adds the `e-animate`, `e-animation-id` attributes, and CSS style to the HTML element and removes them after the animation effect is completed.

### Animation effects

An animation effect refers to the visual change that occurs over a period of time in HTML elements. The [Animation](#) library supports different types of animation [effects](#). The [name](#) property is used to specify the animation effect of an animation.

Here is an example code snippet using the `FadeOut` and `ZoomOut` animation effects:

#### APP.VUE

```
<template>
  <div id='container'>
    <div id='element1'></div>
    <div id='element2'></div>
  </div>
</template>
<script>
  import { Animation } from '@syncfusion/ej2-base';
  import Vue from "vue";
  export default {
    mounted: function () {
      var animation = new Animation();
      animation.animate('#element1', { name: 'FadeOut' });
      animation.animate('#element2', { name: 'ZoomOut' });
    }
  }
</script>
<style>

#element1, #element2 {
  background: #333333;
  border: 1px solid #cecece;
  box-sizing: border-box;
  float: left;
  height: 100px;
  width: 100px;
}
#element2 {
  margin-left: 20px;
}
</style>
```

{% previewsample "page.domainurl/code-snippet/common/animation-multiple-cs4" %}

### Animation duration

Animation [duration](#) is the animation property that specifies the length of time that an animation should take to complete. The animation duration is specified in milliseconds (ms) and determines the total amount of time that an animation should run.

For example, if an animation has a duration of 2 seconds, it will take 2 seconds to complete from start to finish. The duration of an animation affects the overall pace of the animation and can be adjusted to match the desired speed and style of the animation.

The value of the animation duration can be adjusted to change the speed of the animation, with shorter durations resulting in faster animations and longer durations resulting in slower animations.

Here is an example code snippet using the animation effects with a duration of `5000` milliseconds:

**APP.VUE**

```

<template>
  <div id='container'>
    <div id='element1'></div>
    <div id='element2'></div>
  </div>
</template>
<script>
  import { Animation } from '@syncfusion/ej2-base';
  import Vue from "vue";
  export default {
    mounted: function () {
      var animation = new Animation({ duration: 5000 });
      animation.animate('#element1', { name: 'FadeOut' });
      animation.animate('#element2', { name: 'ZoomOut' });
    }
  }
</script>
<style>

#element1, #element2 {
  background: #333333;
  border: 1px solid #cecece;
  box-sizing: border-box;
  float: left;
  height: 100px;
  width: 100px;
}
#element2 {
  margin-left: 20px;
}
</style>

```

{% previewsample "page.domainurl/code-snippet/common/animation-multiple1-cs1" %}

**Animation delay**

The animation [delay](#) is the animation property that specifies the amount of time to wait before starting an animation. The animation delay is specified in milliseconds (ms) and determines the amount of time that should elapse before an animation begins.

For example, if an animation has a delay of 2 seconds, it will wait for 2 seconds before starting. This can be useful in creating more complex animations, where multiple elements are animated in sequence, or in creating animations that start only after a user interaction has taken place.

Here is an example code snippet using the animation effects with a delay of 2000 milliseconds:

**APP.VUE**

```

<template>
  <div id='container'>
    <div id='element1'></div>
    <div id='element2'></div>
  </div>
</template>
<script>

```

```

import { Animation } from '@syncfusion/ej2-base';
import Vue from "vue";
export default {
  mounted: function () {
    var animation = new Animation({ delay: 2000, duration: 5000 });
    animation.animate('#element1', { name: 'FadeOut' });
    animation.animate('#element2', { name: 'ZoomOut' });
  }
}
</script>
<style>

#element1, #element2 {
  background: #333333;
  border: 1px solid #cecece;
  box-sizing: border-box;
  float: left;
  height: 100px;
  width: 100px;
}
#element2 {
  margin-left: 20px;
}
</style>

```

{% previewsample "page.domainurl/code-snippet/common/animation-multiple2-cs1" %}

### Enable or disable animation globally

Enable or disable animation for all Vue components globally by using the `setGlobalAnimation` method with one of the below options:

- `GlobalAnimationMode.Enable` - Enables the animation for all components, regardless of the individual component's animation settings.
- `GlobalAnimationMode.Disable` - Disables the animation for all components, regardless of the individual component's animation settings.
- `GlobalAnimationMode.Default` - Animation is enabled or disabled based on the component's animation settings.

In the below code snippet, animation is disabled.

### ~/SRC/MAIN.JS

```

import { GlobalAnimationMode, setGlobalAnimation } from "@syncfusion/ej2-base";
setGlobalAnimation(GlobalAnimationMode.Disable);

```

`setGlobalAnimation` method controls script-level animations only, and it is not applicable for direct CSS-level animations (animations defined from CSS classes or properties).

### Getting started with Localization

Localization library allows you to localize the text content of the Syncfusion React UI Components. This is useful if you want to display the UI in a language other than English.

### Loading translations

To load a translation object in your application, you can use the load function from the @syncfusion/ej2-base module. This function takes an object that contains the translations for various languages, with the keys being the language codes and the values being the translation objects.

### APP.VUE

```
<template>
  <div id="app">
    <ejs-grid :dataSource='data' locale='de-DE' :allowGrouping='true'
:allowPaging='true' :pageSettings='pageOptions' height='220px'>
      <e-columns>
        <e-column field='OrderID' headerText='Order ID'
textAlign='Right' width=120></e-column>
        <e-column field='CustomerID' headerText='Customer ID'
width=150></e-column>
        <e-column field='ShipCity' headerText='Ship City'
width=150></e-column>
        <e-column field='ShipName' headerText='Ship Name'
width=150></e-column>
      </e-columns>
    </ejs-grid>
  </div>
</template>
<script>
import Vue from "vue";
import { L10n, setCulture } from '@syncfusion/ej2-base';
import { GridPlugin, Page, Group } from "@syncfusion/ej2-vue-grids";
import { data } from './datasource.js';
setCulture('de-DE');
L10n.load({
  'de-DE': {
    'grid': {
      'EmptyRecord': 'Keine Aufzeichnungen angezeigt',
      'GroupDropArea': 'Ziehen Sie einen Spaltenkopf hier, um die
Gruppe ihre Spalte',
      'UnGroup': 'Klicken Sie hier, um die Gruppierung aufheben',
      'EmptyDataSourceError': 'DataSource darf bei der Erstaustellung
nicht leer sein, da Spalten aus der dataSource im AutoGenerate
Spaltenraster',
      'Item': 'Artikel',
      'Items': 'Artikel'
    },
    'pager': {
      'currentPageInfo': '{0} von {1} Seiten',
      'totalItemsInfo': '({0} Beiträge)',
      'firstPageTooltip': 'Zur ersten Seite',
      'lastPageTooltip': 'Zur letzten Seite',
      'nextPageTooltip': 'Zur nächsten Seite',
      'previousPageTooltip': 'Zurück zur letzten Seit',
      'nextPagerTooltip': 'Zum nächsten Pager',
      'previousPagerTooltip': 'Zum vorherigen Pager'
    }
  }
});
Vue.use(GridPlugin);
export default {
```

```

data() {
  return {
    data: data,
    pageOptions: { pageSize: 7 }
  };
},
provide: {
  grid: [Page, Group]
}
}
</script>
<style>
  @import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/common/intl-cs1" %}

### Changing current locale

The current locale can be changed for all the Syncfusion React UI Components in your application by invoking `setCulture` function with your desired culture name.

`ts

```
import {L10n, setCulture} from '@syncfusion/ej2-base';
```

```
L10n.load({
```

```
'fr-BE': {
```

```
'Button': {
```

```
'id': 'Numéro de commande',
```

```
'date': 'Date de commande'
```

```
}
```

```
}
```

```
});
```

```
setCulture('fr-BE');
```

```
`
```

Note: Before changing a culture globally, you need to ensure that locale text for the concerned culture is loaded through `L10n.load` function.

### Internationalization

The `Internationalization` library provides support for formatting and parsing date and number objects using the official [Unicode CLDR](#) JSON data. The `en-US` locale is set as default *culture* and `USD` is set as default *currencyCode* for all Syncfusion Vue UI Components.

### Loading culture data

It requires the following CLDR data to be load using `loadCldr` function for cultures other than `en-US`.

| File Name | Path |



```
| ----- | ----- |
| ca-gregorian | cldr/main/en/ca-gregorian.json |
| timeZoneNames | cldr/main/en/timeZoneNames.json |
| numbers | cldr/main/en/numbers.json |
| numberingSystems | cldr/supplemental/numberingSystems.json |
| currencies | cldr/main/en/currencies.json |
```

Note: For `en`, dependency files are already loaded in the library.

#### Installing CLDR data

CLDR data is available as npm package. So, we can install it through below command to our package.

```
`bash
npm install cldr-data
```

```
,
```

#### Binding to i18n library

```
`ts
import { loadCldr } from '@syncfusion/ej2-base';
loadCldr(enNumberData, entimeZoneData);
```

```
,
```

#### Changing Global Culture and Currency Code

To set the default culture and the currency code for all Syncfusion Vue UI Components, you can use the methods `setCulture` for setting the default locale and `setCurrencyCode` for setting the currency code.

```
`ts
import { setCulture, setCurrencyCode } from '@syncfusion/ej2-base';
setCulture('ar');
setCurrencyCode('QAR');
```

```
,
```

Note: If global culture is not set, then `en-US` is set as the default locale, and `USD` is set as the default currency code.

#### Manipulating numbers

```
<!-- markdownlint-disable MD024 -->
```

#### Supported format string

```
<!-- markdownlint-disable MD024 -->
```

Based on the [NumberFormatOptions](#) number formatting and parsing operations are processed. You need to specify some or all of the following properties mentioned below table.

No	Properties	Description
---	---	---

| 1 | **format** | Denotes the format to be set .Possible values are <br /> 1. **N** - denotes numeric type. <br /> 2. **C** - denotes currency type. <br /> 3. **P** - denotes percentage type. <br /> E.g: <br /> `formatNumber( 1234344 ,{format:'N4'})`. <br /> <br /> Note: If no format is specified it takes numeric as default format type. |

| 2 | **minimumFractionDigits** | Indicates the minimum number of fraction digits . Possible values are 0 to 20. |

| 3 | **maximumFractionDigits** | Indicates the maximum number of fraction digits. Possible values are 0 to 20. |

| 4 | **minimumSignificantDigits** | Indicates the minimum number of significant digits. Possible values are 1 to 21. <br /> Note: If **minimumSignificantDigits** is given it is mandatory to give **maximumSignificantDigits** |

| 5 | **maximumSignificantDigits** | Indicates the maximum number of significant digits. . Possible values are 1 to 21. <br /> **Note:** If **maximumSignificantDigits** is given it is mandatory to give **minimumSignificantDigits** |

| 6 | **useGrouping** | Indicates whether to enable the group separator or not . By default grouping value will be true. |

| 7 | **minimumIntegerDigits** | Indicates the minimum number of the integer digits to be placed in the value. Possible values are 1 to 21. |

| 8 | **currency** | Indicates the currency code which needs to be considered for the currency formatting. |

Note: The **minimumIntegerDigits**, **minimumFractionDigits** and **maximumFractionDigits** are categorized

as group one,

**minimumSignificantDigits** and **maximumSignificantDigits** are categorized as group two.

If group two properties are defined, then the group one properties will be ignored.

#### *Custom number formatting and parsing*

Custom number formatting and parsing can also be achieved by directly specifying the pattern in the **format** property of **NumberFormatOptions**. One or more of the custom format specifiers listed in the table below can be used to create custom number format.

Specifier	Description	Input	Format Output
0	Replaces the zero with the corresponding digit if one is present. Otherwise, zero appears in the result string.	<code>instance.formatNumber(123,{format: '0000' })</code>	<code>'0123'</code>
#	Replaces the "#" symbol with the corresponding digit if one is present; otherwise, no digit appears in the result string.	<code>instance.formatNumber(1234,{format: '####' })</code>	<code>'1234'</code>
.	Denotes the number of digits allowed after the decimal points if it's not specified then no need to specify decimal point values.	<code>instance.formatNumber(546321,{format: '###0.##0#' })</code>	<code>'546321.000'</code>

| % | Percent specifier denotes the percentage type format. | `instance.formatNumber(1,{format: '0000 %'})` | `'0100 %'` |

| \$ | Denotes the currency type format based on the global currency code specified. | `instance.formatNumber(13,{format: '$ ###.00'})` | `'$ 13.00'` |

| ; | Denotes separate formats for positive, negative and zero values. | `instance.formatNumber(-120,{format: '###.##;(###.00);-0'})` | `'(120.00)'` |

| 'String' (single Quotes) | Denotes the characters enclosed within single Quote(') to be replaced in the resultant string. | `instance.formatNumber(-123.44,{format: "####.## '@"})` | `'123.44 @'` |

Note: If a custom format pattern is specified, other [NumberFormatOptions](#) properties will not be considered.

### Number Parsing

``getNumberParser``

The [getNumberParser](#) method, which will return a function that parses a given string based on the [NumberFormatOptions](#) specified.

### APP.VUE

```
<template>
<div class="result"></div>
</template>
<script>
import Vue from 'vue';
import { Internationalization } from '@syncfusion/ej2-base';
export default {
  mounted:function() {
    var intl = new Internationalization();
    var nParser = intl.getNumberParser({ format:'P2' , useGrouping: false});
    var val = nParser('123567.45%');
    document.querySelector('.result').innerHTML = val + '';
  }
}
</script>
<style>
</style>
```

{% previewsample "page.domainurl/code-snippet/common/intl-parseNumber1-cs1" %}

``parseNumber``

The [parseNumber](#) method, which takes two arguments, the string value and [NumberFormatOptions](#) and returns the numeric value.

### APP.VUE

```
<template>
<div class="result"></div>
</template>
<script>

import Vue from 'vue';
import { Internationalization } from '@syncfusion/ej2-base';
```

```
export default {
  mounted:function() {
    var intl = new Internationalization();
    var val = intl.parseNumber('$01,234,545.650', { format: 'C3', currency:
'USD', minimumIntegerDigits: 8 });
    document.querySelector('.result').innerHTML = val + '';
  }
}
</script>
<style>
</style>
```

{% previewsample "page.domainurl/code-snippet/common/intl-parseNumber-cs1" %}

### Number formatting

#### `getNumberFormat`

The [getNumberFormat](#) method will return a function that formats a given number based on the [NumberFormatOptions](#) specified.

### APP.VUE

```
<template>
  <div class="result"></div>
</template>
<script>
import Vue from 'vue';
import { Internationalization } from '@syncfusion/ej2-base';
export default {
  mounted:function() {
    var intl = new Internationalization();
    var nFormatter = intl.getNumberFormat({ skeleton: 'C3', currency:
'USD',minimumIntegerDigits:8});
    var formattedValue = nFormatter(1234545.65)
    document.querySelector('.result').innerHTML = formattedValue;
  }
}
</script>
<style>
</style>
```

{% previewsample "page.domainurl/code-snippet/common/intl-getNumber-format-cs1" %}

#### `formatNumber`

The [formatNumber](#) method, which takes two arguments, a numeric value and [NumberFormatOptions](#) and returns the formatted string.

### APP.VUE

```
<template>
  <div class="result"> </div>
</template>
<script>
import Vue from 'vue';
import { Internationalization } from '@syncfusion/ej2-base';
export default {
  mounted: function() {
```

```

var intl = new Internationalization();
var formattedString = intl.formatNumber(12345.65, { format: 'C5' ,
useGrouping: false,
minimumSignificantDigits:1, maximumSignificantDigits:3 });
document.querySelector('.result').innerHTML = formattedString;
}
}
</script>
<style>
</style>

```

{% previewsample "page.domainurl/code-snippet/common/intl-getNumber-format-cs2" %}

## Manipulating DateTime

### Supported format string

Date formatting and parsing operations are performed based on the [DateFormatOptions](#). You need to specify some or all of the following properties mentioned in the table below.

Options	Descriptions
---------	--------------

---	---	---
-----	-----	-----

Type	It specifies the type of format to be used supported types .
1. <b>date</b>	2. <b>dateTime</b>
3. <b>time</b>	Based on the type specified the supported skeletons are given below.
1. short	2. medium, 3. long 4. full
E.g: <code>formatDate(new Date(), {type: 'date', skeleton:medium})</code>	
Note: If no type is specified then <b>date</b> type is set by default.	

skeleton	Specifies the format in which the <code>dateTime</code> format will process
----------	---

<!-- markdownlint-disable MD036 -->

### Date type skeletons

skeleton	Option input	Format	Output
----------	--------------	--------	--------

---	---	---
-----	-----	-----

short	{type: 'date', skeleton:'short'}	11/4/16
-------	----------------------------------	---------

medium	{type: 'date', skeleton:'medium'}	Nov 4, 2016
--------	-----------------------------------	-------------

long	{type: 'date', skeleton:'long'}	November 4, 2016
------	---------------------------------	------------------

full	{type: 'date', skeleton:'full'}	Friday, November 4, 2016
------	---------------------------------	--------------------------

### Time type skeletons

skeleton	Option input	Format	Output
----------	--------------	--------	--------

---	---	---
-----	-----	-----

short	{type: 'time', skeleton:'short'}	1:03 PM
-------	----------------------------------	---------

medium	{type: 'time', skeleton:'medium'}	1:03:04 PM
--------	-----------------------------------	------------

Long	{type: 'time', skeleton:'long'}	1:03:04 PM GMT+5
------	---------------------------------	------------------

full	{type: 'time', skeleton:'full'}	1:03:04 PM GMT+05:30
------	---------------------------------	----------------------

### DateTime type skeletons

	Skeleton	Option input	Format Output
	---	---	---
short	{type: 'dateTime', skeleton: 'short'}		11/4/16, 1:03 PM
medium	{type: 'dateTime', skeleton: 'medium'}		Nov 4, 2016, 1:03:04 PM
Long	{type: 'dateTime', skeleton: 'long'}		November 4, 2016 at 1:03:04 PM GMT+5
full	{type: 'dateTime', skeleton: 'full'}		Friday, November 4, 2016 at 1:03:04 PM GMT+05:30

### Additional skeletons

Apart from the standard date type formats, additional formats are supported by using the additional skeletons given in the below table.

	skeleton	Option input	Format Output
	---	---	---
d	{skeleton: 'd'}		7
E	{skeleton: 'E'}		Mon
Ed	{skeleton: 'Ed'}		7 Mon
Ehm	{skeleton: 'Ehm'}		Mon 12:43 AM
EHm	{skeleton: 'EHm'}		Mon 12:43
Ehms	{skeleton: 'Ehms'}		Mon 2:45:23 PM
EHms	{skeleton: 'EHms'}		Mon 12:45:45
Gy	{skeleton: 'Gy'}		2016 AD
GyMMM	{skeleton: 'GyMMM'}		: Nov 2016 AD
GyMMMd	{skeleton: 'GyMMMd'}		Nov 7, 2016 AD
GyMMMEd	{skeleton: 'GyMMMEd'}		Mon, Nov 7, 2016 AD
h	{skeleton: 'h'}		12 PM
H	{skeleton: 'H'}		12
hm	{skeleton: 'hm'}		12:59 PM
Hm	{skeleton: 'Hm'}		12:59
hms	{skeleton: 'hms'}		12:59:13 PM
Hms	{skeleton: 'Hms'}		12:59:13
M	{skeleton: 'M'}		11
Md	{skeleton: 'Md'}		11/7
MEd	{skeleton: 'hms'}		Mon, 11/7
MMM	{skeleton: 'MMM'}		Nov
MMMEd	{skeleton: 'MMMEd'}		Mon, Nov 7

```
| MMMd | {skeleton:'MMMEd'} | Nov 7 |
| ms | {skeleton:'ms'} | 59:13 |
| y | {skeleton:'y'} | 2016 |
| yM | {skeleton:'yM'} | 11/2016 |
| yMd | {skeleton:'yMd'} | 11/7/2016 |
| yMEd | {skeleton:'yMEd'} | Mon, 11/7/2016 |
| yMMM | {skeleton:'yMMM'} | Nov 2016 |
| yMMMd | {skeleton:'yMMMd'} | Nov 7, 2016 |
| yMMMEd | {skeleton:'yMMMEd'} | Mon, Nov 7, 2016 |
| yMMM | {skeleton:'yMMM'} | Nov 2016 |
```

Note: Culture specific format skeletons are also supported.

### *Custom formats*

To use the custom date and time formats, we need to specify the date/time pattern directly in the *format* property.

A custom format string must contain one or more of the following standard date/time symbols.

Symbols	Description
-----	Denotes the era in the date
y	Denotes the year.
M / L	Denotes month.
E / c	Denotes the day of week.
d	Denotes the day of month.
h / H	Denotes the hour. <i>h</i> for 12 hour and <i>H</i> for 24 hours format.
m	Denotes minutes.
s	Denotes seconds.
a	Denotes the am/pm designator it will only be displayed if hour is specified in the h format.
z	Denotes the time zone.
' (single quotes)	To display words in the formatted date you can specify the words with in the single quotes

### **Custom format example**

```
`ts
import Vue from 'vue';
import { Internationalization } from '@syncfusion/ej2-base';
var intl = new Internationalization();
```

```
var formattedString = intl.formatDate(new Date('1/12/2014 10:20:33'), { format: '\year:\y' \month:\MM' });
```

```
//Output: "year:2014 month:01"
```

```
,
```

Note: If the format property is given as an option, other properties are not considered.

```
<!-- markdownlint-enable MD036 -->
```

### *Date parsing*

``getDateParser``

The [getDateParser](#) method will return a function that parses a given string based on the [DateFormatOptions](#) specified.

### **APP.VUE**

```
<template>
  <div class="result"> </div>
</template>
<script>
import Vue from 'vue';
import {Internationalization} from '@syncfusion/ej2-base';
export default {
  mounted:function() {
    var intl = new Internationalization();
    var dParser = intl.getDateParser({skeleton: 'full', type: 'dateTime'});
    var val = dParser('Friday, November 4, 2016 at 1:03:04 PM GMT+05:30');
    document.querySelector('.result').innerHTML = val.toString();
  }
}
</script>
<style>
</style>
```

```
{% previewsample "page.domainurl/code-snippet/common/intl-parseDate-cs1" %}
```

``parseDate``

The date object is returned by the [parseDate](#) method, which takes two arguments, a string value and [DateFormatOptions](#).

### **APP.VUE**

```
<template>
  <div class="result"></div>
</template>
<script>
import Vue from 'vue';
import {Internationalization} from '@syncfusion/ej2-base';
export default {
  mounted:function() {
    var intl = new Internationalization();
    var val = intl.parseDate('11/2016',{skeleton: 'yM'});
    document.querySelector('.result').innerHTML = val.toString();
  }
}
```



```

</script>
<style>
</style>

```

{% previewsample "page.domainurl/code-snippet/common/intl-parseDate1-cs1" %}

### *Date Formatting*

#### ``getDateFormat``

The [getDateFormat](#) method, which will return a function that formats a given date object based on the [DateFormatOptions](#) specified.

#### **APP.VUE**

```

<template>
  <div class="result"></div>
</template>
<script>
import Vue from 'vue';
import { Internationalization } from '@syncfusion/ej2-base';
export default {
  mounted:function() {
    var intl = new Internationalization();
    var dFormatter = intl.getDateFormat({ skeleton: 'full', type: 'dateTime' });
    var formattedString = dFormatter(new Date('1/12/2014 10:20:33'));
    document.querySelector('.result').innerHTML = formattedString;
  }
}
</script>
<style>
</style>

```

{% previewsample "page.domainurl/code-snippet/common/intl-formatDate-cs1" %}

#### ``formatDate``

The [formatDate](#) method, which takes two arguments, the date object and [DateFormatOptions](#), returns the formatted string.

#### **APP.VUE**

```

<template>
  <div class="result"></div>
</template>
<script>
import Vue from 'vue';
import { Internationalization } from '@syncfusion/ej2-base';
export default {
  mounted:function() {
    var intl = new Internationalization();
    var date = new Date();
    var formattedString = intl.formatDate(new Date('1/12/2014 10:20:33'), {
      skeleton: 'GyMMM' });
    document.querySelector('.result').innerHTML = formattedString;
  }
}
</script>

```

```
<style>
</style>
```

```
{% previewsample "page.domainurl/code-snippet/common/intl-formatDate-cs2" %}
```

```
<!-- markdownlint-enable MD036 -->
```

## Drag and Drop in Vue

Drag and drop is a feature of a user interface that allows users to select an item or items and then move them to a different location or onto another interface element by "dragging" the selected item(s) with a pointing device (such as a mouse) and then "dropping" them at the desired location.

Syncfusion Vue components support drag and drop feature through two libraries. These are [Draggable](#) and [Droppable](#).

### Draggable

The Syncfusion's [Draggable](#) library allows users to make any DOM element draggable by passing it as a parameter to the [Draggable](#) constructor. This can be useful for creating interactive and user-friendly interfaces, such as allowing a user to reorder items in a list by dragging them. The below code snippet enables the draggable functionality for the specific DOM element passed to the [Draggable](#) constructor.

### APP.VUE

```
<template>
  <div id='container'>
    <div id='element1'>
      <p>Draggable Element </p>
    </div>
  </div>
</template>
<script>
  import { Draggable } from '@syncfusion/ej2-base';
  import Vue from "vue";
  export default {
    mounted: function () {
      var dragElement = document.getElementById('element1');
      var draggable = new Draggable(dragElement, { clone: false });
    }
  }
</script>
<style>
  #element1,
  .helper {
    height: 100px;
    width: 150px;
    border: 1px solid #cecece;
    cursor: move;
    user-select: none;
    color: #6a77a7;
    touch-action: none;
  }
  p {
    padding-top: 23px;
    text-align: center;
  }
</style>
```

```

.helper {
  opacity: 0.6;
}
.select {
  border: 1px solid #cccccc;
  background: #ededed;
}
</style>

```

{% previewsample "page.domainurl/code-snippet/common/draggable-default-cs1" %}

### *Clone draggable element*

Syncfusion provides the option to create a clone of a draggable element while the user is dragging it. It can be achieved by setting the [clone](#) property to `true`. Here's an example of how to create a clone of a draggable element.

### **APP.VUE**

```

<template>
  <div id='container'>
    <div id='element1'>
      <p>Draggable Element </p>
    </div>
  </div>
</template>
<script>
  import { Draggable } from '@syncfusion/ej2-base';
  import Vue from "vue";
  export default {
    mounted: function () {
      var dragElement = document.getElementById('element1');
      var draggable = new Draggable(dragElement, { clone: true });
    }
  }
</script>
<style>
  #element1,
  .helper {
    height: 100px;
    width: 150px;
    border: 1px solid #cecece;
    cursor: move;
    user-select: none;
    color: #6a77a7;
    touch-action: none;
  }
  p {
    padding-top: 23px;
    text-align: center;
  }
  .helper {
    opacity: 0.6;
  }
  .select {

```

```

        border: 1px solid #cccccc;
        background: #ededed;
    }
</style>

```

{% previewsample "page.domainurl/code-snippet/common/draggable-default-cs2" %}

### Drag area

A drag area is a specific area within a user interface that is designated for drag and drop operations. When an object or element is dragged within a drag area, certain actions or events may be triggered. The user can specify the drag area by using the [dragArea](#) property. Refer to the below sample.

### APP.VUE

```

<template>
  <div id='container'>
    <div id='droppable'>
      <p class='drop'><span>Drag Area </span></p>
    </div>
    <div id='element1'>
      <p class='drag-text'>Drag </p>
    </div>
  </div>
</template>
<script>
  import { Draggable, Droppable } from '@syncfusion/ej2-base';
  import Vue from "vue";
  export default {
    mounted: function () {
      var draggable = new
      Draggable(document.getElementById('element1'), {
        clone: false, dragArea: "#droppable"
      });
    }
  }
</script>
<style>
  #element1 {
    height: 100px;
    width: 150px;
    border: 1px solid #cecece;
    cursor: move;
    background: #cdf3e3;
    user-select: none;
    touch-action: none;
  }
  #element1 p {
    padding-top: 23px;
    text-align: center;
  }
  .drop {
    padding-top: 23px;
    text-align: center;
  }
</style>

```

```
#draggable {
  margin: 5px;
  line-height: 170px;
  font-size: 14px;
  width: 250px;
  border: 1px solid #cecece;
  background: #f6f6f6;
  touch-action: none;
}
</style>
```

```
{% previewsample "page.domainurl/code-snippet/common/drag-drop-action-cs1" %}
```

### Droppable

Droppable component refers to an area of a user interface that can receive a draggable component that is being moved by a user. Syncfusion's [Droppable](#) library converts any DOM element into a droppable zone, which accepts draggable elements.

When a draggable component is moved over a droppable component, the [drop](#) event can be triggered. The user can get details about the dropped element through the event argument. Based on the event argument, the user can append the dragged element to the target element.

Refer to the following code snippet to enable droppable zones.

### APP.VUE

```
<template>
  <div id='container'>
    <div id='draggable'>
      <p class='drop'><span>Drop Target </span></p>
    </div>
    <div id='element1'>
      <p class='drag-text'>Drag </p>
    </div>
  </div>
</template>
<script>
  import { Draggable, Droppable } from '@syncfusion/ej2-base';
  import Vue from "vue";
  export default {
    mounted: function () {
      var draggable = new
Draggable(document.getElementById('element1'), {
        clone: false
      });
      var droppable = new
Droppable(document.getElementById('draggable'), {
        drop: (e) => {
          e.droppedElement.querySelector('.drag-text').textContent
= 'Dropped';
        }
      });
    }
  }
</script>
<style>
```

```

#element1 {
  height: 100px;
  width: 150px;
  border: 1px solid #cecece;
  cursor: move;
  background: #cdffe3;
  user-select: none;
  touch-action: none;
}
#element1 p {
  padding-top: 23px;
  text-align: center;
}
.drop {
  padding-top: 23px;
  text-align: center;
}
#droppable {
  margin: 5px;
  line-height: 170px;
  font-size: 14px;
  width: 250px;
  border: 1px solid #cecece;
  background: #f6f6f6;
  touch-action: none;
}
</style>

```

{% previewsample "page.domainurl/code-snippet/common/drag-drop-action-cs2" %}

See also

[Define handle element for draggable](#)<br/>

[Restricting draggable within container](#)<br>

[Visual feedback of draggable element](#)<br>

[Accepting specific drag element in droppable](#)

## Deployment

### CDN

The CDN links are provided individually for all the scripts and style sheets of Syncfusion Vue UI components.

The CDN link is provided to following files in the package

1. UMD Files
2. CSS Files

The un-versioned CDN links, which always maintain the latest version scripts are deprecated from 2022 Vol1 - 20.1 version. These links are available with **19.4** version scripts to avoid breaking in sites and apps that use un-versioned links.

*The latest minified versions of all UMD, Global and CSS files are available on CDN:*

- <https://cdn.syncfusion.com/ej2/PACKAGENAME/dist/PACKAGENAME.umd.min.js>
- <https://cdn.syncfusion.com/ej2/PACKAGENAME/styles/THEMENAME.css>

For example

- <https://cdn.syncfusion.com/ej2/ej2-vue-inputs/dist/ej2-vue-inputs.umd.min.js>
- <https://cdn.syncfusion.com/ej2/ej2-vue-inputs/styles/material.css>

Versioned files are also available on CDN.

- <https://cdn.syncfusion.com/ej2/VERSION/PACKAGENAME/dist/PACKAGENAME.umd.min.js>
- <https://cdn.syncfusion.com/ej2/VERSION/PACKAGENAME/styles/THEMENAME.css>

For example

- <https://cdn.syncfusion.com/ej2/1.0.18/ej2-vue-inputs/dist/ej2-vue-inputs.umd.min.js>
- <https://cdn.syncfusion.com/ej2/1.0.18/ej2-vue-inputs/styles/material.css>

## Packages

Syncfusion Vue packages is published and available in public [npm](#) registry.

### Anatomy of NPM packages

Syncfusion Vue UI components are shipped as npm packages. Following table explains the purpose of each file available in the package.

Files	Purpose
----- -----	
-----	
<b>dist/es6</b>	This folder contains the ES6 formatted JS file of the package.
<b>dist/&lt;packagename&gt;.umd.min.js</b> <b>dist/&lt;packagename&gt;.umd.js</b>	For applications using AMD or Common JS based module loader can be utilize these files.
<b>src/</b>	This folder contains the script files in AMD format. You can connect these AMD files as a package in System JS or Require JS.
<b>styles/&lt;theme name&gt;.css</b> <b>styles/&lt;theme name&gt;.scss</b>	This folder contains the CSS and SCSS files of the package.

## How To

### Updating Syncfusion npm packages

Keeping Syncfusion npm packages up to date is important to ensure that you have access to the latest features and bug fixes. The [npm-check-updates](#) package is a helpful tool that can be used to update your Syncfusion packages to their latest versions.

#### Updating All Syncfusion Packages

To update all Syncfusion packages, you can install the `npm-check-updates` package globally by running the following command,

```
`bash
npm install -g npm-check-updates
`
```

Next, use the `ncu` command to update the `package.json` file to the latest version for all `@syncfusion` packages,

```
`bash
ncu -u -f /^@syncfusion/
`
```

Finally, run the following commands to update the packages in `node_modules` and remove any duplicate packages that have been installed,

```
`bash
npm update
npm dedupe
`
```

### Updating a specific npm package

You can also update a specific npm package by running the following commands from the command prompt in the root of your application,

```
`bash
npm update @syncfusion/ej2-grids
npm update @syncfusion/ej2-vue-grids
npm dedupe
`
```

This will update the specific package you have provided and run `npm dedupe` command which will remove any duplicate package.

### How to resolve Content Security Policy (CSP) errors

Enabling the strict Content Security Policy (CSP) may cause the following issue with the Syncfusion Vue components in your application.



### Image loading

Syncfusion license banner utilize the image from **base64**, which is not allowed on strict CSP-enabled sites. To overcome this restriction, it is necessary to add the [img-src data:](#) directive in the meta tag or consider [registering the license key](#).

## Troubleshoot

### Content Security Policy

Content Security Policy (CSP) is a security feature implemented by web browsers that helps to protect against attacks such as cross-site scripting (XSS) and data injection. It limits the sources from which content can be loaded on a web page.

To enable strict [Content Security Policy \(CSP\)](#), certain browser features are disabled by default. In order to use Syncfusion Vue components with strict CSP mode, it is essential to include following directives in the CSP meta tag.

- Syncfusion components are rendered with calculated **inline styles** and **base64** font icons, which are blocked on a strict CSP-enabled site. To allow them, add the [style-src 'self' 'unsafe-inline'](#); and [font-src 'self' data:](#) directives in the meta tag as follows.

### HTML

```
<meta http-equiv="Content-Security-Policy" content="default-src 'self';
style-src 'self' 'unsafe-inline';
font-src 'self' data:;" />
```

- Syncfusion **material** and **tailwind** built-in themes contain a reference to the [Roboto's external font](#), which is also blocked. To allow them, add the [external font](#) reference to the [style-src](#) and [font-src](#) directives in the above meta tag.

The resultant meta tag is included within the `<head>` tag and resolves the CSP violation on the application's side when utilizing Syncfusion Vue components with material and tailwind themes.

### HTML

```
<head>
...
<meta http-equiv="Content-Security-Policy" content="default-src 'self';
style-src 'self' https://fonts.googleapis.com/ 'unsafe-inline';
font-src 'self' https://fonts.googleapis.com/ https://fonts.gstatic.com/
data:;" />
</head>
```

**Note:** From the 2023 Vol2 - 22.1 release onwards, the Content Security Policy for Syncfusion Vue components has been enhanced. The usage of the `unsafe-eval` directive has been eliminated from the CSP meta tag.

[View the Vue sample enabled with strict CSP in Github](#)

*See also*

- [How to resolve the Content Security Policy \(CSP\) errors](#)

## Visual studio code integration

### Visual Studio Code Integration

#### Overview

The Syncfusion Essential Studio Web extension for Visual Studio Code allows you to use the Syncfusion JavaScript components(React, Angular, and Vue) easily by configuring the Syncfusion NPM packages and themes.

The Syncfusion Web Extension provides the following support in Visual Studio Code:

[Project-Template](#): Creates Syncfusion Web applications by adding the required Syncfusion JavaScript components.

### Visual Studio Code Extensions

#### Download and Installation

Syncfusion publishes the Visual Studio Code extension in [Visual Studio Code marketplace](#). You can either install it from Visual Studio Code or download and install it from the Visual Studio Code marketplace.

#### Prerequisites

The following prerequisites software needs to be installed for the Syncfusion Web extension installation and for creating the Syncfusion Web applications along with any one of the Framework(React, Angular, and Vue).

- [Visual Studio Code](#)

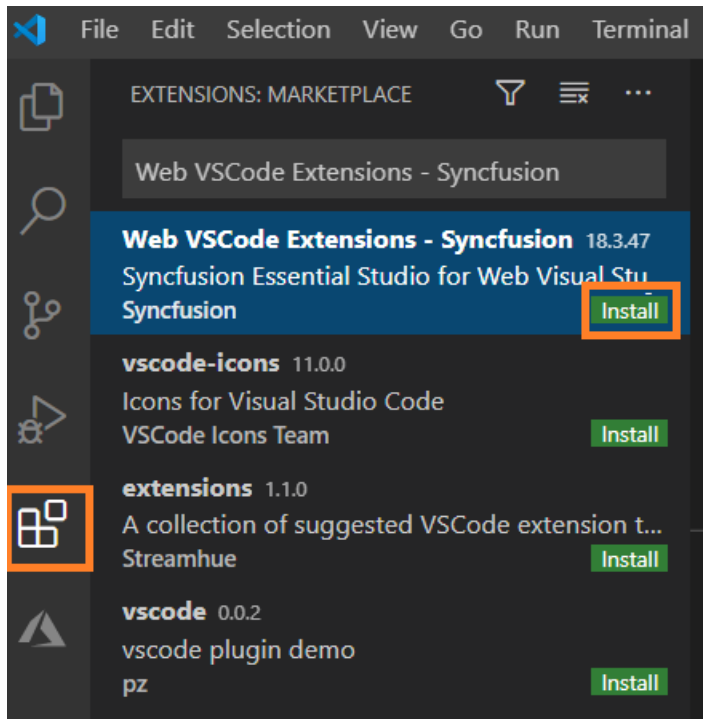
The minimum version of the Visual Studio Code is 1.38.0 to use the Syncfusion Web Extension.

- [C# Extension](#)
- [Node.js](#)

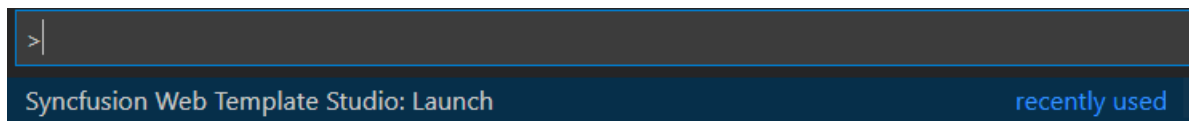
#### Install through the Visual Studio Code extensions

The following steps explain how to install the Syncfusion Web extensions from Visual Studio Code Extensions.

1. Open Visual Studio Code.
2. Go to **View > Extensions**, and open Manage Extensions.
3. Type **"Syncfusion Web"** in the search box.



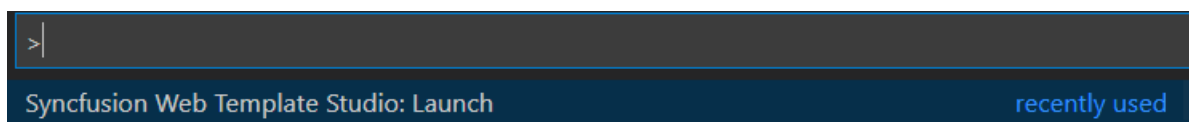
4. Click the Install button on the “Web VSCode Extensions - Syncfusion” extension.
5. After the installation, reload the Visual Studio Code using the **Reload Required** in Visual Studio Code palette.
6. Now, use the Syncfusion Web extensions from the Visual Studio Code palette.



#### *Install from the Visual Studio Code Marketplace*

The following steps explain how to download Syncfusion Web applications from the Visual Studio Code Marketplace and install them.

1. Open the [Syncfusion Web Extension](#)
2. Click Install from Visual Studio Code Marketplace. The browser opens the popup with the information like “Open Visual Studio Code”. Click Open Visual Studio Code, then [Syncfusion Web Extension](#) will open in Visual Studio Code.
3. Click the Install button in the “Web VSCode Extensions - Syncfusion” extension.
4. After the installation, reload the Visual Studio Code using the Reload Required in Visual Studio Code palette.
5. Now, use the Syncfusion Web extensions from the Visual Studio Code palette.



## Visual Studio Code Extensions

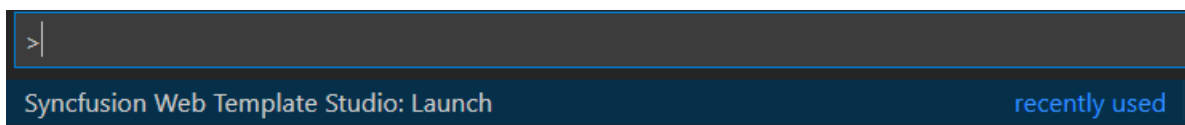
### Create project

Syncfusion provides **project templates** for **Visual Studio Code** to create Syncfusion Web applications. The Syncfusion Web Project template creates applications with the selected Framework (React, Angular, and Vue), required Syncfusion NPM packages, component render code for the Grid, Chart, and Scheduler components, and a style to make development with Syncfusion components easier.

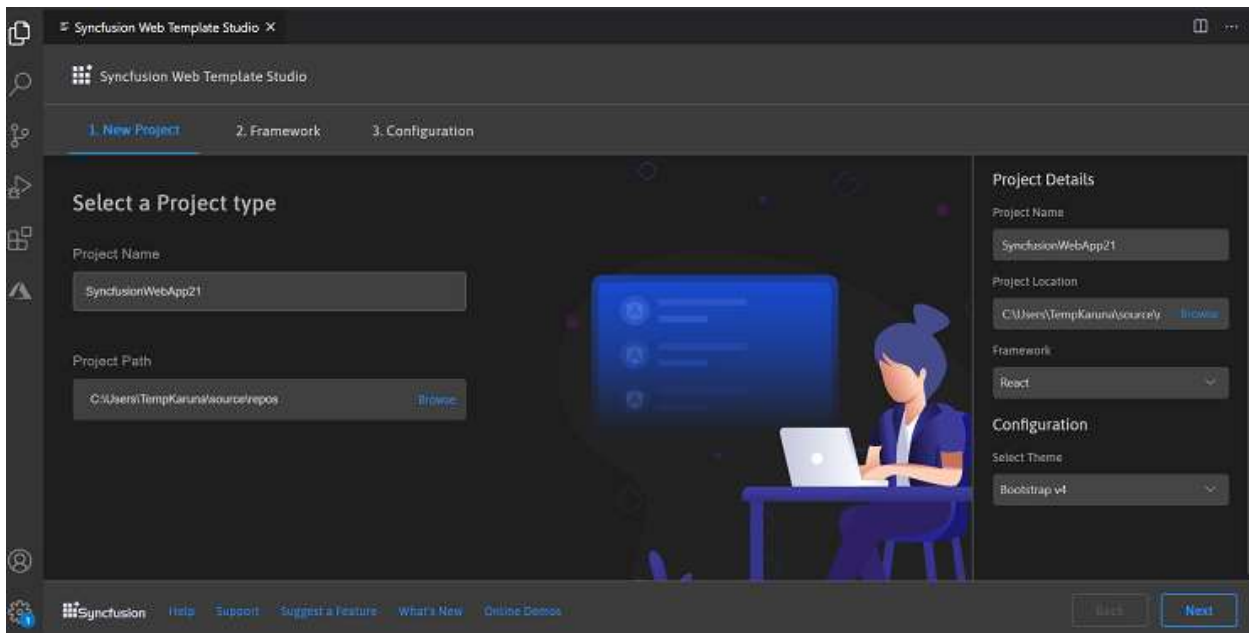
The Syncfusion Visual Studio Code project template provides support for Web project templates from v18.3.0.47.

The steps below help you to create **Syncfusion Web Applications** through the **Visual Studio Code**:

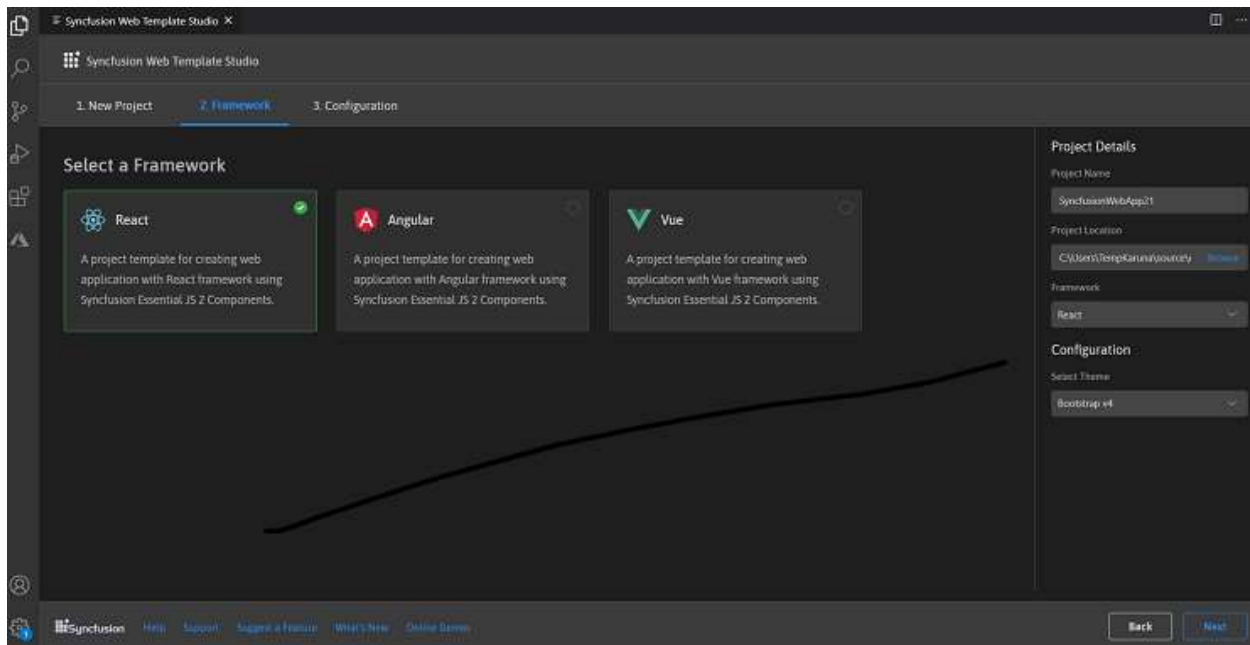
1. In Visual Studio Code, open the command palette by pressing Ctrl+Shift+P. The Visual Studio Code palette opens, search the word Syncfusion, so you can get the templates provided.



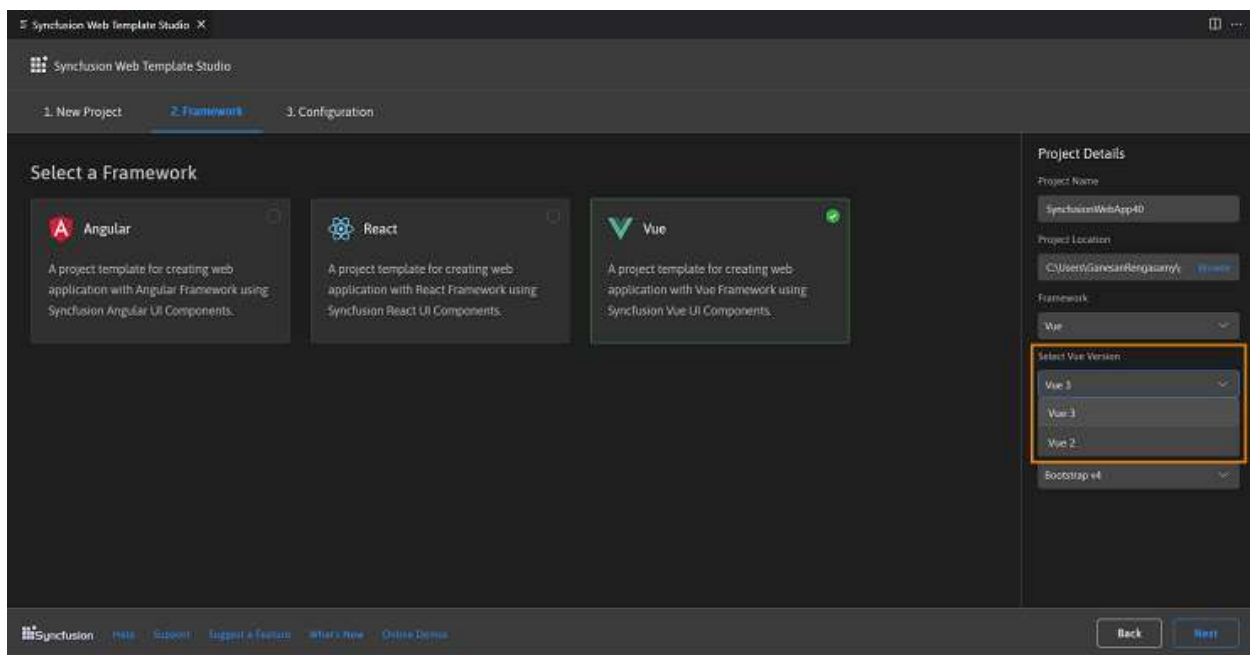
2. Select **Syncfusion Web Template Studio: Launch** and then press enter, Template Studio wizard for configuring the Syncfusion Web app will appear. Provide the require Project Name and Path to create the new Syncfusion Web application along with any one of the Framework (React, Angular, and Vue).



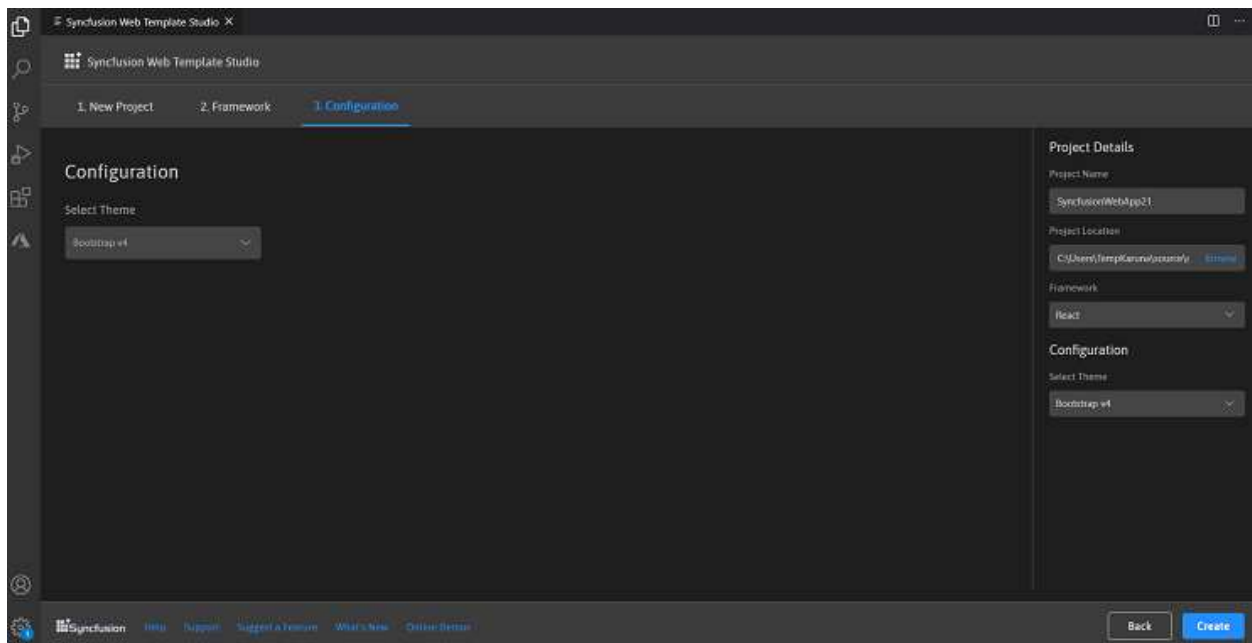
3. Click either **Next** or **Framework** tab, and the Framework types will be appears. Choose any one of the Framework:
  - React
  - Angular
  - Vue



If you choose the Vue framework, the **Select Vue Version** option will appear in the **Project Details** section. You can create the Vue application using either the Vue 3 or Vue 2 versions.



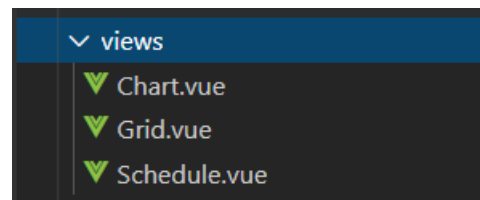
4. Click either **Next** or the **Configuration** tab, and the Configuration section will be loaded. Choose the preferred theme and then click **Create**. The project will be created.



5. The created Syncfusion Web App is configured with the Syncfusion NPM packages, styles, and the component render code for the Syncfusion component added.

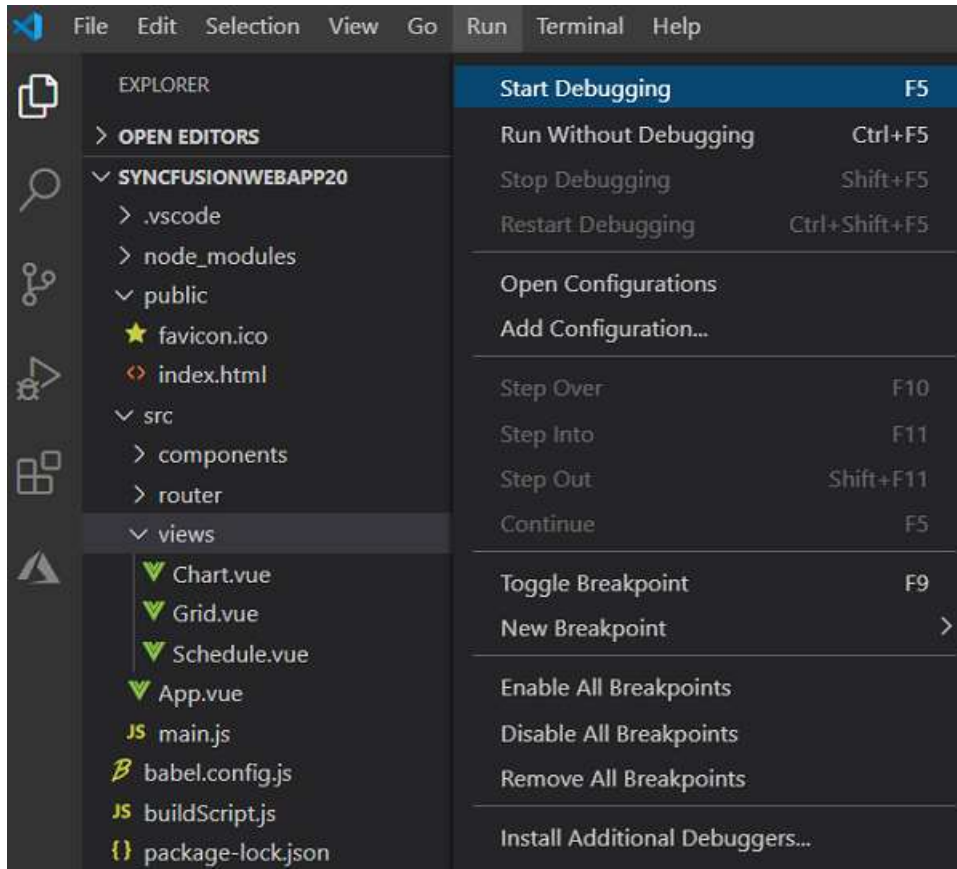
```
"name": "syncfusionwebproject",
"version": "0.1.0",
"private": true,
"dependencies": {
  "cookie-parser": "1.4.5",
  "debug": "4.1.1",
  "express": "4.17.1",
  "http-errors": "1.8.0",
  "morgan": "1.10.0",
  "bootstrap": "4.5.0",
  "bootstrap-vue": "2.15.0",
  "core-js": "3.6.5",
  "vue": "2.6.11",
  "vue-router": "3.3.4",
  "fs-extra": "9.0.1",
  "@syncfusion/ej2-vue-charts": "^18.3.35",
  "@syncfusion/ej2-vue-grids": "^18.3.35",
  "@syncfusion/ej2-vue-schedule": "^18.3.35"
},
```

```
25
26 <style>
27   @import '../..node_modules/@syncfusion/ej2-base/styles/fabric.css';
28   @import '../..node_modules/@syncfusion/ej2-buttons/styles/fabric.css';
29   @import '../..node_modules/@syncfusion/ej2-calendars/styles/fabric.css';
30   @import '../..node_modules/@syncfusion/ej2-dropdowns/styles/fabric.css';
31   @import '../..node_modules/@syncfusion/ej2-inputs/styles/fabric.css';
32   @import '../..node_modules/@syncfusion/ej2-navigations/styles/fabric.css';
33   @import '../..node_modules/@syncfusion/ej2-popups/styles/fabric.css';
34   @import '../..node_modules/@syncfusion/ej2-vue-schedule/styles/fabric.css';
35
36   #app{
37     height: 350px;
38   }
39 </style>
```



*Run the application*

1. Click on **F5** or navigate to **Run>Start debugging**



2. After compilation process completed, open the local host link in browser to see the output.

SyncfusionWebApp20 Grid Chart Schedule

### Syncfusion Vue Grid

Order ID	Customer ID	Freight
10248	VINET	\$32.38
10249	TOMSP	\$11.61
10250	HANAR	\$65.83
10251	VICTE	\$41.34
10252	SUPRD	\$51.30

1 of 2 pages (9 items)

**SyncfusionWebApp20**

Syncfusion Vue is a modern UI Components library that has been built from the ground up to be lightweight, responsive, modular and touch friendly.

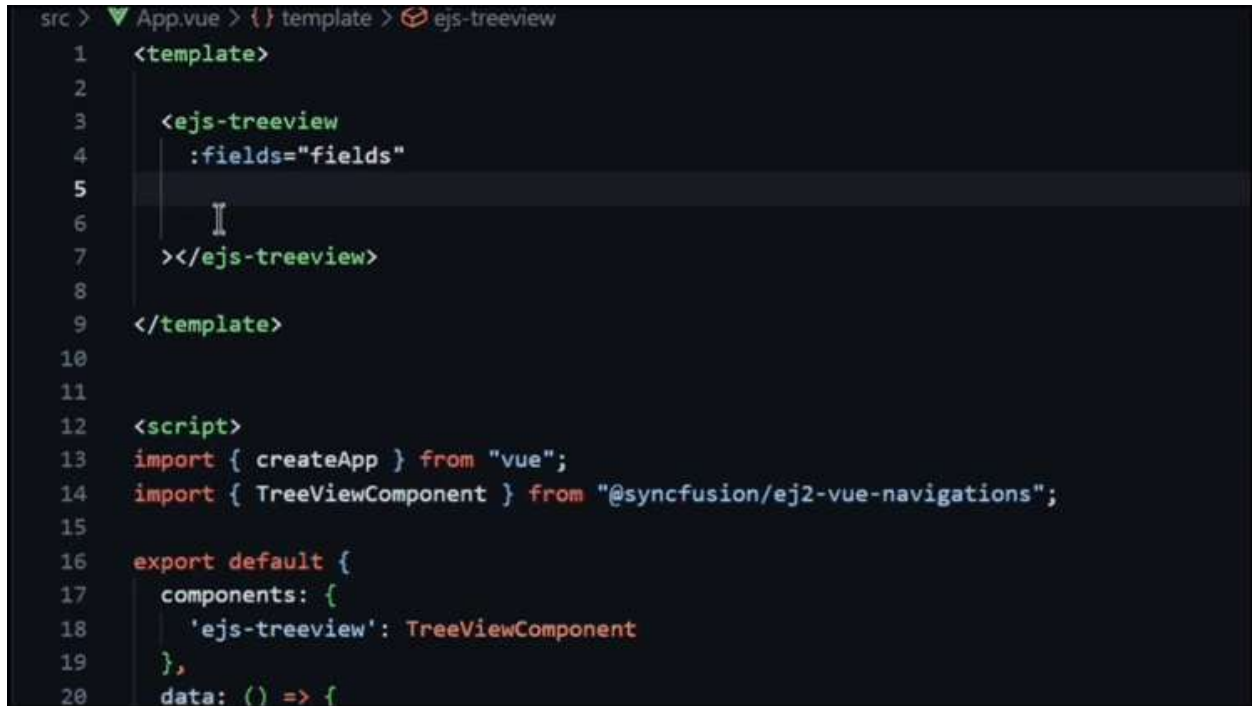
Help  
Online Samples  
Support

## Intellisense of Syncfusion Vue Components

Intellisense of the Vue component provides code suggestions, error checking, property hints, and other useful information while typing. It will save a lot of time and help you develop more precise and efficient code.



Install the [Volar](#) extension for Visual Studio Code to use Intellisense in the Syncfusion Vue component. After installing the addon, typing in your Vue files will generate Intellisense suggestions.



```

src > App.vue > {} template > ejs-treeview
1  <template>
2
3  <ejs-treeview
4    :fields="fields"
5
6    </ejs-treeview>
7
8  </template>
9
10
11
12  <script>
13  import { createApp } from "vue";
14  import { TreeViewComponent } from "@syncfusion/ej2-vue-navigations";
15
16  export default {
17    components: {
18      'ejs-treeview': TreeViewComponent
19    },
20    data: () => {

```

It's significant to note that the Vue component properties and events are listed as **kebab-casing** rather than **camel-casing**. Vue versions 2.7 and above will support Intellisense.

## Accordion

### Getting Started with the Vue Accordion Component in Vue 2

This article provides a step-by-step guide for setting up a Vue 2 project using [Vue-CLI](#) and integrating the Syncfusion Vue Accordion component using the [Composition API](#) / [Options API](#).

#### Prerequisites

[System requirements for Syncfusion Vue UI components](#)

#### Dependencies

The following list of dependencies are required to use the Accordion component in your application.

`javascript

```

|-- @syncfusion/ej2-vue-navigations
|-- @syncfusion/ej2-base
|-- @syncfusion/ej2-vue-base
|-- @syncfusion/ej2-navigations
|-- @syncfusion/ej2-inputs
|-- @syncfusion/ej2-buttons
`

```

### Setting up the Vue 2 project

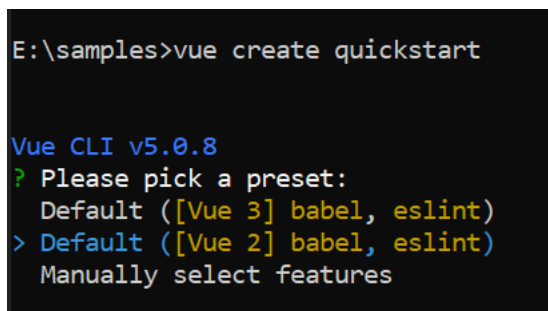
To generate a Vue 2 project using Vue-CLI, use the [vue create](#) command. Follow these steps to install Vue CLI and create a new project:

```
`bash
npm install -g @vue/cli
vue create quickstart
cd quickstart
npm run serve
`
```

or

```
`bash
yarn global add @vue/cli
vue create quickstart
cd quickstart
yarn run serve
`
```

When creating a new project, choose the option **Default ([Vue 2] babel, eslint)** from the menu.



```
E:\samples>vue create quickstart

Vue CLI v5.0.8
? Please pick a preset:
  Default ([Vue 3] babel, eslint)
> Default ([Vue 2] babel, eslint)
  Manually select features
```

Once the **quickstart** project is set up with default settings, proceed to add Syncfusion components to the project.

### Adding syncfusion packages

All the available Essential JS 2 packages are published in [npmjs.com](https://www.npmjs.com) registry. You can choose the component that you want to install. For this application, we are going to use Accordion component.

To install Accordion component, use the following command

```
`bash
npm install @syncfusion/ej2-vue-navigations --save
`

or

`bash
```

```
yarn add @syncfusion/ej2-vue-navigations
```

Import Syncfusion CSS styles

Add Accordion component's styles as given below in `<style>` section of the `App.vue` file.

In this article, the `Material` theme is applied using CSS styles, which are available in installed packages. The necessary Material CSS styles for the Accordion component and its dependents were imported into the `<style>` section of `src/App.vue` file.

#### ~/SRC/APP.VUE

```
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-
navigations/styles/material.css";
</style>
```

#### Add Syncfusion Vue component

Follow the below steps to add the Vue Accordion component using `Composition API` or `Options API`:

1\ First, import and register the Accordion component in the `script` section of the `src/App.vue` file. If you are using the `Composition API`, you should add the `setup` attribute to the `script` tag to indicate that Vue will be using the `Composition API`.

#### COMPOSITION API (~/SRC/APP.VUE)

```
<script setup>
import {
  AccordionComponent as EjsAccordion, AccordionItemsDirective as
  EAccordionitems, AccordionItemDirective as EAccordionitem
} from "@syncfusion/ej2-vue-navigations";
</script>
```

#### COMPOSITION API (~/SRC/APP.VUE)

```
<script>
import { AccordionComponent, AccordionItemDirective, AccordionItemsDirective
} from '@syncfusion/ej2-vue-navigations';
export default {
  name: 'app',
  components: {
    'ejs-accordion': AccordionComponent
  },
}
</script>
```

2\ Add the EJ2 Vue Accordion using to the `App.vue` file in `src` directory.

#### (~/SRC/APP.VUE)

```
<template>
<div id="app">
<ejs-accordion >
```

```

<e-accordionitems>
  <e-accordionitem expanded='true' header='ASP.NET' content='Microsoft ASP.NET
  is a set of technologies in the Microsoft .NET Framework for building Web
  applications and XML Web services.'></e-accordionitem>
  <e-accordionitem header='ASP.NET MVC' content='The Model-View-Controller
  (MVC) architectural pattern separates an application into three main
  components: the model, the view, and the controller.'></e-accordionitem>
  <e-accordionitem header='JavaScript' content='JavaScript (JS) is an
  interpreted computer programming language. It was originally implemented as
  part of web browsers so that client-side scripts could interact with the
  user, control the browser, communicate asynchronously, and alter the
  document content that was displayed.'></e-accordionitem>
</e-accordionitems>
</ejs-accordion>
</div>
</template>

```

### Running the Application

Now run the `npm run dev` command in the console, it will build your application and open in the browser.

### COMPOSITION API (~SRC/APP.VUE)

```

<template>
  <ejs-accordion>
    <e-accordionitems>
      <e-accordionitem expanded="true" header="ASP.NET"
        content="Microsoft ASP.NET is a set of technologies in the Microsoft
        .NET Framework for building Web applications and XML Web services."></e-
        accordionitem>
      <e-accordionitem header="ASP.NET MVC"
        content="The Model-View-Controller (MVC) architectural pattern
        separates an application into three main components: the model, the view,
        and the controller."></e-accordionitem>
      <e-accordionitem header="JavaScript"
        content="JavaScript (JS) is an interpreted computer programming
        language. It was originally implemented as part of web browsers so that
        client-side scripts could interact with the user, control the browser,
        communicate asynchronously, and alter the document content that was
        displayed."></e-accordionitem>
    </e-accordionitems>
  </ejs-accordion>
</template>
<script setup>
import {
  AccordionComponent as EjsAccordion, AccordionItemsDirective as
  EAccordionitems, AccordionItemDirective as EAccordionitem
} from "@syncfusion/ej2-vue-navigations";
</script>
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-
navigations/styles/material.css";
</style>

```

### OPTIONS API (~ /SRC/APP.VUE)

```
<template>
  <div id="app">
    <ejs-accordion >
      <e-accordionitems>
        <e-accordionitem expanded='true' header='ASP.NET' content='Microsoft
ASP.NET is a set of technologies in the Microsoft .NET Framework for
building Web applications and XML Web services.'></e-accordionitem>
        <e-accordionitem header='ASP.NET MVC' content='The Model-View-
Controller (MVC) architectural pattern separates an application into three
main components: the model, the view, and the controller.'></e-
accordionitem>
        <e-accordionitem header='JavaScript' content='JavaScript (JS) is an
interpreted computer programming language.It was originally implemented as
part of web browsers so that client-side scripts could interact with the
user, control the browser, communicate asynchronously, and alter the
document content that was displayed.'></e-accordionitem>
      </e-accordionitems>
    </ejs-accordion>
  </div>
</template>
<script>
import { AccordionComponent, AccordionItemDirective, AccordionItemsDirective
} from '@syncfusion/ej2-vue-navigations';
export default {
  name: 'app',
  components: {
    'ejs-accordion': AccordionComponent,
    'e-accordionitem': AccordionItemDirective,
    'e-accordionitems': AccordionItemsDirective
  },
}
</script>
<style>
  @import "../node_modules/@syncfusion/ej2-base/styles/material.css";
  @import "../node_modules/@syncfusion/ej2-vue-
navigations/styles/material.css";
</style>
```

{% previewsample "page.domainurl/code-snippet/accordion/getting-started-cs1" %}

#### Initialize the Accordion using HTML elements

The Accordion component can be rendered based on the given HTML element using `<ejs-accordion>`. You need to follow the below structure of HTML elements to render the Accordion inside the `<ejs-accordion>` tag.

,

`<ejs-accordion>` --> Root Accordion Element

`<div>` --> Accordion Item Container

`<div>` --> Accordion Header Container

`<div> </div>` --> Accordion Header

```

</div>
<div> --> Accordion Panel Container
<div> </div> --> Accordion Content
</div>
</div>
</ejs-accordion>
,

```

### COMPOITION API (~SRC/APP.VUE)

```

<template>
  <div id="app">
    <ejs-accordion>
      <div>
        <div>
          <div> ASP.NET </div>
        </div>
        <div>
          <div> Microsoft ASP.NET is a set of technologies in the
Microsoft .NET Framework for building Web applications
and XML Web services </div>
        </div>
      </div>
      <div>
        <div>
          <div> ASP.NET MVC </div>
        </div>
        <div>
          <div> The Model-View-Controller (MVC) architectural
pattern separates an application into three main components:
the model, the view, and the controller </div>
        </div>
      </div>
      <div>
        <div>
          <div> JavaScript </div>
        </div>
        <div>
          <div> JavaScript (JS) is an interpreted computer
programming language.It was originally implemented as part
of web browsers so that client-side scripts could
interact with the user, control the browser </div>
        </div>
      </div>
    </ejs-accordion>
  </div>
</template>
<script setup>
import { AccordionComponent as EjsAccordion } from '@syncfusion/ej2-vue-
navigations';
</script>
<style>
  @import "../node_modules/@syncfusion/ej2-base/styles/material.css";

```

```
@import "../node_modules/@syncfusion/ej2-vue-
navigations/styles/material.css";
</style>
```

### OPTIONS API (~SRC/APP.VUE)

```
<template>
  <div id="app">
    <ejs-accordion>
      <div>
        <div>
          <div> ASP.NET </div>
        </div>
        <div>
          <div> Microsoft ASP.NET is a set of technologies in the
Microsoft .NET Framework for building Web applications
and XML Web services </div>
        </div>
      </div>
    </div>
    <div>
      <div>
        <div> ASP.NET MVC </div>
      </div>
      <div>
        <div> The Model-View-Controller (MVC) architectural
pattern separates an application into three main components:
the model, the view, and the controller </div>
      </div>
    </div>
    <div>
      <div>
        <div> JavaScript </div>
      </div>
      <div>
        <div> JavaScript (JS) is an interpreted computer
programming language.It was originally implemented as part
of web browsers so that client-side scripts could
interact with the user, control the browser </div>
      </div>
    </div>
  </ejs-accordion>
</div>
</template>
<script>
import { AccordionComponent } from '@syncfusion/ej2-vue-navigations';
export default {
  name: 'app',
  components: {
    'ejs-accordion': AccordionComponent
  },
}
</script>
<style>
  @import "../node_modules/@syncfusion/ej2-base/styles/material.css";
  @import "../node_modules/@syncfusion/ej2-vue-
navigations/styles/material.css";
```

```
</style>
```

```
{% previewsample "page.domainurl/code-snippet/accordion/accordion-container-cs1" %}
```

**Note:** You can refer to our [Vue Accordion](#) feature tour page for its groundbreaking feature representations. You can also explore our [Vue Accordion Component example](#) that shows how to render the Accordion in Vue.

## Getting Started with Vue Accordion Component in Vue 3

This article provides a step-by-step guide for setting up a [Vite](#) project with a JavaScript environment and integrating the Syncfusion Vue Accordion component using the [Composition API](#) / [Options API](#).

The **Composition API** is a new feature introduced in Vue.js 3 that provides an alternative way to organize and reuse component logic. It allows developers to write components as functions that use smaller, reusable functions called composition functions to manage their properties and behavior.

The **Options API** is the traditional way of writing Vue.js components, where the component logic is organized into a series of options that define the component's properties and behavior. These options include data, methods, computed properties, watchers, lifecycle hooks, and more.

### Prerequisites

#### [System requirements for Syncfusion Vue UI components](#)

#### Setup the Vite project

A recommended approach for beginning with Vue is to scaffold a project using [Vite](#). To create a new Vite project, use one of the commands that are specific to either NPM or Yarn.

```
`bash
```

```
npm create vite@latest
```

```
,
```

```
or
```

```
`bash
```

```
yarn create vite
```

```
,
```

Using one of the above commands will lead you to set up additional configurations for the project as below:

1. Define the project name: We can specify the name of the project directly. Let's specify the name of the project as **my-project** for this article.

```
`bash
```

```
? Project name: » my-project
```

```
,
```

2. Select **Vue** as the framework. It will create a Vue 3 project.

```
`bash
```

```
? Select a framework: » - Use arrow-keys. Return to submit.
```



Vanilla

Vue

React

Preact

Lit

Svelte

Others

,

3. Choose **JavaScript** as framework variant to build this Vite project using JavaScript and Vue.

```
`bash
```

? Select a variant: » - Use arrow-keys. Return to submit.

JavaScript

TypeScript

Customize with create-vue ↗

Nuxt ↗

,

4. Upon completing the aforementioned steps to create the **my-project**, run the following command to install its dependencies:

```
`bash
```

```
cd my-project
```

```
npm install
```

,

or

```
`bash
```

```
cd my-project
```

```
yarn install
```

,

Now that **my-project** is ready to run with default settings, let's add Syncfusion Vue Accordion component to the project.

#### Adding Syncfusion Vue packages

Syncfusion Vue component packages are available at [npmjs.com](https://www.npmjs.com). To use Syncfusion Vue components in the project, install the corresponding npm package.

This article uses the [Vue Accordion component](#) as an example. To use the **Vue Accordion** component in the project, the **@syncfusion/ej2-vue-navigations** package needs to be installed using the following command

```
`bash
npm install @syncfusion/ej2-vue-navigations --save
```

```
,
```

or

```
`bash
yarn add @syncfusion/ej2-vue-navigations
```

```
,
```

### Import Syncfusion CSS styles

You can import themes for the Syncfusion Vue component in various ways, such as using CSS or SASS styles from npm packages, CDN, [CRG](#) and [Theme Studio](#). Refer to [themes topic](#) to know more about built-in themes and different ways to refer to themes in a Vue project.

In this article, **Material** theme is applied using CSS styles, which are available in installed packages. The necessary **Material** CSS styles for the Accordion component and its dependents were imported into the `<style>` section of the **src/App.vue** file.

```
`html
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-navigations/styles/material.css";
</style>
```

### Adding Syncfusion Vue component

Follow the below steps to add the Vue Accordion component using **Composition API** or **Options API**:

1.First, import and register the Accordion component and its child directives in the **script** section of the **src/App.vue** file. If you are using the **Composition API**, you should add the **setup** attribute to the **script** tag to indicate that Vue will be using the **Composition API**.

#### **COMPOSITION API (~SRC/APP.VUE)**

```
<script setup>
import {
  AccordionComponent as EjsAccordion, AccordionItemsDirective as
  EAccordionitems, AccordionItemDirective as EAccordionitem
} from "@syncfusion/ej2-vue-navigations";
</script>
```

#### **OPTIONS API (~SRC/APP.VUE)**

```
<script>
import {
  AccordionComponent,
  AccordionItemsDirective,
  AccordionItemDirective
} from "@syncfusion/ej2-vue-navigations";
```

```
export default {  
  name: "App",  
  components: {  
    "ejs-accordion": AccordionComponent,  
    "e-accordionitems": AccordionItemsDirective,  
    "e-accordionitem": AccordionItemDirective  
  }  
}  
</script>
```

2.Add the component definition in template section.

```
`html  
<template>  
  <ejs-accordion>  
    <e-accordionitems>  
      <e-accordionitem expanded="true" header="ASP.NET"  
        content="Microsoft ASP.NET is a set of technologies in the Microsoft .NET Framework for building Web  
        applications and XML Web services."></e-accordionitem>  
      <e-accordionitem header="ASP.NET MVC"  
        content="The Model-View-Controller (MVC) architectural pattern separates an application into three  
        main components: the model, the view, and the controller."></e-accordionitem>  
      <e-accordionitem header="JavaScript"  
        content="JavaScript (JS) is an interpreted computer programming language.It was originally  
        implemented as part of web browsers so that client-side scripts could interact with the user, control the  
        browser, communicate asynchronously, and alter the document content that was displayed."></e-  
        accordionitem>  
    </e-accordionitems>  
  </ejs-accordion>  
</template>  
`
```

Here is the summarized code for the above steps in the **src/App.vue** file:

#### **COMPOSITION API (~SRC/APP.VUE)**

```
<template>  
<ejs-accordion>  
<e-accordionitems>  
<e-accordionitem expanded="true" header="ASP.NET"  
  content="Microsoft ASP.NET is a set of technologies in the Microsoft .NET  
  Framework for building Web applications and XML Web services."></e-  
  accordionitem>  
<e-accordionitem header="ASP.NET MVC"  
  content="The Model-View-Controller (MVC) architectural pattern separates an  
  application into three main components: the model, the view, and the  
  controller."></e-accordionitem>
```

```
<e-accordionitem header="JavaScript"
content="JavaScript (JS) is an interpreted computer programming language.It
was originally implemented as part of web browsers so that client-side
scripts could interact with the user, control the browser, communicate
asynchronously, and alter the document content that was displayed."></e-
accordionitem>
</e-accordionitems>
</ejs-accordion>
</template>
<script setup>
import {
AccordionComponent as EjsAccordion, AccordionItemsDirective as
EAccordionitems, AccordionItemDirective as EAccordionitem
} from "@syncfusion/ej2-vue-navigations";
</script>
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-
navigations/styles/material.css";
</style>
```

### OPTIONS API (~SRC/APP.VUE)

```
<template>
<ejs-accordion>
<e-accordionitems>
<e-accordionitem expanded="true" header="ASP.NET"
content="Microsoft ASP.NET is a set of technologies in the Microsoft .NET
Framework for building Web applications and XML Web services."></e-
accordionitem>
<e-accordionitem header="ASP.NET MVC"
content="The Model-View-Controller (MVC) architectural pattern separates an
application into three main components: the model, the view, and the
controller."></e-accordionitem>
<e-accordionitem header="JavaScript"
content="JavaScript (JS) is an interpreted computer programming language.It
was originally implemented as part of web browsers so that client-side
scripts could interact with the user, control the browser, communicate
asynchronously, and alter the document content that was displayed."></e-
accordionitem>
</e-accordionitems>
</ejs-accordion>
</template>
<script>
import {
AccordionComponent,
AccordionItemsDirective,
AccordionItemDirective,
} from "@syncfusion/ej2-vue-navigations";
export default {
name: "App",
components: {
"ejs-accordion": AccordionComponent,
"e-accordionitems": AccordionItemsDirective,
"e-accordionitem": AccordionItemDirective,
}
}
```

```

}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-
navigations/styles/material.css";
</style>

```

### Run the project

To run the project, use the following command:

```
`bash
```

```
npm run dev
```

```
,
```

or

```
`bash
```

```
yarn run dev
```

```
,
```

The output will appear as follows:



Refer the sample [Vue 3 using Composition API Accordion getting started](#)

For migrating from Vue 2 to Vue 3, refer to the [migration](#) documentation.

See also

- [Getting Started with Vue UI Components using Composition API and TypeScript](#)
- [Getting Started with Vue UI Components using Options API and TypeScript](#)

### Expand mode in Vue Accordion component

The Accordion supports the two listed types of expand modes while expanding or collapsing the item.

- Single
- Multiple

#### Single

The property enables to expand only one Accordion item at a time. If you expand any new item, the previously expanded one is collapsed and

new item changed to expanded state.

#### **APP.VUE**

```
<template>
```

```

<div id="app">
  <ejs-accordion expandMode='Single'>
    <e-accordionitems>
      <e-accordionitem expanded='true' header='ASP.NET' content='Microsoft
ASP.NET is a set of technologies in the Microsoft .NET Framework for
building Web applications and XML Web services.'></e-accordionitem>
      <e-accordionitem header='ASP.NET MVC' content='The Model-View-
Controller (MVC) architectural pattern separates an application into three
main components: the model, the view, and the controller.'></e-
accordionitem>
      <e-accordionitem header='JavaScript' content='JavaScript (JS) is an
interpreted computer programming language.It was originally implemented as
part of web browsers so that client-side scripts could interact with the
user, control the browser, communicate asynchronously, and alter the
document content that was displayed.'></e-accordionitem>
    </e-accordionitems>
  </ejs-accordion>
</div>
</template>
<script>
import Vue from 'vue';
import { AccordionPlugin } from '@syncfusion/ej2-vue-navigations';
Vue.use(AccordionPlugin);
export default {
  name: 'app',
}
</script>
<style>
  @import "../node_modules/@syncfusion/ej2-base/styles/material.css";
  @import "../node_modules/@syncfusion/ej2-vue-
navigations/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/accordion/expandmode/single-cs1" %}

## Multiple

Default expand mode of the Accordion is **Multiple**. It enables you to expand more than one Accordion item at a time. Expand/collapse action

can also be toggled by clicking on it again. For example, expanded item is collapsed when you click on it again.

## APP.VUE

```

<template>
  <div id="app">
    <ejs-accordion expandMode='Multiple'>
      <e-accordionitems>
        <e-accordionitem expanded='true' header='ASP.NET' content='Microsoft
ASP.NET is a set of technologies in the Microsoft .NET Framework for
building Web applications and XML Web services.'></e-accordionitem>
        <e-accordionitem header='ASP.NET MVC' content='The Model-View-
Controller (MVC) architectural pattern separates an application into three
main components: the model, the view, and the controller.'></e-
accordionitem>
      </e-accordionitems>
    </ejs-accordion>
  </div>
</template>

```

```

        <e-accordionitem header='JavaScript' content='JavaScript (JS) is an
        interpreted computer programming language.It was originally implemented as
        part of web browsers so that client-side scripts could interact with the
        user, control the browser, communicate asynchronously, and alter the
        document content that was displayed.'></e-accordionitem>
    </e-accordionitems>
</ejs-accordion>
</div>
</template>
<script>
import Vue from 'vue';
import { AccordionPlugin } from '@syncfusion/ej2-vue-navigations';
Vue.use(AccordionPlugin);
export default {
    name: 'app',
}
</script>
<style>
    @import "../node_modules/@syncfusion/ej2-base/styles/material.css";
    @import "../node_modules/@syncfusion/ej2-vue-
    navigations/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/accordion/expandmode/multiple-cs1" %}

See Also

- [How to keep single pane open always](#)

## Accessibility in Vue Accordion component

The Accordion component has been designed keeping in mind the [WAI-ARIA](#) specifications, by applying the prompt WAI-ARIA roles, states, and properties along with the keyboard support. Thus, making it usable for people who use assistive WAI-ARIA Accessibility supports that is achieved through the attributes like `aria-labelledby`. It helps to provides information about the elements in a document for assistive technology. The component implements the keyboard navigation support by following the [WAI-ARIA practices](#) and tested in major screen readers.

The accessibility compliance for the Accordion component is outlined below.

| Accessibility Criteria | Compatibility |

| -- | -- |

| [WCAG 2.2](#) Support |  |

| [Section 508](#) Support |  |

| Screen Reader Support |  |

| Right-To-Left Support |  |

| Color Contrast |  |

| Mobile Device Support |  |

| Keyboard Navigation Support |  |

| [Accessibility Checker](#) Validation |  |

| [Axe-core](#) Accessibility Validation |  |

<style>

.post .post-content img {

display: inline-block;

margin: 0.5em 0;

}

</style>

<div> - All features of the component meet the requirement.</div>

<div> - Some features of the component do not meet the requirement.</div>

<div> - The component does not meet the requirement.</div>

## ARIA attributes

### | Roles and Attributes | Functionalities

|

| ----- | ----- |

| role | **Button:** Attribute is set to the Accordion header elements to indicate that the element can be used to toggle the visibility of the associated content section, describing the actual role of the element.<br> **Region:** Attribute is set to the Accordion panel elements to create a landmark region that contains the currently expanded accordion panel, describing the actual role of the element. <br>|

| aria-labelledby | Attribute is set to content (panel) and it points to the corresponding Accordion header. |

| aria-controls | Attribute is set to the header and it points to the corresponding Accordion content. |

| aria-expanded | Attribute is set to the Accordion header elements to indicates the expand state of the Accordion Item. Default value of this attribute is `false`. If an item is expanded, the attribute value changes to 'true'. |

| aria-selected | Attribute set to the Tab items to indicates the selection state for Tab items. Active Tab is set to true for this attribute. |



| **aria-hidden** | Attribute is set to the Accordion panel elements to indicates the content visible state of the Accordion Item. Default value of this attribute is **true**. If an item content is visible, the attribute value changes to **false**. |

| **aria-disabled** | It indicates the disabled state of the Accordion and its items. |

### Keyboard interaction

Keyboard navigation is enabled by default. Possible keys are:

Key	Description
-----	-----
Space or Enter	When focus is on the Accordion header, click on the focused element makes the element to expand and collapse.
Down Arrow	Focus the next Accordion header.
Up Arrow	Focus the previous Accordion header.
Home	Focus the first Accordion header.
End	Focus the last Accordion header.
Tab	To Move focus through the interactive elements.
Shift + Tab	To Move focus through the interactive elements.

### Ensuring accessibility

The Accordion component's accessibility levels are ensured through an [accessibility-checker](#) and [axe-core](#) software tools during automated testing.

The accessibility compliance of the Accordion component is shown in the following sample. Open the [sample](#) in a new window to evaluate the accessibility of the Accordion component with accessibility tools.

```
{% previewsample "https://ej2.syncfusion.com/accessibility/accordion.html" %}
```

See also

- [Accessibility in Syncfusion Vue components](#)

### Style in Vue Accordion component

The following content provides the exact CSS structure that can be used to modify the control's appearance based on user preference.

### Customizing Accordion

Use the following CSS to customize the Accordion.

```
`css
.e-accordion {
border: 5px solid rgb(173, 255, 47);
}
`
```

### Customizing the list items

Use the following CSS to customize the items of Accordion.

```
`css
.e-accordion .e-acrdn-item {
text-align: center;
color: pink;
background-color: #2fa1ff;
}
`
```

### Customizing Accordion's header

Use the following CSS to customize the header of Accordion control.

```
`css
.e-accordion .e-acrdn-item.e-select > .e-acrdn-header {
background: #2fa1ff !important;
justify-content: center;
}
`
```

### Customizing Accordion's expand and collapse icons

Use the following CSS to customize the expand and collapse icons of Accordion control.

```
`css
.e-accordion .e-acrdn-item .e-acrdn-header .e-toggle-icon .e-icons {
color: pink;
}
`
```

### Customizing the hover state of Accordion control

Use the following CSS to customize the accordion item when hovering.

```
`css
.e-accordion .e-acrdn-item .e-acrdn-header:hover {
border: 2px solid gray;
}
`
```

### Customizing selected item of Accordion control

Use the following CSS to customize the selected accordion item.

```
`css
```

```
.e-accordion .e-acrdn-item.e-select.e-active>.e-acrdn-header,
.e-accordion .e-acrdn-item.e-select.e-item-focus>.e-acrdn-header {
background-color: rgb(0, 15, 100) !important;
}
`
```

Use the following CSS to customize the selected accordion item text.

```
`css
.e-accordion .e-acrdn-item.e-select.e-active>.e-acrdn-header .e-acrdn-header-content,
.e-accordion .e-acrdn-item.e-select.e-item-focus>.e-acrdn-header .e-acrdn-header-content {
color: #2fa1ff !important;
}
`
```

## How To

Set the nested accordion in [Vue Accordion component](#)

Accordion supports to render **nested** level of Accordion by using content property. You can give nested Accordion content inside the parent Accordion content property by using **id** of nested element. The nested Accordion can be rendered with the use of provided events, such as **clicked** and **expanding**.

### APP.VUE

```
<template>
  <div id="app">
    <ejs-accordion ref="Accordion_Nested" :expanding="expanding">
      <e-accordionitems>
        <e-accordionitem expanded='true' header='Video' content='<div
id="nested_video"></div>'>
        </e-accordionitem>
        <e-accordionitem header='Music' content='<div
id="nested_music"></div>' ></e-accordionitem>
        <e-accordionitem header='Images' content='<div
id="nested_images"></div>'></e-accordionitem>
      </e-accordionitems>
    </ejs-accordion>
  </div>
</template>
<script>
import Vue from 'vue';
import { AccordionPlugin } from '@syncfusion/ej2-vue-navigations';
import { ExpandEventArgs, Accordion, AccordionClickArgs } from
 '@syncfusion/ej2-navigations';
Vue.use(AccordionPlugin);
export default {
  name: 'app',
  data () {
  },
  methods: {
    clicked: function (e) {
```

```

var ele = e.originalEvent.target;
if (ele.querySelectorAll('.e-accordion').length > 0) {
    return;
}
var nestAcrdn_musNew = new Accordion({
    items: [{
        header: 'New Track1'
    },
    {
        header: 'New Track2'
    }
    ],
    '#nested_musicNew');
}

expanding: function (e) {
var obj = this.$refs.Accordion_Nested.ej2Instances
if (e.isExpanded && [].indexOf.call(obj.items, e.item) === 0) {
    if (e.element.querySelectorAll('.e-accordion').length > 0) {
        return;
    }
    var nestAcrdn_vid = new Accordion({
        items: [{
            header: 'Video Track1'
        },
        {
            header: 'Video Track2'
        }
        ],
        '#nested_video');
    }
if (e.isExpanded && [].indexOf.call(obj.items, e.item) === 1) {
    if (e.element.querySelectorAll('.e-accordion').length > 0) {
        return;
    }
    var nestAcrdn_mus = new Accordion({
        clicked: this.clicked,
        items: [{
            header: 'Music Track1'
        },
        {
            header: 'Music Track2'
        },
        {
            header: 'Music New',
            content: '<div id="nested_musicNew"></div>'
        }
        ],
        '#nested_music');
    }
if (e.isExpanded && [].indexOf.call(obj.items, e.item) === 2) {
    if (e.element.querySelectorAll('.e-accordion').length > 0) {
        return;
    }
    var nestAcrdn_img = new Accordion({
        items: [{
            header: 'Track1'
        },

```

```

        {
            header: 'Track2'
        },
    ],
    }, '#nested_images');
}
}
}
</script>
<style>
    @import "../node_modules/@syncfusion/ej2-base/styles/material.css";
    @import "../node_modules/@syncfusion/ej2-vue-
    navigations/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/accordion/how-to/nestedaccordion-cs1" %}

### Load content through post in Vue Accordion component

Accordion supports to load external contents through **AJAX** library. Refer the below steps.

- Import the **Ajax** module from **ej2-base** and initialize with URL path.
- Get data from the Ajax Success event to initialize Accordion with retrieved external path data.

### APP.VUE

```

<template>
    <div id="app">
        <div id="acrdnContnet1" style="display:none">
            <ul style="margin : 0px;padding:0px 16px; list-style-type: none">
                <li>Testing</li>
                <li>Development</li>
            </ul>
        </div>
        <div id="acrdnContnet2" style="display:none">
            <ul style="margin : 0px;padding:0px 16px; list-style-type: none">
                <li>Mobile</li>
                <li>Web</li>
            </ul>
        </div>
        <ejs-accordion ref="acrdnInstance">
            <e-accordionitems>
                <e-accordionitem header='Department' content = '#acrdnContnet1'></e-
                accordionitem>
                <e-accordionitem header='Platform' content = '#acrdnContnet2'></e-
                accordionitem>
                <e-accordionitem header='Employee Details'></e-accordionitem>
            </e-accordionitems>
        </ejs-accordion>
    </div>
</template>
<script>
    import Vue from 'vue';
    import { AccordionPlugin } from '@syncfusion/ej2-vue-navigations';
    import { ExpandEventArgs, Accordion, AccordionClickArgs } from
    '@syncfusion/ej2-navigations';

```

```
import { Ajax } from '@syncfusion/ej2-base';
Vue.use(AccordionPlugin);
export default {
  name: 'app',
  data () {
    let ajax: Ajax = new Ajax('./Ajax.html', 'GET', true);
    ajax.send().then();
    ajax.onSuccess = (data: string): void => {
      var obj = this.$refs.acrdnInstance.ej2Instances
      obj.items[2].content = data;
      obj.refresh();
    }
  }
}
</script>
<style>
  @import "../node_modules/@syncfusion/ej2-base/styles/material.css";
  @import "../node_modules/@syncfusion/ej2-vue-
  navigations/styles/material.css";
  div#headername {
    margin-left: 100px;
  }
</style>
```

{% previewsample "page.domainurl/code-snippet/accordion/how-to/accordion-ajax-cs1" %}

### Set custom animation in Vue Accordion component

Accordion supports custom animations for both expand and collapse actions from the provided animation option of **Animation** library.

Default animation is given as **SlideDown** for expanding the panel and **SlideUp** for collapsing the panel. You can also disable the animation by setting animation effect as **none**.

The sample demonstrates some types of animation that suits for Accordion. You can check all the animation effects [here](#).

### APP.VUE

```
<template>
  <div id="app">
    <div style='padding-top: 25px'>
      <div class='row'>
        <div class="col-xs-6 col-sm-6 col-lg-6 col-md-6">
          <label> Expand Animation </label>
        </div>
        <div class="col-xs-6 col-sm-6 col-lg-6 col-md-6">
          <div class='custom_drop'><ejs-dropdownlist ref="expandInstance"
            :change='expandAnimationChange' index='0'
            :dataSource='expandAni' placeholder='Expand Animation'></ejs-
            dropdownlist></div>
        </div>
      </div>
      <div class='row'>
        <div class="col-xs-6 col-sm-6 col-lg-6 col-md-6">
          <label> Collapse Animation </label>
        </div>
      </div>
    </div>
  </div>
```

```

        <div class="col-xs-6 col-sm-6 col-lg-6 col-md-6">
            <div class='custom_drop'><ejs-dropdownlist ref="collapseInstance"
:change='collapseAnimationChange' index='1'
            :dataSource='expandAni' placeholder='Collapse
Animation'></ejs-dropdownlist></div>
        </div>
    </div>
    <div style='padding-top: 25px'>
        <ejs-accordion ref="acrdnInstance">
            <e-accordionitems>
                <e-accordionitem expanded='true' header='ASP.NET'
                    content='Microsoft ASP.NET is a set of technologies in the
Microsoft .NET Framework for building Web applications and XML Web
services'></e-accordionitem>
                <e-accordionitem header='ASP.NET MVC'
                    content='The Model-View-Controller (MVC) architectural pattern
separates an application into three main components: the model, the view,
and the controller.'></e-accordionitem>
                <e-accordionitem header='JavaScript'
                    content='JavaScript (JS) is an interpreted computer
programming language.It was originally implemented as part of web browsers
so that client-side scripts could interact with the user, control the
browser, communicate asynchronously, and alter the document content that was
displayed.'></e-accordionitem>
            </e-accordionitems>
        </ejs-accordion>
    </div>
</div>
</template>
<script>
import Vue from 'vue';
import { AccordionPlugin } from '@syncfusion/ej2-vue-navigations';
import { Accordion, AccordionEffect } from '@syncfusion/ej2-navigations';
import { DropDownList } from '@syncfusion/ej2-dropdowns';
import { DropDownListPlugin } from "@syncfusion/ej2-vue-dropdowns";
Vue.use(AccordionPlugin);
Vue.use(DropDownListPlugin);
export default {
    name: 'app',
    data: function () {
        return {
            expandAni: ['SlideDown', 'SlideUp', 'FadeIn', 'FadeOut', 'FadeZoomIn',
'FadeZoomOut', 'ZoomIn', 'ZoomOut', 'None']
        };
    },
    methods: {
        expandAnimationChange: function (e) {
            var obj = this.$refs.acrdnInstance.ej2Instances;
            var ddlobj = this.$refs.expandInstance.ej2Instances;
            obj.animation.expand.effect = ddlobj.value;
        },
        collapseAnimationChange: function (e) {
            var obj = this.$refs.acrdnInstance.ej2Instances;
            var ddlobj = this.$refs.collapseInstance.ej2Instances;
            obj.animation.collapse.effect = ddlobj.value;
        }
    }
}

```

```

    }
  }
</script>
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-
navigations/styles/material.css";
</style>

```

```
{% previewsample "page.domainurl/code-snippet/accordion/how-to/accordion-custom-animation-cs1"
%}
```

To keep single pane open always in Vue Accordion component

By default, all Accordion panels are collapsible. You can customize the Accordion to keep one panel as expanded state always. This is applicable for **Single** expand mode.

### APP.VUE

```

<template>
  <div id="app">
    <ejs-accordion ref="element" expandMode='Single' :expanding="beforeExpand"
    :clicked="clicked">
      <e-accordionitems>
        <e-accordionitem expanded='true' header='ASP.NET' content='Microsoft
ASP.NET is a set of technologies in the Microsoft .NET Framework for
building Web applications and XML Web services'></e-accordionitem>
        <e-accordionitem header='ASP.NET MVC' content='The Model-View-
Controller (MVC) architectural pattern separates an application into three
main components: the model, the view, and the controller.'></e-
accordionitem>
        <e-accordionitem header='JavaScript' content='JavaScript (JS) is an
interpreted computer programming language.It was originally implemented as
part of web browsers so that client-side scripts could interact with the
user, control the browser, communicate asynchronously, and alter the
document content that was displayed.'></e-accordionitem>
      </e-accordionitems>
    </ejs-accordion>
  </div>
</template>
<script>
import Vue from 'vue';
import { AccordionPlugin } from '@syncfusion/ej2-vue-navigations';
import { Accordion, ExpandEventArgs, AccordionClickArgs } from
 '@syncfusion/ej2-navigations';
Vue.use(AccordionPlugin);
var clickEle;
export default {
  name: 'app',
  data() {
  },
  methods:{
    clicked: function(e) {
      clickEle = e.originalEvent.target.closest('.e-acrdn-header');
    },
    beforeExpand: function(e) {
      var obj = this.$refs.element.ej2Instances;

```



```

var expandCount = obj.element.querySelectorAll('.e-selected').length;
var ele = obj.element.querySelectorAll('.e-selected')[0];
if (ele) {
    ele = ele.firstChild
}
if (expandCount === 1 && ele === clickEle) {
    e.cancel = true;
}
},
}
}
</script>
<style>
    @import "../node_modules/@syncfusion/ej2-base/styles/material.css";
    @import "../node_modules/@syncfusion/ej2-vue-
navigations/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/accordion/how-to/open-always-cs1" %}

Create wizard using accordion in Vue Accordion component

Accordion items can be disabled dynamically by passing the index and boolean value with the [enableItem](#) method.

The below Wizard sample is designed for Online Shopping model. In this, each Accordion item is integrated with required components to fill

the details and designed for getting user details and making payment at the end. Each field is provided with validation for all mandatory

option to enable/disable to next Accordion.

### APP.VUE

```

<template>
  <div id="app">
    <div id="Sign_In_Form" style="display: none; padding: 3px 0">
      <form id="formId">
        <div class="form-group">
          <div class="e-float-input">
            <input type="text" id="email" name="Email" required="" />
            <span class="e-float-line"></span>
            <label class="e-float-text" for="email">Email</label>
          </div>
          <div class="e-float-input">
            <input id="password" type="password" name="Password" required=""
/>
            <span class="e-float-line"></span>
            <label class="e-float-text" for="password">Password</label>
          </div>
        </div>
      </form>
      <div style="text-align: center">
        <button class="e-btn" id="Continue_Btn">Continue</button>
        <div id="err1">* Please fill all fields</div>
      </div>
    </div>
  </div>

```

```

<div id="Address_Fill" style="display: none; padding: 3px 0">
  <form id="formId_Address">
    <div class="form-group">
      <div class="e-float-input">
        <input type="text" id="name" name="Name" required="" />
        <span class="e-float-line"></span>
        <label class="e-float-text" for="name">Name</label>
      </div>
    </div>
    <div class="form-group">
      <div class="e-float-input">
        <input type="text" id="address" name="Address" required="" />
        <span class="e-float-line"></span>
        <label class="e-float-text" for="address">Address</label>
      </div>
    </div>
    <div class="form-group">
      <ejs-numerictextbox
        id="mobile"
        format="0"
        placeholder="Mobile"
        floatLabelType="Auto"
        showSpinButton="false"
      >
    </ejs-numerictextbox>
    </div>
  </form>
  <div style="text-align: center">
    <button class="e-btn" id="Continue_BtnAdr">Continue</button>
    <div id="err2">* Please fill all fields</div>
  </div>
</div>
<div id="Card_Fill" style="display: none; padding: 3px 0">
  <div class="col-xs-6 col-sm-6 col-lg-6 col-md-6">
    <div class="form-group">
      <ejs-numerictextbox
        id="cardNo"
        format="0"
        placeholder="Card No"
        floatLabelType="Auto"
        showSpinButton="false"
      >
    </ejs-numerictextbox>
    </div>
  </div>
  <div class="col-xs-6 col-sm-6 col-lg-6 col-md-6">
    <div class="form-group">
      <div class="e-float-input">
        <input type="text" id="cardHolder" name="cardHolder" required=""
        />
        <span class="e-float-line"></span>
        <label class="e-float-text" for="cardHolder">CardHolder
Name</label>
      </div>
    </div>
  </div>
  <div class="col-xs-6 col-sm-6 col-lg-6 col-md-6">

```

```

        <ejs-datepicker
            id="expiry"
            width="100%"
            format="MM/yyyy"
            placeholder="Expiry Date"
            floatLabelType="Auto"
        >
    </ejs-datepicker>
</div>
<div class="col-xs-6 col-sm-6 col-lg-6 col-md-6">
    <div class="form-group">
        <ejs-numerictextbox
            id="CVV"
            format="0"
            placeholder="CVV"
            floatLabelType="Auto"
            showSpinButton="false"
        >
    </ejs-numerictextbox>
    </div>
</div>
<div style="text-align: center">
    <button class="e-btn" id="Back_Btn">Back</button>
    <button class="e-btn" id="Save_Btn">Save</button>
    <div id="err3">* Please fill all fields</div>
</div>
</div>
<ejs-dialog
    ref="alertDlg"
    header="Alert"
    width="200"
    isModal="true"
    content=""
    visible="false"
    :target="dlgTarget"
    :buttons="dlgButtons"
    :created="dlgCreated"
></ejs-dialog>
<ejs-accordion ref="accordionInc" :expanding="expanding">
    <e-accordionitems>
        <e-accordionitem
            expanded="true"
            header="Sign In"
            content="#Sign_In_Form"
        ></e-accordionitem>
        <e-accordionitem
            header="Delivery Address"
            content="#Address_Fill"
        ></e-accordionitem>
        <e-accordionitem
            header="Card Details"
            content="#Card_Fill"
        ></e-accordionitem>
    </e-accordionitems>
</ejs-accordion>
</div>
</template>

```

```
<script>
import Vue from "vue";
import { AccordionPlugin } from "@syncfusion/ej2-vue-navigations";
import { DialogPlugin } from "@syncfusion/ej2-vue-popups";
import { DatePickerPlugin } from "@syncfusion/ej2-vue-calendars";
import { NumericTextBoxPlugin } from "@syncfusion/ej2-vue-inputs";
Vue.use(AccordionPlugin);
Vue.use(DialogPlugin);
Vue.use(DatePickerPlugin);
Vue.use(NumericTextBoxPlugin);
export default {
  name: "app",
  data: function () {
    return {
      dlgButtons: [
        {
          buttonModel: { content: "Ok", isPrimary: true },
          click: () => {
            this.$refs.alertDlg.ej2Instances.hide();
            var obj1 = this.$refs.accordionInc.ej2Instances;
            if (
              obj1.expandedIndices[0] === 2 &&
              this.$refs.alertDlg.ej2Instances.content ===
                "Your payment successfully processed"
            ) {
              obj1.enableItem(0, true);
              obj1.enableItem(1, false);
              obj1.enableItem(2, false);
              obj1.expandItem(true, 0);
            }
          },
        },
      ],
      dlgTarget: document.body,
    };
  },
  methods: {
    checkMail: function (mail) {
      //eslint-disable-next-line
      if (/^\w+([\.-]?\w+)*@\w+([\.-]?\w+)*(\.\w{2,3})+$/i.test(mail)) {
        return true;
      } else {
        this.$refs.alertDlg.ej2Instances.content = "Enter valid email address";
        this.$refs.alertDlg.ej2Instances.show();
        return false;
      }
    },
    checkMobile: function (mobile) {
      if (mobile.match(/^\\d{10}$/)) {
        return true;
      } else {
        this.$refs.alertDlg.ej2Instances.content =
          "Mobile number length should be 10";
        this.$refs.alertDlg.ej2Instances.show();
        return false;
      }
    }
  }
}
```

```

    },
    checkCardNo: function (cardNo) {
        if (cardNo.match(/^\\d{16}$\\/)) {
            return true;
        } else {
            this.$refs.alertDlg.ej2Instances.content =
                "Card number length should be 16";
            this.$refs.alertDlg.ej2Instances.show();
            return false;
        }
    },
    checkCVV: function (cvv) {
        if (cvv.match(/^\\d{3}$\\/)) {
            return true;
        } else {
            this.$refs.alertDlg.ej2Instances.content =
                "CVV number length should be 3";
            this.$refs.alertDlg.ej2Instances.show();
            return false;
        }
    },
    dlgCreated: function () {
        this.$refs.alertDlg.ej2Instances.hide();
    },
    expanding: function (e) {
        if (
            e.name === "expanding" &&
            [].indexOf.call(this.$refs.accordionInc.ej2Instances.items, e.item)
            ===
            0
        ) {
            document.getElementById("Continue_Btn").onclick = () => {
                var email = document.getElementById("email");
                var password = document.getElementById("password");
                if (email.value !== "" && password.value !== "") {
                    if (this.checkMail(email.value)) {
                        email.value = password.value = "";
                        var acrdnObj = this.$refs.accordionInc.ej2Instances;
                        acrdnObj.enableItem(1, true);
                        acrdnObj.enableItem(0, false);
                        acrdnObj.expandItem(true, 1);
                    }
                    document.getElementById("err1").classList.remove("show");
                } else {
                    document.getElementById("err1").classList.add("show");
                }
            };
        } else if (
            e.name === "expanding" &&
            [].indexOf.call(this.$refs.accordionInc.ej2Instances.items, e.item)
            ===
            1
        ) {
            document.getElementById("Continue_BtnAdr").onclick = () => {
                var name = document.getElementById("name");
                var address = document.getElementById("address");
                var mobile = document.getElementById("mobile");
            };
        }
    }
}

```

```

        if (
            name.value !== "" &&
            address.value !== "" &&
            mobile.value !== ""
        ) {
            if (this.checkMobile(mobile.value)) {
                var acrdnObj = this.$refs.accordionInc.ej2Instances;
                acrdnObj.enableItem(2, true);
                acrdnObj.enableItem(1, false);
                acrdnObj.expandItem(true, 2);
            }
            document.getElementById("err2").classList.remove("show");
        } else {
            document.getElementById("err2").classList.add("show");
        }
    };
} else if (
    e.name === "expanding" &&
    [].indexOf.call(this.$refs.accordionInc.ej2Instances.items, e.item)
===
    2
) {
    document.getElementById("Back_Btn").onclick = () => {
        var acrdnObj = this.$refs.accordionInc.ej2Instances;
        acrdnObj.enableItem(1, true);
        acrdnObj.enableItem(2, false);
        acrdnObj.expandItem(true, 1);
    };
    document.getElementById("Save_Btn").onclick = () => {
        var cardHolder = document.getElementById("cardHolder");
        var expiry = document.getElementById("expiry");
        var cardNo = document.getElementById("cardNo");
        var cvv = document.getElementById("CVV");
        if (
            cardNo.value !== "" &&
            cardHolder.value !== "" &&
            expiry.value !== "" &&
            cvv.value !== ""
        ) {
            if (this.checkCardNo(cardNo.value)) {
                if (this.checkCVV(cvv.value)) {
                    this.$refs.alertDlg.ej2Instances.content =
                        "Your payment successfully processed";
                    this.$refs.alertDlg.ej2Instances.show();
                }
            }
            document.getElementById("err3").classList.remove("show");
        } else {
            document.getElementById("err3").classList.add("show");
        }
    };
}
},
},
};
</script>
<style>

```

```
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-
navigations/styles/material.css";
#err1, #err2, #err3 {
  display: none;
  color: red;
  margin-top: 10px;
  font-weight: 500;
}
#err1.show,
#err2.show,
#err3.show {
  display: block;
}
.e-dialog {
  max-height: 300px !important; /* csslint allow: important */
}
.template_title {
  text-align: center;
  padding: 10px 0;
  margin: 20px 0;
  text-overflow: ellipsis;
  font-weight: bold;
  font-size: 16px;
}
</style>
```

{% previewsample "page.domainurl/code-snippet/accordion/how-to/accordion-wizard-cs1" %}

Load accordion with data source in Vue Accordion component

You can bind any data object to Accordion items, by mapping it to [header](#) and [content](#) property.

In the below demo, Data is fetched from an OData service using **DataManager**. The result data is formatted as a JSON object with **header** and **content** fields, which is set to items property of Accordion.

### APP.VUE

```
<template>
  <div id="app">
    <ejs-accordion ref="accordionInc">
    </ejs-accordion>
  </div>
</template>
<script>
import Vue from 'vue';
import { AccordionPlugin } from '@syncfusion/ej2-vue-navigations';
import { DataManager, Query, ODataV4Adaptor, ReturnOption } from
 '@syncfusion/ej2-data';
Vue.use(AccordionPlugin);
export default {
  name: 'app',
  data() {
  }, mounted() {
    new DataManager({ url:
'https://services.odata.org/V4/Northwind/Northwind.svc/Employees', adaptor:
new ODataV4Adaptor })
```

```

        .executeQuery(new Query().range(4, 7)).then((e) => {
            var result = e.result;
            var obj = this.$refs.accordionInc.ej2Instances
            var itemsData = [];
            var mapping = { header: 'FirstName', content: 'Notes' };
            for (var i = 0; i < result.length; i++) {
                itemsData.push({ header: result[i][mapping.header], content:
result[i][mapping.content] });
            }
            obj.items = itemsData;
            obj.refresh();
        });
    }
}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-
navigations/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/accordion/how-to/accordion-datasource-cs1" %}

### Customize expand collapse actions in Vue Accordion component

Accordion component supports customizing the expand or collapse animation action behavior. You can manually change the expand animation action performed after the collapse animation operation performed on already expand pane when the expand icons are clicked.

By default, the Accordion component pane is expanded or collapsed, when click the expand or collapse icon. It is not affected on already expand pane.

The following sample demonstrates, how to expand the collapsed Accordion item after collapse animation performed on the expanded Accordion item using [created](#), [expanding](#), and [expanded](#) event. In the Expanding event, get the previously expanded item index and prevent the expanding behavior using `args.cancel` option. Expand the Accordion item dynamically based on specifying the `index` value using the [expandItem](#) public method and [expanded](#) event.

### APP.VUE

```

<template>
    <div id="app">
        <ejs-accordion ref="acrdnInstance" expandMode='Single'
:expanding="expanding" :expanded="expanded" :created="created" >
            <e-accordionitems>
                <e-accordionitem header='ASP.NET' :content='content0'></e-
accordionitem>
                <e-accordionitem header='ASP.NET MVC' :content='content1'></e-
accordionitem>
                <e-accordionitem header='JavaScript' :content='content2'></e-
accordionitem>
            </e-accordionitems>
        </ejs-accordion>
    </div>
</template>
<script>
import Vue from 'vue';

```



```
import { AccordionPlugin } from '@syncfusion/ej2-vue-navigations';
import { ExpandEventArgs, Accordion } from '@syncfusion/ej2-navigations';
Vue.use(AccordionPlugin);
export default {
  name: 'app',
  data: function() {
    return {
      content0: 'ASP.NET is an open-source server-side web application
framework designed for web development to produce ' +
'dynamic web pages. It was developed by Microsoft to allow
programmers to build dynamic web sites, web applications ' +
'and web services. It was first released in January 2002 with
version 1.0 of the .NET Framework, and is the successor ' +
'to Microsoft\'\'s Active Server Pages (ASP) technology. ASP.NET is
built on the Common Language Runtime (CLR), allowing ' +
'programmers to write ASP.NET code using any supported .NET
language. The ASP.NET SOAP extension framework allows ' +
'ASP.NET components to process SOAP messages.',
      content1: 'The ASP.NET MVC is a web application framework developed
by Microsoft, which implements the ' +
'model-view-controller (MVC) pattern. It is open-source software,
apart from the ASP.NET Web Forms component which is ' +
'proprietary. In the later versions of ASP.NET, ASP.NET MVC, ASP.NET
Web API, and ASP.NET Web Pages (a platform using ' +
'only Razor pages) will merge into a unified MVC 6.The project is
called ASP.NET vNext.',
      content2: 'JavaScript (JS) is an interpreted computer programming
language. It was originally implemented as ' +
'part of web browsers so that client-side scripts could interact
with the user, control the browser, communicate ' +
'asynchronously, and alter the document content that was
displayed.[5] More recently, however, it has become common in ' +
'both game development and the creation of desktop applications.',
    };
  },
  mounted: function() {
    this.initialLoad = true;
    this.isCollapsed = false;
    this.expandIndex;
  },
  methods: {
    expanded: function (e) {
      var obj = this.$refs.acrdnInstance.ej2Instances;
      if (!e.isExpanded && !this.initialLoad && this.isCollapsed) {
        obj.expandItem(true, this.expandIndex);
        this.isCollapsed = false;
      }
    }
    expanding: function (e) {
      var obj = this.$refs.acrdnInstance.ej2Instances;
      if (e.isExpanded && !this.initialLoad && !this.isCollapsed) {
        e.cancel = true;
        this.expandIndex = obj.items.indexOf(e.item);
        this.isCollapsed = true;
      }
    }
  }
  created: function (e) {
```

```

        this.initialLoad = false;
    }
}
</script>
<style>
    @import "../node_modules/@syncfusion/ej2-base/styles/material.css";
    @import "../node_modules/@syncfusion/ej2-vue-
    navigations/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/accordion/accordion-actions-cs1" %}

Render other components in accordion using template in Vue Accordion component

You can render other components inside Accordion using Vue **template**. Through this, we can add content as other components directly with all functionalities to our Accordion. Follow the below guidelines for using the other components as template in Accordion.

- Declare a template in the **template** section of the “.vue” file. An empty object “data” needs to be initialized in the data option of the default export object in **script** section.
- The template function needs to be assigned to the content property of the EJ2 Vue Accordion Component.

## APP.VUE

```

<template>
    <div id="app">
        <ejs-accordion >
            <e-accordionitems>
                <e-accordionitem expanded='true' header='Calendar'
:content='Template1'></e-accordionitem>
                <e-accordionitem header='DatePicker' :content='Template2'></e-
accordionitem>
                <e-accordionitem header='Numeric Textbox' :content='Template3'></e-
accordionitem>
            </e-accordionitems>
        </ejs-accordion>
    </div>
</template>
<script>
import Vue from 'vue';
import { AccordionPlugin } from "@syncfusion/ej2-vue-navigations";
Vue.use(AccordionPlugin);
import { DatePickerComponent, CalendarComponent,
DatePickerPlugin, CalendarPlugin } from '@syncfusion/ej2-vue-calendars';
Vue.use(DatePickerPlugin);
Vue.use(CalendarPlugin);
import { NumericTextBoxComponent, NumericTextBoxPlugin } from
"@syncfusion/ej2-vue-inputs";
Vue.use(NumericTextBoxPlugin);
export default {
    name: 'app',
    data: function() {
        return {

```

```

    Template1: function () {
      return {
        template: Vue.component('CalendarComponent', {
          template: '<ejs-calendar></ejs-calendar>',
          data() { return { }; }
        })
      }
    },
    Template2: function () {
      return {
        template: Vue.component('DatePickerComponent', {
          template: ' <ejs-datepicker :min="minDate" :max="maxDate"
:value="dateVal" ></ejs-datepicker>',
          data() { return { minDate : new Date("05/09/2017"),
                           maxDate : new Date("05/15/2017"),
                           dateVal : new Date("05/11/2017") }; }
        })
      }
    },
    Template3: function () {
      return {
        template: Vue.component('NumericTextBoxComponent', {
          template: '<ejs-numerictextbox value="100"></ejs-
numerictextbox>',
          data() { return { }; }
        })
      }
    }
  }
}
</script>
<style>
  @import "../node_modules/@syncfusion/ej2-base/styles/material.css";
  @import "../node_modules/@syncfusion/ej2-vue-buttons/styles/material.css";
  @import "../node_modules/@syncfusion/ej2-vue-popups/styles/material.css";
  @import "../node_modules/@syncfusion/ej2-vue-
navigations/styles/material.css";
  @import "../node_modules/@syncfusion/ej2-vue-
calendars/styles/material.css";
  @import "../node_modules/@syncfusion/ej2-vue-inputs/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/accordion/how-to/direct-components-cs1" %}

## Accumulation Chart

### Getting Started with the Vue Accumulation chart Component in Vue 2

This article provides a step-by-step guide for setting up a Vue 2 project using [Vue-CLI](#) and integrating the Syncfusion Vue Accumulation chart component

#### Prerequisites

[System requirements for Syncfusion Vue UI components](#)

### Dependencies

The list of minimum dependencies required to use an accumulation chart are follows:

```
`javascript
|-- @syncfusion/ej2-vue-charts
|-- @syncfusion/ej2-base
|-- @syncfusion/ej2-data
|-- @syncfusion/ej2-pdf-export
|-- @syncfusion/ej2-file-utils
|-- @syncfusion/ej2-compression
|-- @syncfusion/ej2-charts
|-- @syncfusion/ej2-vue-base
|-- @syncfusion/ej2-svg-base
`,`
```

### Setting up the Vue 2 project

To generate a Vue 2 project using Vue-CLI, use the [vue create](#) command. Follow these steps to install Vue CLI and create a new project:

```
`bash
npm install -g @vue/cli
vue create quickstart
cd quickstart
npm run serve
`,`
```

or

```
`bash
yarn global add @vue/cli
vue create quickstart
cd quickstart
yarn run serve
`,`
```

When creating a new project, choose the option **Default ([Vue 2] babel, eslint)** from the menu.

```
E:\samples>vue create quickstart

Vue CLI v5.0.8
? Please pick a preset:
  Default ([Vue 3] babel, eslint)
> Default ([Vue 2] babel, eslint)
  Manually select features
```

Once the `quickstart` project is set up with default settings, proceed to add Syncfusion components to the project.

#### Add Syncfusion Vue packages

Syncfusion packages are available at [npmjs.com](https://www.npmjs.com). To use Vue components, install the required npm package.

This article uses the `Vue Accumulation chart component` as an example. Install the `@syncfusion/ej2-vue-charts` package by running the following command:

```
`bash
```

```
npm install @syncfusion/ej2-vue-charts --save
```

```
,
```

or

```
`bash
```

```
yarn add @syncfusion/ej2-vue-charts
```

```
,
```

The `--save` will instruct NPM to include the chart package inside of the `dependencies` section of the `package.json`.

#### Add Syncfusion Vue component

Follow the below steps to add the Vue Accumulation chart component using `Composition API` or `Options API`:

1\ First, import and register the Accumulation chart component in the `script` section of the `src/App.vue` file.

#### ~/SRC/APP.VUE

```
<script>
import { AccumulationChartComponent, AccumulationSeriesCollectionDirective,
AccumulationSeriesDirective, PieSeries } from "@syncfusion/ej2-vue-charts";
export default {
  components: {
    'ejs-accumulationchart': AccumulationChartComponent,
    'e-accumulation-series-collection': AccumulationSeriesCollectionDirective,
    'e-accumulation-series': AccumulationSeriesDirective
  }
}
</script>
```

2\ In the `template` section, define the Accumulation chart component with the `dataSource` property.

**~/SRC/APP.VUE**

```
<template>
<div id="app">
  <ejs-accumulationchart id="container">
    <e-accumulation-series-collection>
      <e-accumulation-series :dataSource='seriesData' xName='x' yName='y'> </e-accumulation-series>
    </e-accumulation-series-collection>
  </ejs-accumulationchart>
</div>
</template>
```

3\ Declare the value for the `dataSource` property in the `script` section.

**~/SRC/APP.VUE**

```
<script>
data() {
  return {
    seriesData: data
  };
}
</script>
```

### Adding Chart Component

- Add the Vue Chart by using selector in section of the `App.vue` file.

The below example shows a basic Charts,

- Pie Series

By default pie series will be rendered on assigning JSON data to the series by using `dataSource` property. Map the field names in the JSON data to the `xName` and `yName` properties of the series.

**~/SRC/APP.VUE**

```
<template>
  <div id="app">
    <ejs-accumulationchart id="container">
      <e-accumulation-series-collection>
        <e-accumulation-series :dataSource='seriesData' xName='x'
yName='y'> </e-accumulation-series>
      </e-accumulation-series-collection>
    </ejs-accumulationchart>
  </div>
</template>
<script>
import { AccumulationChartComponent, AccumulationSeriesCollectionDirective,
AccumulationSeriesDirective, PieSeries } from "@syncfusion/ej2-vue-charts";
```

```
import { data } from "data.ts";
export default {
  components: {
    'ejs-accumulationchart': AccumulationChartComponent,
    'e-accumulation-series-collection':
      AccumulationSeriesCollectionDirective,
    'e-accumulation-series': AccumulationSeriesDirective
  },
  data() {
    return {
      seriesData: data
    };
  },
  provide: {
    accumulationchart: [PieSeries]
  }
};
</script>
<style>
  #container {
    height: 350px;
  }
</style>
```

{% previewsample "page.domainurl/code-snippet/chart/series/pie-cs9" %}

### Run the project

To run the project, use the following command:

```
`bash
```

```
npm run serve
```

```
,
```

or

```
`bash
```

```
yarn run serve
```

```
,
```

### Getting Started with the Vue Accumulation Chart Component in Vue 3

This article provides a step-by-step guide for setting up a [Vite](#) project with a JavaScript environment and integrating the Syncfusion Vue Accumulation Chart component using the [Composition API](#) / [Options API](#).

The **Composition API** is a new feature introduced in Vue.js 3 that provides an alternative way to organize and reuse component logic. It allows developers to write components as functions that use smaller, reusable functions called composition functions to manage their properties and behavior.

The **Options API** is the traditional way of writing Vue.js components, where the component logic is organized into a series of options that define the component's properties and behavior. These options include data, methods, computed properties, watchers, lifecycle hooks, and more.

### Prerequisites

[System requirements for Syncfusion Vue UI components](#)

### Set up the Vite project

A recommended approach for beginning with Vue is to scaffold a project using [Vite](#). To create a new Vite project, use one of the commands that are specific to either NPM or Yarn.

```
`bash
npm create vite@latest
`
```

or

```
`bash
yarn create vite
`
```

Using one of the above commands will lead you to set up additional configurations for the project as below:

1. Define the project name: We can specify the name of the project directly. Let's specify the name of the project as `my-project` for this article.

```
`bash
? Project name: » my-project
`
```

2. Select `Vue` as the framework. It will create a Vue 3 project.

```
`bash
? Select a framework: » - Use arrow-keys. Return to submit.
```

Vanilla

Vue

React

Preact

Lit

Svelte

Others

`

3. Choose `JavaScript` as the framework variant to build this Vite project using JavaScript and Vue.

```
`bash
? Select a variant: » - Use arrow-keys. Return to submit.
```

JavaScript

TypeScript

Customize with create-vue ↗



Nuxt ↗

,

4. Upon completing the aforementioned steps to create the `my-project`, run the following command to install its dependencies:

```
`bash
```

```
cd my-project
```

```
npm install
```

,

or

```
`bash
```

```
cd my-project
```

```
yarn install
```

,

Now that `my-project` is ready to run with default settings, let's add Syncfusion components to the project.

#### Add Syncfusion Vue packages

Syncfusion Vue component packages are available at [npmjs.com](https://www.npmjs.com). To use Syncfusion Vue components in the project, install the corresponding npm package.

This article uses the [Vue Accumulation Chart component](#) as an example. To use the Vue Accumulation Chart component in the project, the `@syncfusion/ej2-vue-charts` package needs to be installed using the following command:

```
`bash
```

```
npm install @syncfusion/ej2-vue-charts --save
```

,

or

```
`bash
```

```
yarn add @syncfusion/ej2-vue-charts
```

,

#### Add Syncfusion Vue component

Follow the below steps to add the Vue Accumulation Chart component using `Composition API` or `Options API`:

1. First, import and register the Accumulation Chart component and its child directives in the `script` section of the `src/App.vue` file. If you are using the `Composition API`, you should add the `setup` attribute to the `script` tag to indicate that Vue will be using the `Composition API`.

#### COMPOSITION API (~SRC/APP.VUE)

```
<script setup>
```

```
import { AccumulationChartComponent as EjsAccumulationchart,
AccumulationSeriesCollectionDirective as EAccumulationSeriesCollection,
AccumulationSeriesDirective as EAccumulationSeries, AccumulationLegend,
PieSeries, AccumulationTooltip } from "@syncfusion/ej2-vue-charts";
</script>
```

### OPTIONS API (~SRC/APP.VUE)

```
<script>
import { AccumulationChartComponent, AccumulationSeriesCollectionDirective,
AccumulationSeriesDirective, AccumulationLegend,
PieSeries, AccumulationTooltip } from "@syncfusion/ej2-vue-charts";
//Component registration
export default {
name: "App",
components: {
"ejs-accumulationchart": AccumulationChartComponent,
"e-accumulation-series-collection": AccumulationSeriesCollectionDirective,
"e-accumulation-series": AccumulationSeriesDirective
}
}
</script>
```

2. In the **template** section, define the Accumulation Chart component with the [dataSource](#) property.

### ~SRC/APP.VUE

```
<template>
<ejs-accumulationchart id="container" :legendSettings="legendSettings"
:tooltip="tooltip">
<e-accumulation-series-collection>
<e-accumulation-series :dataSource='data' xName='x' yName='y'
innerRadius="20%"> </e-accumulation-series>
</e-accumulation-series-collection>
</ejs-accumulationchart>
</template>
```

3. Declare the values for the **dataSource** property in the **script** section.

### COMPOSITION API (~SRC/APP.VUE)

```
<script setup>
const data = [
{ x: 'Argentina', y: 505370 },
{ x: 'Belgium', y: 551500 },
{ x: 'Cuba', y: 312685 },
{ x: 'Dominican Republic', y: 350000 },
{ x: 'Egypt', y: 301000 },
{ x: 'Kazakhstan', y: 300000 },
{ x: 'Somalia', y: 357022 }
];
const legendSettings = { visible: true };
const tooltip = { enable: true };
</script>
```

**OPTIONS API (~SRC/APP.VUE)**

```
<script>
data() {
  return {
    data: [
      { x: 'Argentina', y: 505370 },
      { x: 'Belgium', y: 551500 },
      { x: 'Cuba', y: 312685 },
      { x: 'Dominican Republic', y: 350000 },
      { x: 'Egypt', y: 301000 },
      { x: 'Kazakhstan', y: 300000 },
      { x: 'Somalia', y: 357022 }
    ],
    legendSettings: { visible: true },
    tooltip: {
      enable: true
    }
  };
}
</script>
```

Here is the summarized code for the above steps in the **src/App.vue** file:

**COMPOSITION API (~SRC/APP.VUE)**

```
<template>
<ejs-accumulationchart id="container" :legendSettings="legendSettings"
:tooltip="tooltip">
<e-accumulation-series-collection>
<e-accumulation-series :dataSource='data' xName='x' yName='y'
innerRadius="20%"> </e-accumulation-series>
</e-accumulation-series-collection>
</ejs-accumulationchart>
</template>
<script setup>
import { provide } from 'vue';
import { AccumulationChartComponent as EjsAccumulationchart,
AccumulationSeriesCollectionDirective as EAccumulationSeriesCollection,
AccumulationSeriesDirective as EAccumulationSeries, AccumulationLegend,
PieSeries, AccumulationTooltip } from "@syncfusion/ej2-vue-charts";
const data = [
  { x: 'Argentina', y: 505370 },
  { x: 'Belgium', y: 551500 },
  { x: 'Cuba', y: 312685 },
  { x: 'Dominican Republic', y: 350000 },
  { x: 'Egypt', y: 301000 },
  { x: 'Kazakhstan', y: 300000 },
  { x: 'Somalia', y: 357022 }
];
const legendSettings = { visible: true };
const tooltip = { enable: true };
const accumulationchart = [PieSeries, AccumulationLegend,
AccumulationTooltip];
provide('accumulationchart', accumulationchart);
</script>
```

**OPTIONS API (~SRC/APP.VUE)**

```

<template>
<ejs-accumulationchart id="container" :legendSettings="legendSettings"
:tooltip="tooltip">
<e-accumulation-series-collection>
<e-accumulation-series :dataSource='data' xName='x' yName='y'
innerRadius="20%"> </e-accumulation-series>
</e-accumulation-series-collection>
</ejs-accumulationchart>
</template>
<script>
import { AccumulationChartComponent, AccumulationSeriesCollectionDirective,
AccumulationSeriesDirective, AccumulationLegend,
PieSeries, AccumulationTooltip } from "@syncfusion/ej2-vue-charts";
//Component registration
export default {
name: "App",
components: {
"ejs-accumulationchart": AccumulationChartComponent,
"e-accumulation-series-collection": AccumulationSeriesCollectionDirective,
"e-accumulation-series": AccumulationSeriesDirective
},
data() {
return {
data: [
{ x: 'Argentina', y: 505370 },
{ x: 'Belgium', y: 551500 },
{ x: 'Cuba', y: 312685 },
{ x: 'Dominican Republic', y: 350000 },
{ x: 'Egypt', y: 301000 },
{ x: 'Kazakhstan', y: 300000 },
{ x: 'Somalia', y: 357022 }
],
legendSettings: { visible: true },
tooltip: {
enable: true
},
};
},
provide: {
accumulationchart: [ PieSeries, AccumulationLegend, AccumulationTooltip ]
},
};
</script>

```

**Run the project**

To run the project, use the following command:

```
`bash
```

```
npm run dev
```

```
,
```

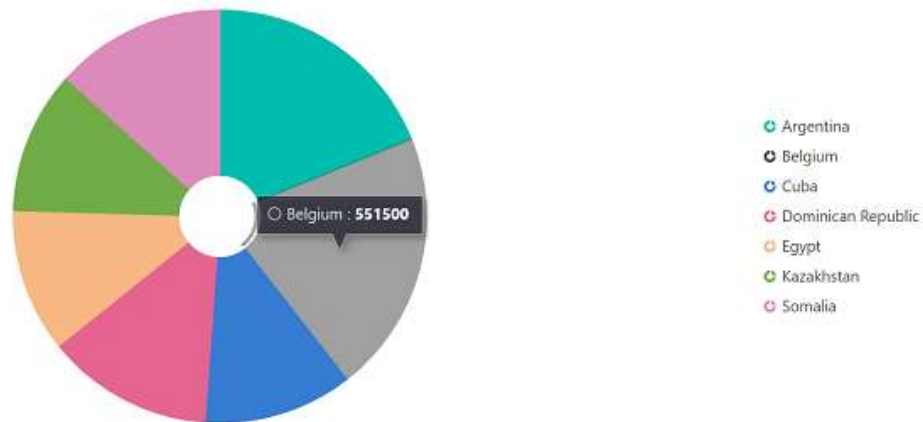
or

```
`bash
```

```
yarn run dev
```

```
,
```

The output will appear as follows:



**Sample:** [vue-3-accumulation-chart-getting-started](#).

For migrating from Vue 2 to Vue 3, refer to the [migration](#) documentation.

See also

- [Getting Started with Vue UI Components using Composition API and TypeScript](#)
- [Getting Started with Vue UI Components using Options API and TypeScript](#)

Pie dough nut in Vue Accumulation chart component

Pie Chart

To render a pie series, use the series [type](#) as `Pie` and inject the `PieSeries` module into the `provide`. If the `PieSeries` module is not injected, this module will be loaded by default.

#### APP.VUE

```
<template>
  <div id="app">
    <ejs-accumulationchart id="container">
      <e-accumulation-series-collection>
        <e-accumulation-series :dataSource='seriesData' xName='x'
yName='y'> </e-accumulation-series>
      </e-accumulation-series-collection>
    </ejs-accumulationchart>
  </div>
</template>
<script>
import Vue from "vue";
```

```

import { AccumulationChartPlugin, PieSeries } from "@syncfusion/ej2-vue-charts";
Vue.use(AccumulationChartPlugin);
export default {
  data() {
    return {
      seriesData: [
        { x: 'Jan', y: 3, fill: '#498fff', text: 'January' }, { x: 'Feb',
y: 3.5, fill: '#ffa060', text: 'February' },
        { x: 'Mar', y: 7, fill: '#ff68b6', text: 'March' }, { x: 'Apr',
y: 13.5, fill: '#81e2a1', text: 'April' }
      ]
    };
  },
  provide: {
    accumulationchart: [PieSeries]
  }
};
</script>
<style>
  #container {
    height: 350px;
  }
</style>

```

{% previewsample "page.domainurl/code-snippet/chart/series/pie-cs20" %}

### Radius Customization

By default, radius of the pie series will be 80% of the size (minimum of chart width and height).

You can customize this using [radius](#) property of the series.

### APP.VUE

```

<template>
  <div id="app">
    <ejs-accumulationchart id="container">
      <e-accumulation-series-collection>
        <e-accumulation-series :dataSource='seriesData' xName='x'
yName='y' radius='90%'> </e-accumulation-series>
      </e-accumulation-series-collection>
    </ejs-accumulationchart>
  </div>
</template>
<script>
import Vue from "vue";
import { AccumulationChartPlugin, PieSeries } from "@syncfusion/ej2-vue-charts";
Vue.use(AccumulationChartPlugin);
export default {
  data() {
    return {
      seriesData: [
        { x: 'Jan', y: 3, fill: '#498fff', text: 'January' }, { x: 'Feb',
y: 3.5, fill: '#ffa060', text: 'February' },
        { x: 'Mar', y: 7, fill: '#ff68b6', text: 'March' }, { x: 'Apr',
y: 13.5, fill: '#81e2a1', text: 'April' }
      ]
    };
  }
};

```

```

    ]
    };
  },
  provide: {
    accumulationchart: [PieSeries]
  }
};
</script>
<style>
#container {
  height: 350px;
}
</style>

```

{% previewsample "page.domainurl/code-snippet/chart/series/pie-cs21" %}

### Pie Center

The center position of the pie can be changed by Center X and Center Y. By default, center value of the pie series x and y is 50%. You can customize this using [center](#) property of the series.

### APP.VUE

```

<template>
  <div id="app">
    <ejs-accumulationchart id='chartcontainer' :title='title'
      :legendSettings='legendSettings' :tooltip='tooltip'
      enableSmartLabels='true' :enableAnimation='enableAnimation'
      :center='center'>
      <e-accumulation-series-collection>
        <e-accumulation-series :dataSource='seriesData'
          :startAngle='startAngle' :endAngle='endAngle' :radius='radius' xName='x'
          yName='y' :dataLabel='dataLabel' name='Browser' innerRadius='0%'> </e-
        accumulation-series>
      </e-accumulation-series-collection>
    </ejs-accumulationchart>
  </div>
</template>
<script>
import Vue from "vue";
import { AccumulationChartPlugin, AccumulationTooltip, PieSeries,
  AccumulationDataLabel, AccumulationLegend } from "@syncfusion/ej2-vue-
  charts";
Vue.use(AccumulationChartPlugin);
export default {
  data() {
    return {
      seriesData: [
        { 'x': 'Chrome', y: 37, text: '37%' }, { 'x': 'UC Browser', y: 17,
        text: '17%' },
        { 'x': 'iPhone', y: 19, text: '19%' },
        { 'x': 'Others', y: 4, text: '4%' }, { 'x': 'Opera', y: 11, text:
        '11%' },
        { 'x': 'Android', y: 12, text: '12%' }
      ],
      dataLabel: {
        visible: true,

```

```

        position: 'Inside', name: 'text',
        font: {
            fontWeight: '600'
        }
    },
    enableSmartLabels: true,
    enableAnimation: false,
    legendSettings: {
        visible: false,
    },
    tooltip: { enable: false, format: '${point.x} : <b>${point.y}%</b>' },
    startAngle: '0',
    endAngle: '360',
    radius: '70%',
    explodeOffset: '10%',
    explodeIndex : 0,
    center: {x: '50%', y: '50%'},
    title: "Mobile Browser Statistics"
    };
    },
    provide: {
        accumulationchart: [PieSeries, AccumulationDataLabel]
    }
    };
</script>
<style>
    #container {
        height: 350px;
    }
</style>

```

{% previewsample "page.domainurl/code-snippet/chart/series/pie-cs22" %}

### Various Radius Pie Chart

You can use radius mapping to render the slice with different [radius](#) pie and also use [border](#), fill properties to customize the point. dataLabel is used to represent individual data and its value.

### APP.VUE

```

<template>
    <div id="app">
        <ejs-accumulationchart id="container" ref="pie"
        style='display:block;' :legendSettings="legendSettings" :tooltip="tooltip"
        :enableAnimation='enableAnimation' :enableSmartLabels='enableSmartLabels' >
            <e-accumulation-series-collection>
                <e-accumulation-series :dataSource='data' xName='x'
                yName='y' :radius='radius' innerRadius="20%" :dataLabel="dataLabel"> </e-
                accumulation-series>
            </e-accumulation-series-collection>
        </ejs-accumulationchart>
    </div>
</template>
<script>
import Vue from "vue";
import { AccumulationChartPlugin, AccumulationDataLabel, PieSeries } from
"@syncfusion/ej2-vue-charts";

```



```

Vue.use(AccumulationChartPlugin);
export default {
  data() {
    return {
      data:[
        { x: 'Argentina', y: 505370, r: '100' },
        { x: 'Belgium', y: 551500, r: '118.7' },
        { x: 'Cuba', y: 312685, r: '124.6' },
        { x: 'Dominican Republic', y: 350000, r: '137.5' },
        { x: 'Egypt', y: 301000, r: '150.8' },
        { x: 'Kazakhstan', y: 300000, r: '155.5' },
        { x: 'Somalia', y: 357022, r: '160.6' }
      ],
      radius: 'r',
      legendSettings: { visible: true },
      dataLabel: { visible: true, position: 'Outside', name: 'x' },
      tooltip: {
        enable: true, header: '<b>${point.x}</b>', format: 'Composition:
        <b>${point.y}</b>'
      },
      enableAnimation: true,
      enableSmartLabels: true
    };
  },
  provide: {
    accumulationchart: [PieSeries, AccumulationDataLabel]
  }
};
</script>
<style>
  #container {
    height: 350px;
  }
</style>

```

{% previewsample "page.domainurl/code-snippet/chart/series/pie-cs23" %}

### Doughnut Chart

To achieve a doughnut in pie series, customize the [innerRadius](#) property of the series. By setting value greater than 0%, a doughnut will appear. The innerRadius property takes value from 0% to 100% of the pie radius.

### APP.VUE

```

<template>
  <div id="app">
    <ejs-accumulationchart id="container">
      <e-accumulation-series-collection>
        <e-accumulation-series :dataSource='seriesData' xName='x'
        yName='y' innerRadius='40%'> </e-accumulation-series>
      </e-accumulation-series-collection>
    </ejs-accumulationchart>
  </div>
</template>
<script>
import Vue from "vue";

```

```

import { AccumulationChartPlugin, PieSeries } from "@syncfusion/ej2-vue-charts";
Vue.use(AccumulationChartPlugin);
export default {
  data() {
    return {
      seriesData: [
        { x: 'Jan', y: 3, fill: '#498fff', text: 'January' }, { x: 'Feb',
y: 3.5, fill: '#ffa060', text: 'February' },
        { x: 'Mar', y: 7, fill: '#ff68b6', text: 'March' }, { x: 'Apr',
y: 13.5, fill: '#81e2a1', text: 'April' }
      ]
    };
  },
  provide: {
    accumulationchart: [PieSeries]
  }
};
</script>
<style>
  #container {
    height: 350px;
  }
</style>

```

{% previewsample "page.domainurl/code-snippet/chart/series/pie-cs24" %}

### Start and End angles

You can customize the start and end angle of the pie series using the [startAngle](#) and [endAngle](#) properties. The default value of [startAngle](#) is 0 degree, and [endAngle](#) is 360 degrees. By customizing this, you can achieve a semi pie series.

### APP.VUE

```

<template>
  <div id="app">
    <ejs-accumulationchart id="container">
      <e-accumulation-series-collection>
        <e-accumulation-series :dataSource='seriesData' xName='x'
yName='y' startAngle=270 endAngle=90> </e-accumulation-series>
      </e-accumulation-series-collection>
    </ejs-accumulationchart>
  </div>
</template>
<script>
import Vue from "vue";
import { AccumulationChartPlugin, PieSeries } from "@syncfusion/ej2-vue-charts";
Vue.use(AccumulationChartPlugin);
export default {
  data() {
    return {
      seriesData: [
        { x: 'Jan', y: 3, fill: '#498fff', text: 'January' }, { x: 'Feb',
y: 3.5, fill: '#ffa060', text: 'February' },

```

```

        { x: 'Mar', y: 7, fill: '#ff68b6', text: 'March' }, { x: 'Apr',
y: 13.5, fill: '#81e2a1', text: 'April' }
    ]
    };
  },
  provide: {
    accumulationchart: [PieSeries]
  }
};
</script>
<style>
  #container {
    height: 350px;
  }
</style>

```

{% previewsample "page.domainurl/code-snippet/chart/series/pie-cs25" %}

### Color and Text Mapping

The fill color and the text in the data source can be mapped to the chart using `pointColorMapping` in series and `name` in `datalabel` respectively.

### APP.VUE

```

<template>
  <div id="app">
    <ejs-accumulationchart id="container">
      <e-accumulation-series-collection>
        <e-accumulation-series :dataSource='seriesData' xName='x'
yName='y' :dataLabel='datalabel' :pointColorMapping=' pointColorMapping'>
      </e-accumulation-series>
    </e-accumulation-series-collection>
  </ejs-accumulationchart>
  </div>
</template>
<script>
import Vue from "vue";
import { AccumulationChartPlugin, PieSeries, AccumulationDataLabel } from
"@syncfusion/ej2-vue-charts";
Vue.use(AccumulationChartPlugin);
export default {
  data() {
    return {
      seriesData: [
        { x: 'Jan', y: 3, fill: '#498fff', text: 'January' }, { x:
'Feb', y: 3.5, fill: '#ffa060', text: 'February' },
        { x: 'Mar', y: 7, fill: '#ff68b6', text: 'March' }, { x:
'Apr', y: 13.5, fill: '#81e2a1', text: 'April' }
      ],
      datalabel: { visible: true, name: 'text' },
      pointColorMapping: 'fill'
    };
  },
  provide: {
    accumulationchart: [PieSeries, AccumulationDataLabel]
  }
}

```

```
};
</script>
<style>
  #container {
    height: 350px;
  }
</style>
```

```
{% previewsample "page.domainurl/code-snippet/chart/series/pie-cs26" %}
```

### Multi-level pie chart

You can achieve a multi-level drill down in pie and doughnut charts using [pointClick](#) event. If user clicks any point in the chart, that corresponding data will be shown in the next level and so on based on point clicked.

You can also achieve drill-up (back to the initial state) by using [chartMouseClicked](#) event. In below sample, you can drill-up by clicking back button in the center of the chart.

### APP.VUE

```
<template>
  <div id="app">
    <div>
      <div id="link">
        <a id="category" @click="onClick" style="visibility:hidden;
display:inline-block">Sales by Category</a>
        <p style="visibility:hidden; display:inline-block" id="symbol">
>> </p>
        <p id="text" style="display:inline-block;"></p>
      </div>
      <button type="button" id="back" style="visibility: hidden;"
@click="onClick">Back</button>
      <ejs-accumulationchart ref="pie" id="container"
style='display:block;' :legendSettings="legendSettings"
:enableSmartLabels='enableSmartLabels' :title="title"
:textRender="onTextRender" :chartMouseClicked="onChartMouseClicked">
        <e-accumulation-series-collection>
          <e-accumulation-series :dataSource='data' xName='x'
yName='y' :startAngle="startAngle" :endAngle="endAngle"
:innerRadius="innerRadius" radius="70%" :dataLabel="dataLabel"
:explode="isExplode" explodeOffset='10%' :explodeIndex='explodeIndex'>
            </e-accumulation-series>
          </e-accumulation-series-collection>
        </ejs-accumulationchart>
      </div>
    </div>
  </template>
<script>
import Vue from "vue";
import { extend } from '@syncfusion/ej2-base';
import { getElement, indexFinder, AccumulationLegend, PieSeries,
AccumulationTooltip, AccumulationDataLabel, AccumulationChartPlugin } from
"@syncfusion/ej2-vue-charts";
Vue.use(AccumulationChartPlugin);
export default {
  data() {
```

```

return {
  innerRadius: '0%',
  innerChart: false,
  enableSmartLabels: false,
  initialContent: null,
  data: [
    { x: 'SUV', y: 25 }, { x: 'Car', y: 37 }, { x: 'Pickup', y: 15 },
    { x: 'Minivan', y: 23 }
  ],
  suvs: [{ x: 'Toyota', y: 8 }, { x: 'Ford', y: 12 }, { x: 'GM', y: 17 },
  { x: 'Renault', y: 6 }, { x: 'Fiat', y: 3 },
  { x: 'Hyundai', y: 16 }, { x: 'Honda', y: 8 }, { x: 'Maruthi', y: 10 },
  { x: 'BMW', y: 20 }],
  cars: [{ x: 'Toyota', y: 7 }, { x: 'Chrysler', y: 12 }, { x: 'Nissan',
  y: 9 }, { x: 'Ford', y: 15 },
  { x: 'Tata', y: 10 },
  { x: 'Mahindra', y: 7 }, { x: 'Renault', y: 8 }, { x: 'Skoda', y: 5 }, {
  x: 'Volkswagen', y: 15 }, { x: 'Fiat', y: 3 }],
  pickups: [{ x: 'Nissan', y: 9 }, { x: 'Chrysler', y: 4 }, { x: 'Ford',
  y: 7 }, { x: 'Toyota', y: 20 },
  { x: 'Suzuki', y: 13 }, { x: 'Lada', y: 12 }, { x: 'Bentley', y: 6 }, {
  x: 'Volvo', y: 10 }, { x: 'Audi', y: 19 }],
  minivans: [{ x: 'Hummer', y: 11 }, { x: 'Ford', y: 5 }, { x: 'GM', y: 12
  }, { x: 'Chrysler', y: 3 },
  { x: 'Jaguar', y: 9 },
  { x: 'Fiat', y: 8 }, { x: 'Honda', y: 15 }, { x: 'Hyundai', y: 4 }, { x:
  'Scion', y: 11 }, { x: 'Toyota', y: 17 }],
  legendSettings: {
    visible: false,
  },
  dataLabel: {
    visible: true, position: 'Inside', connectorStyle: { type: 'Curve',
    length: '5%' }, font: { size: '14px', color: 'white' }
  },
  startAngle: 0,
  explodeIndex: 2,
  isExplode: false,
  endAngle: 360,
  title: 'Automobile Sales by Category'
},
provide: {
  accumulationchart: [AccumulationLegend, PieSeries, AccumulationTooltip,
  AccumulationDataLabel]
},
methods: {
  onTextRender: function (args) {
    args.text = args.point.x + ' ' + args.point.y + ' %';
  },
  onChartMouseClicked: function (args) {
    let index = indexFinder(args.target);
    this.isExplode = false;
    if (document.getElementById('container_Series_' + index.series +
    '_Point_' + index.point) && !this.innerChart) {
      this.innerRadius = '30%';
      switch (index.point) {
        case 0:

```

```

        this.data = this.suvs;
        this.title = 'Automobile Sales in the SUV Segment';
        document.getElementById('text').innerHTML = 'SUV';
        break;
    case 1:
        this.data = this.cars;
        this.title = 'Automobile Sales in the Car Segment';
        document.getElementById('text').innerHTML = 'Car';
        break;
    case 2:
        this.data = this.pickups;
        this.title = 'Automobile Sales in the Pickup Segment';
        document.getElementById('text').innerHTML = 'Pickup';
        break;
    case 3:
        this.data = this.minivans;
        this.title = 'Automobile Sales in the Minivan Segment';
        document.getElementById('text').innerHTML = 'Minivan';
        break;
    }
    let dataLabel = extend({}, this.dataLabel);
    dataLabel.position = 'Outside';
    dataLabel.font.color = 'black';
    this.dataLabel = dataLabel;
    let legendSettings = this.legendSettings;
    legendSettings.visible = false;
    this.legendSettings = legendSettings;
    this.enableSmartLabels = true;
    document.getElementById('category').style.visibility =
'visible';
    document.getElementById('symbol').style.visibility = 'visible';
    document.getElementById('text').style.visibility = 'visible';
    this.innerChart = true;
}
if (args.target.indexOf('back') > -1) {
    this.data = [{ x: 'SUV', y: 25 }, { x: 'Car', y: 37 }, { x:
'Pickup', y: 15 }, { x: 'Minivan', y: 23 }]
    this.isExplode = false;
    let dataLabel = extend({}, this.dataLabel);
    dataLabel.position = 'Inside';
    dataLabel.font.color = 'white';
    this.dataLabel = dataLabel;
    let legendSettings = this.legendSettings;
    legendSettings.visible = false;
    this.legendSettings = legendSettings;
    this.enableSmartLabels = false;
    this.innerRadius = '0%';
    getElement('category').style.visibility = 'hidden';
    document.getElementById('symbol').style.visibility = 'hidden';
    document.getElementById('text').style.visibility = 'hidden';
    this.innerChart = false;
}
},
onClick: function (e) {
    this.isExplode = false;
    this.data = [{ x: 'SUV', y: 25 }, { x: 'Car', y: 37 }, { x:
'Pickup', y: 15 }, { x: 'Minivan', y: 23 }]

```

```

        let dataLabel = extend({}, this.dataLabel);
        dataLabel.position = 'Inside';
        dataLabel.font.color = 'white';
        this.dataLabel = dataLabel;
        let legendSettings = this.legendSettings;
        legendSettings.visible = false;
        this.legendSettings = legendSettings;
        this.enableSmartLabels = false;
        this.innerRadius = '0%';
        getElement('category').style.visibility = 'hidden';
        document.getElementById('symbol').style.visibility = 'hidden';
        document.getElementById('text').style.visibility = 'hidden';
        e.target.style.visibility = 'hidden';
        document.getElementById('symbol').style.visibility = 'hidden';
        document.getElementById('text').style.visibility = 'hidden';
        this.innerChart = false;
    },
},
    updated: function() {
        this.$nextTick(function() {
            this.$refs.pie.ej2Instances.refresh();
        });
    }
};
</script>
<style scoped>
#category: hover {
    cursor: pointer;
}
</style>

```

{% previewsample "page.domainurl/code-snippet/chart/axis/category-cs1" %}

### Pyramid in Vue Accumulation chart component

To render a pyramid series, use the series [type](#) as **Pyramid** and inject **PyramidSeries** module into the **provide**.

#### APP.VUE

```

<template>
  <div id="app">
    <ejs-accumulationchart id="container">
      <e-accumulation-series-collection>
        <e-accumulation-series :dataSource='seriesData'
type='Pyramid' xName='x' yName='y'> </e-accumulation-series>
      </e-accumulation-series-collection>
    </ejs-accumulationchart>
  </div>
</template>
<script>
import Vue from "vue";
import { AccumulationChartPlugin, PyramidSeries } from "@syncfusion/ej2-vue-
charts";
Vue.use(AccumulationChartPlugin);
export default {
  data() {

```

```

    return {
      seriesData: [
        { x: 'Australia', y: 20, text: 'Australia 20%' },
        { x: 'France', y: 22, text: 'France 22%' },
        { x: 'China', y: 23, text: 'China 23%' },
        { x: 'India', y: 24, text: 'India 24%' },
        { x: 'Japan', y: 25, text: 'Japan 25%' },
        { x: 'Germany', y: 27, text: 'Germany 27%' }
      ]
    };
  },
  provide: {
    accumulationchart: [PyramidSeries]
  }
};
</script>
<style>
  #container {
    height: 350px;
  }
</style>

```

{% previewsample "page.domainurl/code-snippet/chart/series/pyramid-cs1" %}

### Mode

The Pyramid chart supports linear and surface modes of rendering. The default type of the `pyramidMode` is `linear`.

### APP.VUE

```

<template>
  <div id="app">
    <ejs-accumulationchart id="container">
      <e-accumulation-series-collection>
        <e-accumulation-series :dataSource='seriesData'
type='Pyramid' xName='x' yName='y' pyramidMode='Surface'> </e-accumulation-
series>
      </e-accumulation-series-collection>
    </ejs-accumulationchart>
  </div>
</template>
<script>
import Vue from "vue";
import { AccumulationChartPlugin, PyramidSeries } from "@syncfusion/ej2-vue-
charts";
Vue.use(AccumulationChartPlugin);
export default {
  data() {
    return {
      seriesData: [
        { x: 'Australia', y: 20, text: 'Australia 20%' },
        { x: 'France', y: 22, text: 'France 22%' },
        { x: 'China', y: 23, text: 'China 23%' },
        { x: 'India', y: 24, text: 'India 24%' },
        { x: 'Japan', y: 25, text: 'Japan 25%' },
        { x: 'Germany', y: 27, text: 'Germany 27%' }
      ]
    }
  }
}

```



```

    ];
  },
  provide: {
    accumulationchart: [PyramidSeries]
  }
};
</script>
<style>
  #container {
    height: 350px;
  }
</style>

```

{% previewsample "page.domainurl/code-snippet/chart/series/pyramid-cs2" %}

### Size

The size of the pyramid chart can be customized by using the **width** and **height** properties.

### APP.VUE

```

<template>
  <div id="app">
    <ejs-accumulationchart id="container">
      <e-accumulation-series-collection>
        <e-accumulation-series :dataSource='seriesData'
type='Pyramid' xName='x' yName='y' width='60%' height='80%'> </e-
accumulation-series>
      </e-accumulation-series-collection>
    </ejs-accumulationchart>
  </div>
</template>
<script>
import Vue from "vue";
import { AccumulationChartPlugin, PyramidSeries } from "@syncfusion/ej2-vue-
charts";
Vue.use(AccumulationChartPlugin);
export default {
  data() {
    return {
      seriesData: [
        { x: 'Australia', y: 20, text: 'Australia 20%' },
        { x: 'France', y: 22, text: 'France 22%' },
        { x: 'China', y: 23, text: 'China 23%' },
        { x: 'India', y: 24, text: 'India 24%' },
        { x: 'Japan', y: 25, text: 'Japan 25%' },
        { x: 'Germany', y: 27, text: 'Germany 27%' }
      ]
    };
  },
  provide: {
    accumulationchart: [PyramidSeries]
  }
};
</script>
<style>

```

```
#container {
  height: 350px;
}
</style>
```

{% previewsample "page.domainurl/code-snippet/chart/series/pyramid-cs3" %}

### Gap Between the Segments

Pyramid chart provides options to customize the space between the segments by using the `gapRatio` property of the series. It ranges from 0 to 1.

### APP.VUE

```
<template>
  <div id="app">
    <ejs-accumulationchart id="container">
      <e-accumulation-series-collection>
        <e-accumulation-series :dataSource='seriesData'
type='Pyramid' xName='x' yName='y' :gapRatio='gapRatio'> </e-accumulation-
series>
      </e-accumulation-series-collection>
    </ejs-accumulationchart>
  </div>
</template>
<script>
import Vue from "vue";
import { AccumulationChartPlugin, PyramidSeries } from "@syncfusion/ej2-vue-
charts";
Vue.use(AccumulationChartPlugin);
export default {
  data() {
    return {
      seriesData: [
        { x: 'Australia', y: 20, text: 'Australia 20%' },
        { x: 'France', y: 22, text: 'France 22%' },
        { x: 'China', y: 23, text: 'China 23%' },
        { x: 'India', y: 24, text: 'India 24%' },
        { x: 'Japan', y: 25, text: 'Japan 25%' },
        { x: 'Germany', y: 27, text: 'Germany 27%' }
      ],
      gapRatio: 0.08
    };
  },
  provide: {
    accumulationchart: [PyramidSeries]
  }
};
</script>
<style>
  #container {
    height: 350px;
  }
</style>
```

{% previewsample "page.domainurl/code-snippet/chart/series/pyramid-cs4" %}

## Explode

Points can be exploded on mouse click by setting the `explode` property to true. You can also explode the point on load using `explodeIndex`. Explode distance can be set by using `explodeOffset` property.

### APP.VUE

```
<template>
  <div id="app">
    <ejs-accumulationchart id="container">
      <e-accumulation-series-collection>
        <e-accumulation-series :dataSource='seriesData'
type='Pyramid' xName='x' yName='y' explode=true
        explodeOffset='10' :explodeIndex='explodeIndex'> </e-
accumulation-series>
      </e-accumulation-series-collection>
    </ejs-accumulationchart>
  </div>
</template>
<script>
import Vue from "vue";
import { AccumulationChartPlugin, PyramidSeries } from "@syncfusion/ej2-vue-
charts";
Vue.use(AccumulationChartPlugin);
export default {
  data() {
    return {
      seriesData: [
        { x: 'Australia', y: 20, text: 'Australia 20%' },
        { x: 'France', y: 22, text: 'France 22%' },
        { x: 'China', y: 23, text: 'China 23%' },
        { x: 'India', y: 24, text: 'India 24%' },
        { x: 'Japan', y: 25, text: 'Japan 25%' },
        { x: 'Germany', y: 27, text: 'Germany 27%' }
      ],
      explodeIndex:3
    };
  },
  provide: {
    accumulationchart: [PyramidSeries]
  }
};
</script>
<style>
#container {
  height: 350px;
}
</style>
```

{% previewsample "page.domainurl/code-snippet/chart/series/pyramid-cs5" %}

## Customization

Individual points can be customized using the `pointRender` event.

### APP.VUE

```
<template>
```

```

<div id="app">
  <ejs-accumulationchart id="container" :pointRender='onPointRender'>
    <e-accumulation-series-collection>
      <e-accumulation-series :dataSource='seriesData'
type='Pyramid' xName='x' yName='y'> </e-accumulation-series>
    </e-accumulation-series-collection>
  </ejs-accumulationchart>
</div>
</template>
<script>
import Vue from "vue";
import { AccumulationChartPlugin, PyramidSeries } from "@syncfusion/ej2-vue-
charts";
Vue.use(AccumulationChartPlugin);
export default {
  data() {
    return {
      seriesData: [
        { x: 'Australia', y: 20, text: 'Australia 20%' },
        { x: 'France', y: 22, text: 'France 22%' },
        { x: 'China', y: 23, text: 'China 23%' },
        { x: 'India', y: 24, text: 'India 24%' },
        { x: 'Japan', y: 25, text: 'Japan 25%' },
        { x: 'Germany', y: 27, text: 'Germany 27%' }
      ]
    };
  },
  provide: {
    accumulationchart: [PyramidSeries]
  },
  methods: {
    onPointRender: function (args) {
      if ((args.point.x as string).indexOf('China') > -1) {
        args.fill = '#f4bc42';
      }
      else {
        args.fill = '#597cf9';
      }
    }
  }
};
</script>
<style>
#container {
  height: 350px;
}
</style>

```

{% previewsample "page.domainurl/code-snippet/chart/series/pyramid-cs6" %}

### Funnel in Vue Accumulation chart component

To render a funnel series, use the series [type](#) as `Funnel` and inject, the `FunnelSeries` module into the `provide`.

#### APP.VUE

```

<template>
  <div id="app">
    <ejs-accumulationchart id="container">
      <e-accumulation-series-collection>
        <e-accumulation-series :dataSource='seriesData'
type='Funnel' xName='x' yName='y'> </e-accumulation-series>
      </e-accumulation-series-collection>
    </ejs-accumulationchart>
  </div>
</template>
<script>
import Vue from "vue";
import { AccumulationChartPlugin, FunnelSeries } from "@syncfusion/ej2-vue-
charts";
Vue.use(AccumulationChartPlugin);
export default {
  data() {
    return {
      seriesData: [
        { x: 'Renewed', y: 18.20, text: 'Renewed 18.20%' },
        { x: 'Subscribe', y: 27.3, text: 'Subscribe 27.3%' },
        { x: 'Support', y: 55.9, text: 'Support 55.9%' },
        { x: 'Downloaded', y: 76.8, text: 'Downloaded 76.8%' },
        { x: 'Visited', y: 100, text: 'Visited 100%' }
      ]
    };
  },
  provide: {
    accumulationchart: [FunnelSeries]
  }
};
</script>
<style>
  #container {
    height: 350px;
  }
</style>

```

{% previewsample "page.domainurl/code-snippet/chart/series/funnel-cs1" %}

### Size

The size of the funnel chart can be customized by using the `width` and `height` properties.

### APP.VUE

```

<template>
  <div id="app">
    <ejs-accumulationchart id="container">
      <e-accumulation-series-collection>
        <e-accumulation-series :dataSource='seriesData'
type='Funnel' xName='x' yName='y' width='60%' height='80%'> </e-
accumulation-series>
      </e-accumulation-series-collection>
    </ejs-accumulationchart>
  </div>
</template>

```

```

<script>
import Vue from "vue";
import { AccumulationChartPlugin, FunnelSeries } from "@syncfusion/ej2-vue-charts";
Vue.use(AccumulationChartPlugin);
export default {
  data() {
    return {
      seriesData: [
        { x: 'Renewed', y: 18.20, text: 'Renewed 18.20%' },
        { x: 'Subscribe', y: 27.3, text: 'Subscribe 27.3%' },
        { x: 'Support', y: 55.9, text: 'Support 55.9%' },
        { x: 'Downloaded', y: 76.8, text: 'Downloaded 76.8%' },
        { x: 'Visited', y: 100, text: 'Visited 100%' }
      ]
    };
  },
  provide: {
    accumulationchart: [FunnelSeries]
  }
};
</script>
<style>
#container {
  height: 350px;
}
</style>

```

{% previewsample "page.domainurl/code-snippet/chart/series/funnel-cs2" %}

### Neck Size

The funnel's neck size can be customized by using the `neckWidth` and `neckHeight` properties.

### APP.VUE

```

<template>
  <div id="app">
    <ejs-accumulationchart id="container">
      <e-accumulation-series-collection>
        <e-accumulation-series :dataSource='seriesData'
type='Funnel' xName='x' yName='y' :neckWidth='neckWidth'
:neckHeight='neckHeight'> </e-accumulation-series>
      </e-accumulation-series-collection>
    </ejs-accumulationchart>
  </div>
</template>
<script>
import Vue from "vue";
import { AccumulationChartPlugin, FunnelSeries } from "@syncfusion/ej2-vue-charts";
Vue.use(AccumulationChartPlugin);
export default {
  data() {
    return {
      seriesData: [
        { x: 'Renewed', y: 18.20, text: 'Renewed 18.20%' },

```

```

        { x: 'Subscribe', y: 27.3, text: 'Subscribe 27.3%' },
        { x: 'Support', y: 55.9, text: 'Support 55.9%' },
        { x: 'Downloaded', y: 76.8, text: 'Downloaded 76.8%' },
        { x: 'Visited', y: 100, text: 'Visited 100%' }
    ],
    neckWidth: '25%', neckHeight: '5%'
  };
},
provide: {
  accumulationchart: [FunnelSeries]
}
};
</script>
<style>
  #container {
    height: 350px;
  }
</style>

```

{% previewsample "page.domainurl/code-snippet/chart/series/funnel-cs3" %}

### Gap between the segments

Funnel chart provides options to customize the space between the segments by using the `gapRatio` property of the series. It ranges from 0 to 1.

### APP.VUE

```

<template>
  <div id="app">
    <ejs-accumulationchart id="container">
      <e-accumulation-series-collection>
        <e-accumulation-series :dataSource='seriesData'
type='Funnel' xName='x' yName='y' :gapRatio='gapRatio'> </e-accumulation-
series>
      </e-accumulation-series-collection>
    </ejs-accumulationchart>
  </div>
</template>
<script>
import Vue from "vue";
import { AccumulationChartPlugin, FunnelSeries } from "@syncfusion/ej2-vue-
charts";
Vue.use(AccumulationChartPlugin);
export default {
  data() {
    return {
      seriesData: [
        { x: 'Renewed', y: 18.20, text: 'Renewed 18.20%' },
        { x: 'Subscribe', y: 27.3, text: 'Subscribe 27.3%' },
        { x: 'Support', y: 55.9, text: 'Support 55.9%' },
        { x: 'Downloaded', y: 76.8, text: 'Downloaded 76.8%' },
        { x: 'Visited', y: 100, text: 'Visited 100%' }
      ],
      gapRatio: 0.08
    };
  },
};

```

```

    provide: {
      accumulationchart: [FunnelSeries]
    }
  };
</script>
<style>
  #container {
    height: 350px;
  }
</style>

```

{% previewsample "page.domainurl/code-snippet/chart/series/funnel-cs4" %}

### Explode

Points can be exploded on mouse click by setting the **explode** property to true. You can also explode the point

on load using **explodeIndex**. Explode distance can be set by using **explodeOffset** property.

### APP.VUE

```

<template>
  <div id="app">
    <ejs-accumulationchart id="container">
      <e-accumulation-series-collection>
        <e-accumulation-series :dataSource='seriesData'
type='Funnel' xName='x' yName='y' explode=true
        explodeOffset='10' :explodeIndex='explodeIndex'> </e-
accumulation-series>
      </e-accumulation-series-collection>
    </ejs-accumulationchart>
  </div>
</template>
<script>
import Vue from "vue";
import { AccumulationChartPlugin, FunnelSeries } from "@syncfusion/ej2-vue-
charts";
Vue.use(AccumulationChartPlugin);
export default {
  data() {
    return {
      seriesData: [
        { x: 'Renewed', y: 18.20, text: 'Renewed 18.20%' },
        { x: 'Subscribe', y: 27.3, text: 'Subscribe 27.3%' },
        { x: 'Support', y: 55.9, text: 'Support 55.9%' },
        { x: 'Downloaded', y: 76.8, text: 'Downloaded 76.8%' },
        { x: 'Visited', y: 100, text: 'Visited 100%' }
      ],
      explodeIndex:3
    };
  },
  provide: {
    accumulationchart: [FunnelSeries]
  }
};
</script>

```



```
<style>
  #container {
    height: 350px;
  }
</style>
```

{% previewsample "page.domainurl/code-snippet/chart/series/funnel-cs5" %}

### Smart Data Label

It provides the data label smart arrangement of the funnel and pyramid series. The overlap data label will be placed on left side of the funnel/pyramid series.

### APP.VUE

```
<template>
  <div id="app">
    <ejs-accumulationchart id="container"
      :legendSettings="legendSettings" :title="title" :tooltip="tooltip"
      :load="load" :resized="resized" >
      <e-accumulation-series-collection>
        <e-accumulation-series :dataSource='seriesData'
          type='Funnel' xName='x' yName='y' neckWidth='15%'
            neckHeight='18%' name='2017 Population' explode="false"
            :dataLabel="dataLabel"> </e-accumulation-series>
        </e-accumulation-series-collection>
      </ejs-accumulationchart>
    </div>
  </template>
<script>
import Vue from "vue";
import { AccumulationChartPlugin, AccumulationLegend, FunnelSeries,
  AccumulationTooltip, AccumulationDataLabel } from "@syncfusion/ej2-vue-
  charts";
Vue.use(AccumulationChartPlugin);
export default {
  data() {
    return {
      seriesData: [
        { 'x': 'China', y: 1409517397, text: 'China' },
        { 'x': 'India', y: 1339180127, text: 'India' },
        { 'x': 'United States', y: 324459463, text: 'United States' },
        { 'x': 'Indonesia', y: 263991379, text: 'Indonesia' },
        { 'x': 'Brazil', y: 209288278, text: 'Brazil' },
        { 'x': 'Pakistan', y: 197015955, text: 'Pakistan' },
        { 'x': 'Nigeria', y: 190886311, text: 'Nigeria' },
        { 'x': 'Bangladesh', y: 164669751, text: 'Bangladesh' },
        { 'x': 'Russia', y: 143989754, text: 'Russia' },
        { 'x': 'Mexico', y: 129163276, text: 'Mexico' },
        { 'x': 'Japan', y: 127484450, text: 'Japan' },
        { 'x': 'Ethiopia', y: 104957438, text: 'Ethiopia' },
        { 'x': 'Philippines', y: 104918090, text: 'Philippines' },
        { 'x': 'Egypt', y: 97553151, text: 'Egypt' },
        { 'x': 'Vietnam', y: 95540800, text: 'Vietnam' },
        { 'x': 'Germany', y: 82114224, text: 'Germany' }],
      legendSettings: {visible: false},
      title: 'Top population countries in the world 2017',
```

```

        tooltip: { enable: true, format: '${point.x} : <b>${point.y}</b>' },
        dataLabel: {
            visible: true, position: 'Outside',
            connectorStyle: { length: '6%' }, name: 'text',
        },
    },
    };
},
provide: {
    accumulationchart: [FunnelSeries, AccumulationTooltip,
        AccumulationDataLabel, AccumulationLegend]
},
methods: {
    load: function(args) {
        if (args.accumulation.availableSize) {
            if (args.accumulation.availableSize.width <
args.accumulation.availableSize.height) {
                args.accumulation.series[0].width = '80%';
                args.accumulation.series[0].height = '70%';
            }
        }
    },
    resized: function(args) {
        let bounds =
document.getElementById('container').getBoundingClientRect();
        if (bounds.width < bounds.height) {
            args.accumulation.series[0].width = '80%';
            args.accumulation.series[0].height = '70%';
        } else {
            args.accumulation.series[0].width = '60%';
            args.accumulation.series[0].height = '80%';
        }
    },
},
    };
</script>
<style>
    #container {
        height: 350px;
    }
</style>

```

{% previewsample "page.domainurl/code-snippet/chart/series/funnel-cs6" %}

### Customization

Individual points can be customized using the `pointRender` event.

### APP.VUE

```

<template>
    <div id="app">
        <ejs-accumulationchart id="container" :pointRender='onPointRender'>
            <e-accumulation-series-collection>
                <e-accumulation-series :dataSource='seriesData'
type='Funnel' xName='x' yName='y'> </e-accumulation-series>
            </e-accumulation-series-collection>
        </ejs-accumulationchart>
    </div>
</template>

```

```

    </div>
  </template>
  <script>
import Vue from "vue";
import { AccumulationChartPlugin, FunnelSeries } from "@syncfusion/ej2-vue-charts";
Vue.use(AccumulationChartPlugin);
export default {
  data() {
    return {
      seriesData: [
        { x: 'Renewed', y: 18.20, text: 'Renewed 18.20%' },
        { x: 'Subscribe', y: 27.3, text: 'Subscribe 27.3%' },
        { x: 'Support', y: 55.9, text: 'Support 55.9%' },
        { x: 'Downloaded', y: 76.8, text: 'Downloaded 76.8%' },
        { x: 'Visited', y: 100, text: 'Visited 100%' }
      ]
    };
  },
  provide: {
    accumulationchart: [FunnelSeries]
  },
  methods: {
    onPointRender: function (args) {
      if ((args.point.x as string).indexOf('Downloaded') > -1) {
        args.fill = '#f4bc42';
      }
      else {
        args.fill = '#597cf9';
      }
    }
  }
};
  </script>
  <style>
    #container {
      height: 350px;
    }
  </style>

```

{% previewsample "page.domainurl/code-snippet/chart/series/funnel-cs7" %}

See Also

- [Data label](#)
- [Grouping](#)

### Data label in Vue Accumulation chart component

Data label can be added to a chart series by enabling the [visible](#) option in the dataLabel property.

#### APP.VUE

```

<template>
  <div id="app">

```

```

        <ejs-accumulationchart id="container"
enableSmartLabels='enableSmartLabels'>
            <e-accumulation-series-collection>
                <e-accumulation-series :dataSource='seriesData' xName='x'
yName='y' :dataLabel='datalabel'> </e-accumulation-series>
            </e-accumulation-series-collection>
        </ejs-accumulationchart>
    </div>
</template>
<script>
import Vue from "vue";
import { AccumulationChartPlugin, PieSeries, AccumulationDataLabel } from
"@syncfusion/ej2-vue-charts";
Vue.use(AccumulationChartPlugin);
export default {
    data() {
        return {
            seriesData: [
                { x: 'Jan', y: 3, text: 'Jan: 3' }, { x: 'Feb', y: 3.5,
text: 'Feb: 3.5' },
                { x: 'Mar', y: 7, text: 'Mar: 7' }, { x: 'Apr', y: 13.5,
text: 'Apr: 13.5' },
                { x: 'May', y: 19, text: 'May: 19' }, { x: 'Jun', y: 23.5,
text: 'Jun: 23.5' },
                { x: 'Jul', y: 26, text: 'Jul: 26' }, { x: 'Aug', y: 25,
text: 'Aug: 25' },
                { x: 'Sep', y: 21, text: 'Sep: 21' }, { x: 'Oct', y: 15,
text: 'Oct: 15' }
            ],
            datalabel: { visible: true, name: 'text' },
            enableSmartLabels: true
        };
    },
    provide: {
        accumulationchart: [PieSeries, AccumulationDataLabel]
    }
};
</script>
<style>
    #container {
        height: 350px;
    }
</style>

```

{% previewsample "page.domainurl/code-snippet/chart/series/smartlabel-cs1" %}

Note: To use the data label feature, inject the `DataLabel` into the `provide`.

### Positioning

Accumulation chart provides support for placing the data label either `inside` or `outside` the chart.

### APP.VUE

```

<template>
    <div id="app">
        <ejs-accumulationchart id="container">

```

```

        <e-accumulation-series-collection>
            <e-accumulation-series :dataSource='seriesData' xName='x'
yName='y' :dataLabel='datalabel'> </e-accumulation-series>
        </e-accumulation-series-collection>
    </ejs-accumulationchart>
</div>
</template>
<script>
import Vue from "vue";
import { AccumulationChartPlugin, PieSeries, AccumulationDataLabel } from
"@syncfusion/ej2-vue-charts";
Vue.use(AccumulationChartPlugin);
export default {
    data() {
        return {
            seriesData: [
                { x: 'Jan', y: 3, text: 'Jan: 3' }, { x: 'Feb', y: 3.5,
text: 'Feb: 3.5' },
                { x: 'Mar', y: 7, text: 'Mar: 7' }, { x: 'Apr', y: 13.5,
text: 'Apr: 13.5' },
                { x: 'May', y: 19, text: 'May: 19' }, { x: 'Jun', y: 23.5,
text: 'Jun: 23.5' },
                { x: 'Jul', y: 26, text: 'Jul: 26' }, { x: 'Aug', y: 25,
text: 'Aug: 25' },
                { x: 'Sep', y: 21, text: 'Sep: 21' }, { x: 'Oct', y: 15,
text: 'Oct: 15' }
            ],
            datalabel: { visible: true, position: 'Outside', name: 'text'
},
        };
    },
    provide: {
        accumulationchart: [PieSeries, AccumulationDataLabel]
    }
};
</script>
<style>
    #container {
        height: 350px;
    }
</style>

```

{% previewsample "page.domainurl/code-snippet/chart/series/smartlabel-cs2" %}

### DataLabel rotation

Using **angle** property, you can rotate the data label by its given angle.

### APP.VUE

```

<template>
    <div id="app">
        <ejs-accumulationchart id="container">
            <e-accumulation-series-collection>
                <e-accumulation-series :dataSource='seriesData' xName='x'
yName='y' :dataLabel='datalabel'> </e-accumulation-series>
            </e-accumulation-series-collection>
        </ejs-accumulationchart>
    </div>
</template>

```

```

        </ejs-accumulationchart>
    </div>
</template>
<script>
import Vue from "vue";
import { AccumulationChartPlugin, PieSeries, AccumulationDataLabel } from
"@syncfusion/ej2-vue-charts";
Vue.use(AccumulationChartPlugin);
export default {
    data() {
        return {
            seriesData: [
                { x: 'Jan', y: 3, text: 'Jan: 3' }, { x: 'Feb', y: 3.5,
text: 'Feb: 3.5' },
                { x: 'Mar', y: 7, text: 'Mar: 7' }, { x: 'Apr', y: 13.5,
text: 'Apr: 13.5' },
                { x: 'May', y: 19, text: 'May: 19' }, { x: 'Jun', y: 23.5,
text: 'Jun: 23.5' },
                { x: 'Jul', y: 26, text: 'Jul: 26' }, { x: 'Aug', y: 25,
text: 'Aug: 25' },
                { x: 'Sep', y: 21, text: 'Sep: 21' }, { x: 'Oct', y: 15,
text: 'Oct: 15' }
            ],
            datalabel: { visible: true, name: 'text', angle: 90,
enableRotation: true },
        };
    },
    provide: {
        accumulationchart: [PieSeries, AccumulationDataLabel]
    }
};
</script>
<style>
    #container {
        height: 350px;
    }
</style>

```

{% previewsample "page.domainurl/code-snippet/chart/series/smartlabel-cs3" %}

### Smart labels

Data labels will be arranged smartly without overlapping with each other. You can enable or disable this feature using the [enableSmartLabels](#) property.

### APP.VUE

```

<template>
    <div id="app">
        <ejs-accumulationchart id="container"
enableSmartLabels='enableSmartLabels'>
            <e-accumulation-series-collection>
                <e-accumulation-series :dataSource='seriesData' xName='x'
yName='y' :dataLabel='datalabel'> </e-accumulation-series>
            </e-accumulation-series-collection>
        </ejs-accumulationchart>
    </div>

```

```

</template>
<script>
import Vue from "vue";
import { AccumulationChartPlugin, PieSeries, AccumulationDataLabel } from
"@syncfusion/ej2-vue-charts";
Vue.use(AccumulationChartPlugin);
export default {
  data() {
    return {
      seriesData: [
        { x: 'Jan', y: 3, text: 'Jan: 3' }, { x: 'Feb', y: 3.5,
text: 'Feb: 3.5' },
        { x: 'Mar', y: 7, text: 'Mar: 7' }, { x: 'Apr', y: 13.5,
text: 'Apr: 13.5' },
        { x: 'May', y: 19, text: 'May: 19' }, { x: 'Jun', y: 23.5,
text: 'Jun: 23.5' },
        { x: 'Jul', y: 26, text: 'Jul: 26' }, { x: 'Aug', y: 25,
text: 'Aug: 25' },
        { x: 'Sep', y: 21, text: 'Sep: 21' }, { x: 'Oct', y: 15,
text: 'Oct: 15' } ],
      datalabel: { visible: true, name: 'text', position:
'Outside', name: 'text' },
      enableSmartLabels: true
    };
  },
  provide: {
    accumulationchart: [PieSeries, AccumulationDataLabel]
  }
};
</script>
<style>
#container {
  height: 350px;
}
</style>

```

{% previewsample "page.domainurl/code-snippet/chart/series/smartlabel-cs4" %}

### Datalabel template

Label content can be formatted by using the template option. Inside the template, you can add the placeholder text `${point.x}` and `${point.y}` to display corresponding data points x & y value. Using [template](#) property, you can set data label

template in chart.

### APP.VUE

```

<template>
  <div id="app">
    <ejs-accumulationchart id="container"
enableSmartLabels='enableSmartLabels'>
      <e-accumulation-series-collection>
        <e-accumulation-series :dataSource='seriesData' xName='x'
yName='y' :dataLabel='datalabel'> </e-accumulation-series>
      </e-accumulation-series-collection>
    </ejs-accumulationchart>
  </div>
</template>

```

```

    </div>
  </template>
  <script>
import Vue from "vue";
import { AccumulationChartPlugin, PieSeries, AccumulationDataLabel } from
"@syncfusion/ej2-vue-charts";
Vue.use(AccumulationChartPlugin);
export default {
  data() {
    return {
      seriesData: [
        { x: 'Jan', y: 3, text: 'Jan: 3' }, { x: 'Feb', y: 3.5,
text: 'Feb: 3.5' },
        { x: 'Mar', y: 7, text: 'Mar: 7' }, { x: 'Apr', y: 13.5,
text: 'Apr: 13.5' },
        { x: 'May', y: 19, text: 'May: 19' }, { x: 'Jun', y: 23.5,
text: 'Jun: 23.5' },
        { x: 'Jul', y: 26, text: 'Jul: 26' }, { x: 'Aug', y: 25,
text: 'Aug: 25' },
        { x: 'Sep', y: 21, text: 'Sep: 21' }, { x: 'Oct', y: 15,
text: 'Oct: 15' } ],
      datalabel: { visible: true, name: 'text', position:
'Inside',
        template: "<div id='templateWrap' style='background-
color:#bd18f9;border-radius: 3px; float: right;padding: 2px;line-height:
20px;text-align: center;'>" + "<img src='sun_annotation.png' />" + "<div
style='color:white; font-family:Roboto; font-style: medium; fontp-
size:14px;float: right;padding: 2px;line-height: 20px;text-align:
center;padding-right:6px'><span>${point.y}</span></div></div>" },
      enableSmartLabels: true
    };
  },
  provide: {
    accumulationchart: [PieSeries, AccumulationDataLabel]
  }
};
</script>
<style>
  #container {
    height: 350px;
  }
</style>

```

{% previewsample "page.domainurl/code-snippet/chart/series/smartlabel-cs5" %}

### Connector Line

Connector line will be visible when the data label is placed outside the chart. The connector line can be customized using the type, color, width, length and dashArray properties

### APP.VUE

```

<template>
  <div id="app">
    <ejs-accumulationchart id="container">
      <e-accumulation-series-collection>

```



```

        <e-accumulation-series :dataSource='seriesData' xName='x'
yName='y' :dataLabel='datalabel'> </e-accumulation-series>
    </e-accumulation-series-collection>
</ejs-accumulationchart>
</div>
</template>
<script>
import Vue from "vue";
import { AccumulationChartPlugin, PieSeries, AccumulationDataLabel } from
"@syncfusion/ej2-vue-charts";
Vue.use(AccumulationChartPlugin);
export default {
    data() {
        return {
            seriesData: [
                { x: 'Jan', y: 3, text: 'Jan: 3' }, { x: 'Feb', y: 3.5,
text: 'Feb: 3.5' },
                { x: 'Mar', y: 7, text: 'Mar: 7' }, { x: 'Apr', y: 13.5,
text: 'Apr: 13.5' },
                { x: 'May', y: 19, text: 'May: 19' }, { x: 'Jun', y: 23.5,
text: 'Jun: 23.5' },
                { x: 'Jul', y: 26, text: 'Jul: 26' }, { x: 'Aug', y: 25,
text: 'Aug: 25' },
                { x: 'Sep', y: 21, text: 'Sep: 21' }, { x: 'Oct', y: 15,
text: 'Oct: 15' } ],
            datalabel: { visible: true, name: 'text', position:
'Outside',
connectorStyle:{
    //Length of the connector line in pixels
    length: '50px',
    //Width of the connector line in pixels
    width: 2,
    //dashArray of the connector line
    dashArray: '5,3',
    //Color of the connector line
    color: '#f4429e',
    //Specifies the type of the connector line either Line
or Curve
    type: 'Curve'
}
        };
    },
    provide: {
        accumulationchart: [PieSeries, AccumulationDataLabel]
    }
};
</script>
<style>
    #container {
        height: 350px;
    }
</style>

```

{% previewsample "page.domainurl/code-snippet/chart/series/smartlabel-cs6" %}

### Text Mapping

The fill color and the text in the data source can be mapped to the chart using `pointColorMapping` in series and `name` in `datalabel` respectively.

### APP.VUE

```
<template>
  <div id="app">
    <ejs-accumulationchart id="container">
      <e-accumulation-series-collection>
        <e-accumulation-series :dataSource='seriesData' xName='x'
yName='y' :dataLabel='datalabel'> </e-accumulation-series>
      </e-accumulation-series-collection>
    </ejs-accumulationchart>
  </div>
</template>
<script>
import Vue from "vue";
import { AccumulationChartPlugin, PieSeries, AccumulationDataLabel } from
"@syncfusion/ej2-vue-charts";
Vue.use(AccumulationChartPlugin);
export default {
  data() {
    return {
      seriesData: [
        { x: 'Jan', y: 3, text: 'Jan: 3' }, { x: 'Feb', y: 3.5,
text: 'Feb: 3.5' },
        { x: 'Mar', y: 7, text: 'Mar: 7' }, { x: 'Apr', y: 13.5,
text: 'Apr: 13.5' },
        { x: 'May', y: 19, text: 'May: 19' }, { x: 'Jun', y: 23.5,
text: 'Jun: 23.5' },
        { x: 'Jul', y: 26, text: 'Jul: 26' }, { x: 'Aug', y: 25,
text: 'Aug: 25' },
        { x: 'Sep', y: 21, text: 'Sep: 21' }, { x: 'Oct', y: 15,
text: 'Oct: 15' } ],
      datalabel: { visible: true, name: 'text' }
    };
  },
  provide: {
    accumulationchart: [PieSeries, AccumulationDataLabel]
  }
};
</script>
<style>
  #container {
    height: 350px;
  }
</style>
```

{% previewsample "page.domainurl/code-snippet/chart/series/pie-cs7" %}

### Format

Data label for the accumulation chart can be formatted using `format` property. You can use the global formatting options, such as 'n', 'p', and 'c'.

### APP.VUE

```

<template>
  <div id="app">
    <ejs-accumulationchart id="container">
      <e-accumulation-series-collection>
        <e-accumulation-series :dataSource='seriesData' xName='x'
yName='y' :dataLabel='datalabel'> </e-accumulation-series>
      </e-accumulation-series-collection>
    </ejs-accumulationchart>
  </div>
</template>
<script>
import Vue from "vue";
import { AccumulationChartPlugin, PieSeries, AccumulationDataLabel } from
"@syncfusion/ej2-vue-charts";
Vue.use(AccumulationChartPlugin);
export default {
  data() {
    return {
      seriesData: [
        { x: 'Jan', y: 3, text: 'Jan: 3' }, { x: 'Feb', y: 3.5,
text: 'Feb: 3.5' },
        { x: 'Mar', y: 7, text: 'Mar: 7' }, { x: 'Apr', y: 13.5,
text: 'Apr: 13.5' },
        { x: 'May', y: 19, text: 'May: 19' }, { x: 'Jun', y: 23.5,
text: 'Jun: 23.5' },
        { x: 'Jul', y: 26, text: 'Jul: 26' }, { x: 'Aug', y: 25,
text: 'Aug: 25' },
        { x: 'Sep', y: 21, text: 'Sep: 21' }, { x: 'Oct', y: 15,
text: 'Oct: 15' } ],
      datalabel: { visible: true, format: 'n2' }
    };
  },
  provide: {
    accumulationchart: [PieSeries, AccumulationDataLabel]
  }
};
</script>
<style>
  #container {
    height: 350px;
  }
</style>

```

{% previewsample "page.domainurl/code-snippet/chart/series/pie-cs8" %}

Value	Format	Resultant Value	Description
1000	n1	1000.0	The number is rounded to 1 decimal place.
1000	n2	1000.00	The number is rounded to 2 decimal places.
1000	n3	1000.000	The number is rounded to 3 decimal place.
0.01	p1	1.0%	The number is converted to percentage with 1 decimal place.
0.01	p2	1.00%	The number is converted to percentage with 2 decimal place.

0.01	p3	1.000%	The number is converted to percentage with 3 decimal place.
1000	c1	\$1000.0	The currency symbol is appended to number and number is rounded to 1 decimal place.
1000	c2	\$1000.00	The currency symbol is appended to number and number is rounded to 2 decimal place.

### Customization

Individual text can be customized using the `textRender` event.

#### APP.VUE

```

<template>
  <div id="app">
    <ejs-accumulationchart id="container" :textRender='onTextRender'>
      <e-accumulation-series-collection>
        <e-accumulation-series :dataSource='seriesData' xName='x'
yName='y' :dataLabel='datalabel'> </e-accumulation-series>
      </e-accumulation-series-collection>
    </ejs-accumulationchart>
  </div>
</template>
<script>
import Vue from "vue";
import { AccumulationChartPlugin, PieSeries, AccumulationDataLabel } from
"@syncfusion/ej2-vue-charts";
Vue.use(AccumulationChartPlugin);
export default {
  data() {
    return {
      seriesData: [
        { x: 'Jan', y: 3, text: 'Jan: 3' }, { x: 'Feb', y: 3.5,
text: 'Feb: 3.5' },
        { x: 'Mar', y: 7, text: 'Mar: 7' }, { x: 'Apr', y: 13.5,
text: 'Apr: 13.5' },
        { x: 'May', y: 19, text: 'May: 19' }, { x: 'Jun', y: 23.5,
text: 'Jun: 23.5' },
        { x: 'Jul', y: 26, text: 'Jul: 26' }, { x: 'Aug', y: 25,
text: 'Aug: 25' },
        { x: 'Sep', y: 21, text: 'Sep: 21' }, { x: 'Oct', y: 15,
text: 'Oct: 15' } ],
      datalabel: { visible: true, name: 'text', position:
'Outside' }
    };
  },
  provide: {
    accumulationchart: [PieSeries, AccumulationDataLabel]
  },
  methods: {
    onTextRender: function (args) {
      if (args.text.indexOf('Mar') > -1) {
        args.color = 'red';
        args.border.width = 1;
      }
    }
  }
}

```

```

    }
  }
};
</script>
<style>
  #container {
    height: 350px;
  }
</style>

```

{% previewsample "page.domainurl/code-snippet/chart/series/smartlabel-cs7" %}

### Text wrap

When the data label text exceeds the container, the text can be wrapped by using [textWrap](#) property. End user can also wrap the data label text based on [maxWidth](#) property.

### APP.VUE

```

<template>
  <div id="app">
    <ejs-accumulationchart id="container">
      <e-accumulation-series-collection>
        <e-accumulation-series :dataSource='seriesData' xName='x'
yName='y' startAngle='270' endAngle='90' innerRadius='40%'
tooltipMappingName='tooltipMappingName' :dataLabel='datalabel'>
        </e-accumulation-series>
      </e-accumulation-series-collection>
    </ejs-accumulationchart>
  </div>
</template>
<script>
import Vue from "vue";
import { AccumulationChartPlugin, PieSeries, AccumulationDataLabel } from
"@syncfusion/ej2-vue-charts";
Vue.use(AccumulationChartPlugin);
export default {
  data() {
    return {
      seriesData: [
        { x: 'Chrome', y: 100, text: 'Chrome (100M)<br>40%',
tooltipMappingName: '40%' },
        { x: 'UC Browser', y: 40, text: 'UC Browser
(40M)<br>16%', tooltipMappingName: '16%' },
        { x: 'Opera', y: 30, text: 'Opera (30M)<br>12%',
tooltipMappingName: '12%' },
        { x: 'Safari', y: 30, text: 'Safari (30M)<br>12%',
tooltipMappingName: '12%' },
        { x: 'Firefox', y: 25, text: 'Firefox (25M)<br>10%',
tooltipMappingName: '10%' },
        { x: 'Others', y: 25, text: 'Others (25M)<br>10%',
tooltipMappingName: '10%' }],
      datalabel: { visible: true, position: 'Inside'
,maxWidth:100, textWrap:'Wrap',
      name: 'text', enableRotation:true }
    };
  },

```

```

provide: {
  accumulationchart: [PieSeries, AccumulationDataLabel]
},
methods: {
}
};
</script>
<style>
#container {
  height: 350px;
}
</style>

```

{% previewsample "page.domainurl/code-snippet/chart/series/smartlabel-cs8" %}

### Show percentages in data labels of pie chart

You can show the percentages in data labels of pie chart using `textRender` event and `template` option.

#### Using `textRender` event

You can customize the data label of pie chart using `textRender` event as follows to show percentage.

#### APP.VUE

```

<template>
  <div id="app">
    <ejs-accumulationchart id="container"
      enableSmartLabels='enableSmartLabels' :textRender='onTextRender'>
      <e-accumulation-series-collection>
        <e-accumulation-series :dataSource='seriesData' xName='x'
          yName='y' :dataLabel='datalabel'> </e-accumulation-series>
      </e-accumulation-series-collection>
    </ejs-accumulationchart>
  </div>
</template>
<script>
import Vue from "vue";
import { AccumulationChartPlugin, PieSeries, AccumulationDataLabel } from
"@syncfusion/ej2-vue-charts";
Vue.use(AccumulationChartPlugin);
export default {
  data() {
    return {
      seriesData: [
        { x: 'Jan', y: 3, text: 'Jan: 3' }, { x: 'Feb', y: 3.5,
text: 'Feb: 3.5' },
        { x: 'Mar', y: 7, text: 'Mar: 7' }, { x: 'Apr', y: 13.5,
text: 'Apr: 13.5' },
        { x: 'May', y: 19, text: 'May: 19' }, { x: 'Jun', y: 23.5,
text: 'Jun: 23.5' },
        { x: 'Jul', y: 26, text: 'Jul: 26' }, { x: 'Aug', y: 25,
text: 'Aug: 25' },
        { x: 'Sep', y: 21, text: 'Sep: 21' }, { x: 'Oct', y: 15,
text: 'Oct: 15' }
      ],
      datalabel: { visible: true },
      enableSmartLabels: true
    }
  }
}

```

```

    };
    },
    provide: {
      accumulationchart: [PieSeries, AccumulationDataLabel]
    },
    methods: {
      onTextRender: function (args) {
        args.text = args.point.percentage + "%";
      }
    }
  };
</script>
<style>
  #container {
    height: 350px;
  }
</style>

```

{% previewsample "page.domainurl/code-snippet/chart/series/smartlabel-cs9" %}

### Using template

You can display the percentage values in data label of pie chart using **template** option.

### APP.VUE

```

<template>
  <div id="app">
    <ejs-accumulationchart id="container"
      enableSmartLabels='enableSmartLabels'>
      <e-accumulation-series-collection>
        <e-accumulation-series :dataSource='seriesData' xName='x'
          yName='y' :dataLabel='datalabel'> </e-accumulation-series>
      </e-accumulation-series-collection>
    </div>
  </template>
<script>
import Vue from "vue";
import { AccumulationChartPlugin, PieSeries, AccumulationDataLabel } from
"@syncfusion/ej2-vue-charts";
Vue.use(AccumulationChartPlugin);
export default {
  data() {
    return {
      seriesData: [
        { x: 'Jan', y: 3, text: 'Jan: 3' }, { x: 'Feb', y: 3.5,
text: 'Feb: 3.5' },
        { x: 'Mar', y: 7, text: 'Mar: 7' }, { x: 'Apr', y: 13.5,
text: 'Apr: 13.5' },
        { x: 'May', y: 19, text: 'May: 19' }, { x: 'Jun', y: 23.5,
text: 'Jun: 23.5' },
        { x: 'Jul', y: 26, text: 'Jul: 26' }, { x: 'Aug', y: 25,
text: 'Aug: 25' },
        { x: 'Sep', y: 21, text: 'Sep: 21' }, { x: 'Oct', y: 15,
text: 'Oct: 15' }
      ],
    }
  }
}

```

```

        datalabel: { visible: true, template: "<div
id='dataLabelTemplate'>${point.percentage}%</div>" },
        enableSmartLabels: true
    };
},
provide: {
    accumulationchart: [PieSeries, AccumulationDataLabel]
}
};
</script>
<style>
    #container {
        height: 350px;
    }
</style>

```

{% previewsample "page.domainurl/code-snippet/chart/series/smartlabel-cs10" %}

### Grouping in Vue Accumulation chart component

You can club/group few points of the series based on [groupTo](#) property. For example, if the club value is 11, then the points with value less than 11 is grouped together and will be showed as a single point with label **others**. The property also takes value in percentage (percentage of total data points value).

#### APP.VUE

```

<template>
    <div id="app">
        <ejs-accumulationchart id="container">
            <e-accumulation-series-collection>
                <e-accumulation-series :dataSource='seriesData' xName='x'
yName='y' groupTo='11' :dataLabel='datalabel'> </e-accumulation-series>
            </e-accumulation-series-collection>
        </ejs-accumulationchart>
    </div>
</template>
<script>
import Vue from "vue";
import { AccumulationChartPlugin, PieSeries, AccumulationDataLabel } from
"@syncfusion/ej2-vue-charts";
Vue.use(AccumulationChartPlugin);
export default {
    data() {
        return {
            seriesData: [
                { x: 'Jan', y: 3, text: 'Jan: 3' }, { x: 'Feb', y: 3.5,
text: 'Feb: 3.5' },
                { x: 'Mar', y: 7, text: 'Mar: 7' }, { x: 'Apr', y: 13.5,
text: 'Apr: 13.5' },
                { x: 'May', y: 19, text: 'May: 19' }, { x: 'Jun', y: 23.5,
text: 'Jun: 23.5' },
                { x: 'Jul', y: 26, text: 'Jul: 26' }, { x: 'Aug', y: 25,
text: 'Aug: 25' },
                { x: 'Sep', y: 21, text: 'Sep: 21' }, { x: 'Oct', y: 15,
text: 'Oct: 15' },
                { x: 'Nov', y: 9, text: 'Nov: 9' }, { x: 'Dec', y: 3.5,
text: 'Dec: 3.5' }
            ]
        }
    }
}

```



```

    ],
    datalabel: { visible: true, position: 'Outside', name: 'text'
  }
  };
  },
  provide: {
    accumulationchart: [PieSeries, AccumulationDataLabel]
  }
};
</script>
<style>
#container {
  height: 350px;
}
</style>

```

{% previewsample "page.domainurl/code-snippet/chart/series/clubpoint-cs1" %}

### Broken slice

You can visualize all points available in club/group points by clicking on the grouped point. For example, if 5 points are grouped together it will be showed as a single slice with label **others**. If we click on **others** slice it will explode and broke into 5 seperate slices.

### APP.VUE

```

<template>
  <div id="app">
    <ejs-accumulationchart id="container">
      <e-accumulation-series-collection>
        <e-accumulation-series :dataSource='seriesData' xName='x'
yName='y' groupTo='11' explode=true :dataLabel='datalabel'> </e-
accumulation-series>
      </e-accumulation-series-collection>
    </ejs-accumulationchart>
  </div>
</template>
<script>
import Vue from "vue";
import { AccumulationChartPlugin, PieSeries, AccumulationDataLabel } from
"@syncfusion/ej2-vue-charts";
Vue.use(AccumulationChartPlugin);
export default {
  data() {
    return {
      seriesData: [
        { x: 'Jan', y: 3, text: 'Jan: 3' }, { x: 'Feb', y: 3.5,
text: 'Feb: 3.5' },
        { x: 'Mar', y: 7, text: 'Mar: 7' }, { x: 'Apr', y: 13.5,
text: 'Apr: 13.5' },
        { x: 'May', y: 19, text: 'May: 19' }, { x: 'Jun', y: 23.5,
text: 'Jun: 23.5' },
        { x: 'Jul', y: 26, text: 'Jul: 26' }, { x: 'Aug', y: 25,
text: 'Aug: 25' },
        { x: 'Sep', y: 21, text: 'Sep: 21' }, { x: 'Oct', y: 15,
text: 'Oct: 15' },

```

```

        { x: 'Nov', y: 9, text: 'Nov: 9' }, { x: 'Dec', y: 3.5,
text: 'Dec: 3.5' }
      ],
      datalabel: { visible: true, position: 'Outside', name: 'text'
    }
  };
},
provide: {
  accumulationchart: [PieSeries, AccumulationDataLabel]
}
};
</script>
<style>
  #container {
    height: 350px;
  }
</style>

```

{% previewsample "page.domainurl/code-snippet/chart/series/clubpoint-cs2" %}

### GroupMode

Slice can also be grouped based on number of points by specifying the **groupMode** to Point. For example, if the group to value is 11, accumulation chart will show 1st 11 points and will group remaining entries in the collection as a single point.

### APP.VUE

```

<template>
  <div id="app">
    <ejs-accumulationchart id="container">
      <e-accumulation-series-collection>
        <e-accumulation-series :dataSource='seriesData' xName='x'
yName='y' groupTo='4' groupMode='groupMode' :dataLabel='datalabel'> </e-
accumulation-series>
      </e-accumulation-series-collection>
    </ejs-accumulationchart>
  </div>
</template>
<script>
import Vue from "vue";
import { AccumulationChartPlugin, PieSeries, AccumulationDataLabel } from
"@syncfusion/ej2-vue-charts";
Vue.use(AccumulationChartPlugin);
export default {
  data() {
    return {
      seriesData: [
        { x: 'Jan', y: 3, text: 'Jan: 3' }, { x: 'Feb', y: 3.5,
text: 'Feb: 3.5' },
        { x: 'Mar', y: 7, text: 'Mar: 7' }, { x: 'Apr', y: 13.5,
text: 'Apr: 13.5' },
        { x: 'May', y: 19, text: 'May: 19' }, { x: 'Jun', y: 23.5,
text: 'Jun: 23.5' },
        { x: 'Jul', y: 26, text: 'Jul: 26' }, { x: 'Aug', y: 25,
text: 'Aug: 25' },

```

```

      { x: 'Sep', y: 21, text: 'Sep: 21' }, { x: 'Oct', y: 15,
text: 'Oct: 15' },
      { x: 'Nov', y: 9, text: 'Nov: 9' }, { x: 'Dec', y: 3.5,
text: 'Dec: 3.5' }
    ],
    datalabel: { visible: true, position: 'Outside', name: 'text'
  },
    groupMode: 'Point'
  };
},
provide: {
  accumulationchart: [PieSeries, AccumulationDataLabel]
}
};
</script>
<style>
  #container {
    height: 350px;
  }
</style>

```

{% previewsample "page.domainurl/code-snippet/chart/series/clubpoint-cs3" %}

### Customization

You can customize the grouped point and its data label using `pointRender` and `textRender` event.

### APP.VUE

```

<template>
  <div id="app">
    <ejs-accumulationchart id="container">
      <e-accumulation-series-collection :textRender='onTextRender'
:pointRender='onPointRender' :dataLabel='datalabel'>
        <e-accumulation-series :dataSource='seriesData' xName='x'
yName='y' groupTo='11' :dataLabel='datalabel'> </e-accumulation-series>
      </e-accumulation-series-collection>
    </div>
  </template>
<script>
import Vue from "vue";
import { AccumulationChartPlugin, PieSeries, AccumulationDataLabel } from
"@syncfusion/ej2-vue-charts";
Vue.use(AccumulationChartPlugin);
export default {
  data() {
    return {
      seriesData: [
        { x: 'Jan', y: 3, text: 'Jan: 3' }, { x: 'Feb', y: 3.5,
text: 'Feb: 3.5' },
        { x: 'Mar', y: 7, text: 'Mar: 7' }, { x: 'Apr', y: 13.5,
text: 'Apr: 13.5' },
        { x: 'May', y: 19, text: 'May: 19' }, { x: 'Jun', y: 23.5,
text: 'Jun: 23.5' },
        { x: 'Jul', y: 26, text: 'Jul: 26' }, { x: 'Aug', y: 25,
text: 'Aug: 25' },

```

```

        { x: 'Sep', y: 21, text: 'Sep: 21' }, { x: 'Oct', y: 15,
text: 'Oct: 15' },
        { x: 'Nov', y: 9, text: 'Nov: 9' }, { x: 'Dec', y: 3.5,
text: 'Dec: 3.5' }
    ],
    datalabel: { visible: true, position: 'Outside', name: 'text'
}
    };
},
provide: {
    accumulationchart: [PieSeries, AccumulationDataLabel]
},
methods: {
    onTextRender: function (args) {
        if (args.text.indexOf('Others') > -1) {
            args.text = 'Grouped Slices';
            args.color = 'red';
            args.border.width = 1;
        }
    },
    onPointRender: function (args) {
        if ((args.point.x as string).indexOf('Others') > -1) {
            args.fill = '#D3D3D3';
        }
    }
}
};
</script>
<style>
    #container {
        height: 350px;
    }
</style>

```

{% previewsample "page.domainurl/code-snippet/chart/series/clubpoint-cs4" %}

### Empty points in Vue Accumulation chart component

The data points those uses the **null** or **undefined** as value are considered as empty points. The empty data points are ignored and not plotted in the chart. You can customize those points, using the [emptyPointSettings](#) property in

series. The default mode of the empty point is **Gap**. Other supported modes are **Average** and **Zero**.

### APP.VUE

```

<template>
    <div id="app">
        <ejs-accumulationchart id="container">
            <e-accumulation-series-collection>
                <e-accumulation-series :dataSource='seriesData' xName='x'
yName='y' :emptyPointSettings='emptyPointSettings'> </e-accumulation-series>
            </e-accumulation-series-collection>
        </ejs-accumulationchart>
    </div>
</template>
<script>

```

```

import Vue from "vue";
import { AccumulationChartPlugin, PieSeries, AccumulationDataLabel } from
"@syncfusion/ej2-vue-charts";
Vue.use(AccumulationChartPlugin);
export default {
  data() {
    return {
      seriesData: [
        { x: 'Jan', y: 3, text: 'Jan: 3' }, { x: 'Feb', y: 3.5,
text: 'Feb: 3.5' },
        { x: 'Mar', y: null, text: 'Mar: 7' }, { x: 'Apr', y: 13.5,
text: 'Apr: 13.5' },
        { x: 'May', y: 19, text: 'May: 19' }, { x: 'Jun', y: 23.5,
text: 'Jun: 23.5' },
        { x: 'Jul', y: 26, text: 'Jul: 26' }, { x: 'Aug', y:
undefined, text: 'Aug: 25' },
        { x: 'Sep', y: 21, text: 'Sep: 21' }, { x: 'Oct', y: 15,
text: 'Oct: 15' } ],
      emptyPointSettings: { mode: 'Zero', fill: 'pink' },
    };
  },
  provide: {
    accumulationchart: [PieSeries, AccumulationDataLabel]
  }
};
</script>
<style>
  #container {
    height: 350px;
  }
</style>

```

{% previewsample "page.domainurl/code-snippet/chart/series/radius-cs3" %}

### Customization

Specific color for an empty point can be set by using the `fill` property in `emptyPointSettings` and the border for an empty point can be set by using the `border` property.

### APP.VUE

```

<template>
  <div id="app">
    <ejs-accumulationchart id="container">
      <e-accumulation-series-collection>
        <e-accumulation-series :dataSource='seriesData' xName='x'
yName='y' :emptyPointSettings='emptyPointSettings'> </e-accumulation-series>
      </e-accumulation-series-collection>
    </ejs-accumulationchart>
  </div>
</template>
<script>
import Vue from "vue";
import { AccumulationChartPlugin, PieSeries } from "@syncfusion/ej2-vue-
charts";
Vue.use(AccumulationChartPlugin);

```

```

export default {
  data() {
    return {
      seriesData: [
        { x: 'Jan', y: 3, text: 'Jan: 3' }, { x: 'Feb', y: 3.5,
text: 'Feb: 3.5' },
        { x: 'Mar', y: null, text: 'Mar: 7' }, { x: 'Apr', y: 13.5,
text: 'Apr: 13.5' },
        { x: 'May', y: 19, text: 'May: 19' }, { x: 'Jun', y: 23.5,
text: 'Jun: 23.5' },
        { x: 'Jul', y: 26, text: 'Jul: 26' }, { x: 'Aug', y:
undefined, text: 'Aug: 25' },
        { x: 'Sep', y: 21, text: 'Sep: 21' }, { x: 'Oct', y: 15,
text: 'Oct: 15' }],
      emptyPointSettings: { mode: 'Average', fill: 'red' }
    };
  },
  provide: {
    accumulationchart: [PieSeries]
  }
};
</script>
<style>
  #container {
    height: 350px;
  }
</style>

```

{% previewsample "page.domainurl/code-snippet/chart/series/radius-cs4" %}

### Annotation in Vue Accumulation chart component

The annotations are used to mark the specific area of interest in the chart area with texts, shapes or images.

<!-- markdownlint-disable MD033 -->

By using the <code>content</code> option of annotation property, you can specify the Id of the element that needs

to be displayed in the chart area.

### APP.VUE

```

<template>
  <div id="app">
    <ejs-accumulationchart id="container" :annotations='annotations'>
      <e-accumulation-series-collection>
        <e-accumulation-series :dataSource='seriesData' xName='x'
yName='y'> </e-accumulation-series>
      </e-accumulation-series-collection>
    </ejs-accumulationchart>
  </div>
</template>
<script>
import Vue from "vue";
import { AccumulationChartPlugin, PieSeries, AccumulationAnnotation } from
"@syncfusion/ej2-vue-charts";

```

```

Vue.use(AccumulationChartPlugin);
export default {
  data() {
    return {
      seriesData: [{ x: 'Jan', y: 3 }, { x: 'Feb', y: 3.5 }, { x: 'Mar', y:
7 }, { x: 'Apr', y: 13.5 }, { x: 'May', y: 19 }, { x: 'Jun', y: 23.5 }, { x:
'Jul', y: 26 }, { x: 'Aug', y: 25 },
        { x: 'Sep', y: 21 }, { x: 'Oct', y: 15 }],
      annotations:[{
        content: '<div style="border: 1px solid black;background-
color:#f5f5f5; padding: 5px 5px 5px 5px">13.5</div>',
        region: 'Series',
        coordinateUnits: 'Point',
        x: 'Jan',
        y: 3
      }]
    };
  },
  provide: {
    accumulationchart: [AccumulationAnnotation, PieSeries]
  }
};
</script>
<style>
  #container {
    height: 350px;
  }
</style>

```

{% previewsample "page.domainurl/code-snippet/chart/series/pie-cs1" %}

Note: To use annotation feature in accumulation chart, we need to inject **AccumulationAnnotation** using **provide**.

### Region

The annotation can be placed with respect to either **Series** or **Chart**.

### APP.VUE

```

<template>
  <div id="app">
    <ejs-accumulationchart id="container" :annotations='annotations'>
      <e-accumulation-series-collection>
        <e-accumulation-series :dataSource='seriesData' xName='x'
yName='y'> </e-accumulation-series>
      </e-accumulation-series-collection>
    </ejs-accumulationchart>
  </div>
</template>
<script>
import Vue from "vue";
import { AccumulationChartPlugin, PieSeries, AccumulationAnnotation } from
"@syncfusion/ej2-vue-charts";
Vue.use(AccumulationChartPlugin);
export default {
  data() {

```

```

    return {
      seriesData: [{ x: 'Jan', y: 3 }, { x: 'Feb', y: 3.5 }, { x: 'Mar', y:
7 }, { x: 'Apr', y: 13.5 }, { x: 'May', y: 19 }, { x: 'Jun', y: 23.5 }, { x:
'Jul', y: 26 }, { x: 'Aug', y: 25 },
        { x: 'Sep', y: 21 }, { x: 'Oct', y: 15 }],
      annotations:[{
        content:'<div style="border: 1px solid black;background-
color:#f5f5f5; padding: 5px 5px 5px 5px">13.5</div>',
        region: 'Chart',
        coordinateUnits: 'Pixel',
        x: 250,
        y: 150
      }]
    };
  },
  provide: {
    accumulationchart: [AccumulationAnnotation, PieSeries]
  }
};
</script>
<style>
  #container {
    height: 350px;
  }
</style>

```

{% previewsample "page.domainurl/code-snippet/chart/series/pie-cs2" %}

### Co-ordinate Units

Specifies the coordinate units of an annotation either in **Pixel** or **Point**.

### APP.VUE

```

<template>
  <div id="app">
    <ejs-accumulationchart id="container" :annotations='annotations'>
      <e-accumulation-series-collection>
        <e-accumulation-series :dataSource='seriesData' xName='x'
yName='y'> </e-accumulation-series>
      </e-accumulation-series-collection>
    </ejs-accumulationchart>
  </div>
</template>
<script>
import Vue from "vue";
import { AccumulationChartPlugin, PieSeries, AccumulationAnnotation } from
"@syncfusion/ej2-vue-charts";
Vue.use(AccumulationChartPlugin);
export default {
  data() {
    return {
      seriesData: [{ x: 'Jan', y: 3 }, { x: 'Feb', y: 3.5 }, { x: 'Mar', y:
7 }, { x: 'Apr', y: 13.5 }, { x: 'May', y: 19 }, { x: 'Jun', y: 23.5 }, { x:
'Jul', y: 26 }, { x: 'Aug', y: 25 },
        { x: 'Sep', y: 21 }, { x: 'Oct', y: 15 }],
      annotations:[{

```



```

        content: '<div style="border: 1px solid black;background-
color:#f5f5f5; padding: 5px 5px 5px 5px">Jan : 13.5</div>',
        region: 'Series',
        coordinateUnits: 'Point',
        x: 'Jan',
        y: 3
    }
  ];
},
provide: {
  accumulationchart: [AccumulationAnnotation, PieSeries]
}
};
</script>
<style>
  #container {
    height: 350px;
  }
</style>

```

{% previewsample "page.domainurl/code-snippet/chart/series/pie-cs3" %}

### Alignment

The annotations provides the `verticalAlignment` or `horizontalAlignment` properties. The `verticalAlignment` property can be customized via `Top`, `Bottom` or `Middle` and the `horizontalAlignment` property can be customized via `Near`, `Far` or `Center`.

### APP.VUE

```

<template>
  <div id="app">
    <ejs-accumulationchart id="container" :annotations='annotations'>
      <e-accumulation-series-collection>
        <e-accumulation-series :dataSource='seriesData' xName='x'
yName='y'> </e-accumulation-series>
      </e-accumulation-series-collection>
    </ejs-accumulationchart>
  </div>
</template>
<script>
import Vue from "vue";
import { AccumulationChartPlugin, PieSeries, AccumulationAnnotation } from
"@syncfusion/ej2-vue-charts";
Vue.use(AccumulationChartPlugin);
export default {
  data() {
    return {
      seriesData: [{ x: 'Jan', y: 3 }, { x: 'Feb', y: 3.5 }, { x: 'Mar', y:
7 }, { x: 'Apr', y: 13.5 }, { x: 'May', y: 19 }, { x: 'Jun', y: 23.5 }, { x:
'Jul', y: 26 }, { x: 'Aug', y: 25 },
        { x: 'Sep', y: 21 }, { x: 'Oct', y: 15 }],
      annotations:[{
        content: '<div style="border: 1px solid black;background-
color:#f5f5f5; padding: 5px 5px 5px 5px">Jan : 13.5</div>',
        region: 'Series',
        coordinateUnits: 'Point',

```

```

        x: 'Jan',
        y: 3,
        verticalAlignment: 'Top',
        horizontalAlignment: 'Near'
    }]
    };
},
provide: {
    accumulationchart: [AccumulationAnnotation, PieSeries]
}
};
</script>
<style>
#container {
    height: 350px;
}
</style>

```

{% previewsample "page.domainurl/code-snippet/chart/series/pie-cs4" %}

### Tooltip in Vue Accumulation chart component

Tooltip for the accumulation chart can be enabled by using the [enable](#) property.

#### APP.VUE

```

<template>
  <div id="app">
    <ejs-accumulationchart id="container" :tooltip='tooltip'>
      <e-accumulation-series-collection>
        <e-accumulation-series :dataSource='seriesData' xName='x'
yName='y' radius='70%'> </e-accumulation-series>
      </e-accumulation-series-collection>
    </ejs-accumulationchart>
  </div>
</template>
<script>
import Vue from "vue";
import { AccumulationChartPlugin, PieSeries, AccumulationTooltip } from
"@syncfusion/ej2-vue-charts";
Vue.use(AccumulationChartPlugin);
export default {
  data() {
    return {
      seriesData: [
        { x: 'Jan', y: 3 }, { x: 'Feb', y: 3.5 },
        { x: 'Mar', y: 7 }, { x: 'Apr', y: 13.5 },
        { x: 'May', y: 19 }, { x: 'Jun', y: 23.5 },
        { x: 'Jul', y: 26 }, { x: 'Aug', y: 25 },
        { x: 'Sep', y: 21 }, { x: 'Oct', y: 15 }
      ],
      tooltip:{enable: true}
    };
  },
  provide: {
    accumulationchart: [PieSeries, AccumulationTooltip]
  }
}

```

```
};
</script>
<style>
  #container {
    height: 350px;
  }
</style>
```

{% previewsample "page.domainurl/code-snippet/chart/series/pie-cs27" %}

Note: To use tooltip feature, inject the `AccumulationTooltip` into the `provide`.

### Header

We can specify header for the tooltip using `header` property.

### APP.VUE

```
<template>
  <div id="app">
    <ejs-accumulationchart id="container" :tooltip='tooltip'>
      <e-accumulation-series-collection>
        <e-accumulation-series :dataSource='seriesData' xName='x'
yName='y' radius='70%'> </e-accumulation-series>
      </e-accumulation-series-collection>
    </ejs-accumulationchart>
  </div>
</template>
<script>
import Vue from "vue";
import { AccumulationChartPlugin, PieSeries, AccumulationTooltip } from
"@syncfusion/ej2-vue-charts";
Vue.use(AccumulationChartPlugin);
export default {
  data() {
    return {
      seriesData: [
        { x: 'Jan', y: 3 }, { x: 'Feb', y: 3.5 },
        { x: 'Mar', y: 7 }, { x: 'Apr', y: 13.5 },
        { x: 'May', y: 19 }, { x: 'Jun', y: 23.5 },
        { x: 'Jul', y: 26 }, { x: 'Aug', y: 25 },
        { x: 'Sep', y: 21 }, { x: 'Oct', y: 15 }
      ],
      tooltip:{enable: true, header:"Pie Chart"}
    };
  },
  provide: {
    accumulationchart: [PieSeries, AccumulationTooltip]
  }
};
</script>
<style>
  #container {
    height: 350px;
  }
</style>
```

```
{% previewsample "page.domainurl/code-snippet/chart/series/pie-cs28" %}
```

### Format

By default, tooltip shows information of x and y value in points. In addition to that, you can show more information in tooltip. For example the format `${series.name} ${point.x}` shows series name and point x value.

### APP.VUE

```
<template>
  <div id="app">
    <ejs-accumulationchart id="container" :tooltip='tooltip'>
      <e-accumulation-series-collection>
        <e-accumulation-series :dataSource='seriesData' xName='x'
yName='y' radius='70%'> </e-accumulation-series>
      </e-accumulation-series-collection>
    </ejs-accumulationchart>
  </div>
</template>
<script>
import Vue from "vue";
import { AccumulationChartPlugin, PieSeries, AccumulationTooltip } from
"@syncfusion/ej2-vue-charts";
Vue.use(AccumulationChartPlugin);
export default {
  data() {
    return {
      seriesData: [
        { x: 'Jan', y: 3 }, { x: 'Feb', y: 3.5 },
        { x: 'Mar', y: 7 }, { x: 'Apr', y: 13.5 },
        { x: 'May', y: 19 }, { x: 'Jun', y: 23.5 },
        { x: 'Jul', y: 26 }, { x: 'Aug', y: 25 },
        { x: 'Sep', y: 21 }, { x: 'Oct', y: 15 }
      ],
      tooltip:{enable: true, format: '${point.x} : <b>${point.y}</b>'}
    };
  },
  provide: {
    accumulationchart: [PieSeries, AccumulationTooltip]
  }
};
</script>
<style>
#container {
  height: 350px;
}
</style>
```

```
{% previewsample "page.domainurl/code-snippet/chart/series/pie-cs29" %}
```

### Tooltip template

Any HTML element can be displayed in the tooltip by using the [template](#) property.

### APP.VUE

```
<template>
```

```

<div id="app">
  <ejs-accumulationchart id="container" :tooltip='tooltip'>
    <e-accumulation-series-collection>
      <e-accumulation-series :dataSource='seriesData' xName='x'
yName='y' radius='70%'> </e-accumulation-series>
    </e-accumulation-series-collection>
  </ejs-accumulationchart>
</div>
</template>
<script>
import Vue from "vue";
import { AccumulationChartPlugin, PieSeries, AccumulationTooltip } from
"@syncfusion/ej2-vue-charts";
Vue.use(AccumulationChartPlugin);
export default {
  data() {
    return {
      seriesData: [
        { x: 'Jan', y: 3 }, { x: 'Feb', y: 3.5 },
        { x: 'Mar', y: 7 }, { x: 'Apr', y: 13.5 },
        { x: 'May', y: 19 }, { x: 'Jun', y: 23.5 },
        { x: 'Jul', y: 26 }, { x: 'Aug', y: 25 },
        { x: 'Sep', y: 21 }, { x: 'Oct', y: 15 }
      ],
      tooltip:{
        enable: true,
        template: "<div id='templateWrap' style='background-
color:#bd18f9;border-radius: 3px; float: right;padding: 2px;line-height:
20px;text-align: center;'>" +
          "<img src='sun_annotation.png' />" +
          "<div style='color:white; font-family:Roboto; font-style: medium;
fontp-size:14px;float: right;padding: 2px;line-height: 20px;text-align:
center;padding-right:6px'><span>${y}</span></div></div>"
        }
      };
    },
    provide: {
      accumulationchart: [PieSeries, AccumulationTooltip]
    }
  };
</script>
<style>
#container {
  height: 350px;
}
</style>

```

{% previewsample "page.domainurl/code-snippet/chart/series/pie-cs30" %}

### Fixed tooltip

By default, tooltip track the mouse movement, but you can set a fixed position for the tooltip by using the [location](#) property.

### APP.VUE

```
<template>
```

```

<div id="app">
  <ejs-accumulationchart id="container" :tooltip='tooltip'>
    <e-accumulation-series-collection>
      <e-accumulation-series :dataSource='seriesData' xName='x'
yName='y' radius='70%'> </e-accumulation-series>
    </e-accumulation-series-collection>
  </ejs-accumulationchart>
</div>
</template>
<script>
import Vue from "vue";
import { AccumulationChartPlugin, PieSeries, AccumulationTooltip } from
"@syncfusion/ej2-vue-charts";
Vue.use(AccumulationChartPlugin);
export default {
  data() {
    return {
      seriesData: [
        { x: 'Jan', y: 3 }, { x: 'Feb', y: 3.5 },
        { x: 'Mar', y: 7 }, { x: 'Apr', y: 13.5 },
        { x: 'May', y: 19 }, { x: 'Jun', y: 23.5 },
        { x: 'Jul', y: 26 }, { x: 'Aug', y: 25 },
        { x: 'Sep', y: 21 }, { x: 'Oct', y: 15 }
      ],
      tooltip:{
        enable: true,
        location: { x: 200, y: 20 }
      }
    };
  },
  provide: {
    accumulationchart: [PieSeries, AccumulationTooltip]
  }
};
</script>
<style>
#container {
  height: 350px;
}
</style>

```

{% previewsample "page.domainurl/code-snippet/chart/series/pie-cs37" %}

### Customization

The [fill](#) and [border](#) properties are used to customize the background color and border of the tooltip respectively. The [textStyle](#) property in the tooltip is used to customize the font of the tooltip text. The [highlightColor](#) property can be used to change the color of the data point when hovering.

### APP.VUE

```

<template>
  <div id="app">
    <ejs-accumulationchart id="container" :tooltip='tooltip'
:highlightColor='highlightColor'>
      <e-accumulation-series-collection>

```

```

        <e-accumulation-series :dataSource='seriesData' xName='x'
yName='y' radius='70%'> </e-accumulation-series>
    </e-accumulation-series-collection>
</ejs-accumulationchart>
</div>
</template>
<script>
import Vue from "vue";
import { AccumulationChartPlugin, PieSeries, AccumulationTooltip } from
"@syncfusion/ej2-vue-charts";
Vue.use(AccumulationChartPlugin);
export default {
    data() {
        return {
            seriesData: [
                { x: 'Jan', y: 3 }, { x: 'Feb', y: 3.5 },
                { x: 'Mar', y: 7 }, { x: 'Apr', y: 13.5 },
                { x: 'May', y: 19 }, { x: 'Jun', y: 23.5 },
                { x: 'Jul', y: 26 }, { x: 'Aug', y: 25 },
                { x: 'Sep', y: 21 }, { x: 'Oct', y: 15 }
            ],
            tooltip: {
                enable: true,
                format: '${series.name} ${point.x} : ${point.y}',
                //fill for tooltip
                fill: '#7bb4eb',
                //border for tooltip
                border: {
                    width: 2,
                    color: 'grey'
                }
            },
            highlightColor: 'red',
        };
    },
    provide: {
        accumulationchart: [PieSeries, AccumulationTooltip]
    }
};
</script>
<style>
    #container {
        height: 350px;
    }
</style>

```

{% previewsample "page.domainurl/code-snippet/chart/series/pie-cs31" %}

To customize individual tooltip

Using [tooltipRender](#) event, you can customize a tooltip for particular point.

#### APP.VUE

```

<template>
    <div id="app">

```

```

        <ejs-accumulationchart id="container" :tooltip='tooltip'
:tooltipRender='tooltipRender'>
            <e-accumulation-series-collection>
                <e-accumulation-series :dataSource='seriesData' xName='x'
yName='y' radius='70%'> </e-accumulation-series>
            </e-accumulation-series-collection>
        </ejs-accumulationchart>
    </div>
</template>
<script>
import Vue from "vue";
import { AccumulationChartPlugin, PieSeries, AccumulationTooltip } from
"@syncfusion/ej2-vue-charts";
Vue.use(AccumulationChartPlugin);
export default {
    data() {
        return {
            seriesData: [
                { x: 'Jan', y: 3 }, { x: 'Feb', y: 3.5 },
                { x: 'Mar', y: 7 }, { x: 'Apr', y: 13.5 },
                { x: 'May', y: 19 }, { x: 'Jun', y: 23.5 },
                { x: 'Jul', y: 26 }, { x: 'Aug', y: 25 },
                { x: 'Sep', y: 21 }, { x: 'Oct', y: 15 }
            ],
            tooltip:{
                enable: true
            }
        };
    },
    provide: {
        accumulationchart: [PieSeries, AccumulationTooltip]
    },
    methods: {
        tooltipRender: function(args) {
            if (args.point.index === 3) {
                args.text = args.point.x + ' ' + ':' + args.point.y + ' ' + ' '
+ 'customtext';
                args.textStyle.color = '#f48042';
            }
        }
    }
};
</script>
<style>
    #container {
        height: 350px;
    }
</style>

```

{% previewsample "page.domainurl/code-snippet/chart/series/pie-cs32" %}

### Tooltip mapping name

By default, tooltip shows information of x and y value in points. You can show more information from datasource in tooltip by using the [tooltipMappingName](#) property of the tooltip. You can use the `${point.tooltip}` as place holders to display the specified tooltip content.



**APP.VUE**

```

<template>
  <div id="app">
    <ejs-accumulationchart id="container" :tooltip='tooltip'>
      <e-accumulation-series-collection>
        <e-accumulation-series :dataSource='seriesData' xName='x'
yName='y' tooltipMappingName='text' radius='70%'> </e-accumulation-series>
      </e-accumulation-series-collection>
    </ejs-accumulationchart>
  </div>
</template>
<script>
import Vue from "vue";
import { AccumulationChartPlugin, PieSeries, AccumulationTooltip } from
"@syncfusion/ej2-vue-charts";
Vue.use(AccumulationChartPlugin);
export default {
  data() {
    return {
      seriesData: [
        { x: 'Jan', y: 13, text: 'Jan: 13' }, { x: 'Feb', y: 13,
text: 'Feb: 13' },
        { x: 'Mar', y: 17, text: 'Mar: 17' }, { x: 'Apr', y:
13.5, text: 'Apr: 13.5' }
      ],
      tooltip:{
        enable: true, format: '${point.tooltip}'
      }
    };
  },
  provide: {
    accumulationchart: [PieSeries, AccumulationTooltip]
  },
};
</script>
<style>
  #container {
    height: 350px;
  }
</style>

```

```
{% previewsample "page.domainurl/code-snippet/chart/series/pie-cs33" %}
```

**Legend in Vue Accumulation chart component**

As like a chart, the legend is also available for accumulation charts, which gives information about the points. By default, the legend will be placed on the right, if the width of the chart is high or will be placed on the bottom, if the height of the chart is high. Other customization features regarding the legend element are same as the [chart legend](#). Here, the legend for a point can be collapsed by giving the empty string to the x value of the point.

**APP.VUE**

```

<template>
  <div id="app">

```

```

        <ejs-accumulationchart id="container"
:legendSettings='legendSettings'>
            <e-accumulation-series-collection>
                <e-accumulation-series :dataSource='seriesData' xName='x'
yName='y' radius='70%'> </e-accumulation-series>
            </e-accumulation-series-collection>
        </ejs-accumulationchart>
    </div>
</template>
<script>
import Vue from "vue";
import { AccumulationChartPlugin, PieSeries, AccumulationLegend } from
"@syncfusion/ej2-vue-charts";
Vue.use(AccumulationChartPlugin);
export default {
    data() {
        return {
            seriesData: [
                { x: 'Jan', y: 3 }, { x: 'Feb', y: 3.5 },
                { x: 'Mar', y: 7 }, { x: 'Apr', y: 13.5 },
                { x: 'May', y: 19 }, { x: 'Jun', y: 23.5 },
                { x: 'Jul', y: 26 }, { x: 'Aug', y: 25 },
                { x: 'Sep', y: 21 }, { x: 'Oct', y: 15 }
            ],
            legendSettings: {
                visible: true
            }
        };
    },
    provide: {
        accumulationchart: [PieSeries, AccumulationLegend]
    }
};
</script>
<style>
    #container {
        height: 350px;
    }
</style>

```

{% previewsample "page.domainurl/code-snippet/chart/series/legend-cs1" %}

### Position and Alignment

By using the position property, you can position the legend at the **left**, **right**, **top** or **bottom** of the chart. You can also align the legend to **center**, **far** or **near** of the chart using the alignment property.

### APP.VUE

```

<template>
    <div id="app">
        <ejs-accumulationchart id="container"
:legendSettings='legendSettings'>
            <e-accumulation-series-collection>
                <e-accumulation-series :dataSource='seriesData' xName='x'
yName='y' radius='70%'> </e-accumulation-series>
            </e-accumulation-series-collection>
        </ejs-accumulationchart>
    </div>
</template>

```

```

        </ejs-accumulationchart>
    </div>
</template>
<script>
import Vue from "vue";
import { AccumulationChartPlugin, PieSeries, AccumulationLegend } from
"@syncfusion/ej2-vue-charts";
Vue.use(AccumulationChartPlugin);
export default {
    data() {
        return {
            seriesData: [
                { x: 'Jan', y: 3 }, { x: 'Feb', y: 3.5 },
                { x: 'Mar', y: 7 }, { x: 'Apr', y: 13.5 },
                { x: 'May', y: 19 }, { x: 'Jun', y: 23.5 },
                { x: 'Jul', y: 26 }, { x: 'Aug', y: 25 },
                { x: 'Sep', y: 21 }, { x: 'Oct', y: 15 }
            ],
            legendSettings:{ position:'Top' ,alignment:'Near'}
        };
    },
    provide: {
        accumulationchart: [PieSeries, AccumulationLegend]
    }
};
</script>
<style>
#container {
    height: 350px;
}
</style>

```

{% previewsample "page.domainurl/code-snippet/chart/series/pie-cs10" %}

### Legend Reverse

You can reverse the order of the legend items by using the [reverse](#) property. By default, legend for the first series in the collection will be placed first.

### APP.VUE

```

<template>
  <ejs-accumulationchart
    id="container"
    ref="pie"
    style="display: block"
    :legendSettings="legendSettings"
    :tooltip="tooltip"
    :enableAnimation="enableAnimation"
    :enableSmartLabels="enableSmartLabels"
  >
    <e-accumulation-series-collection>
      <e-accumulation-series
        :dataSource="data"
        xName="x"
        yName="y"
        :radius="radius"
      >
    </e-accumulation-series-collection>
  </ejs-accumulationchart>
</template>

```

```

        innerRadius="20%"
        :dataLabel="dataLabel"
      >
    </e-accumulation-series>
  </e-accumulation-series-collection>
</ejs-accumulationchart>
</template>
<script>
import Vue from "vue";
import {
  AccumulationChartPlugin,
  AccumulationLegend,
  PieSeries,
  AccumulationDataLabel,
  AccumulationTooltip,
} from "@syncfusion/ej2-vue-charts";
Vue.use(AccumulationChartPlugin);
export default Vue.extend({
  data: function () {
    return {
      data: [
        { x: "Argentina", y: 505370, r: "50%" },
        { x: "Belgium", y: 551500, r: "70%" },
        { x: "Cuba", y: 312685, r: "84%" },
        { x: "Dominican Republic", y: 350000, r: "97%" },
        { x: "Egypt", y: 301000, r: "84%" },
        { x: "Kazakhstan", y: 300000, r: "70%" },
        { x: "Somalia", y: 357022, r: "90%" },
      ],
      radius: "r",
      legendSettings: { visible: true, reverse: true },
      dataLabel: { visible: true, position: "Outside", name: "x" },
      tooltip: {
        enable: true,
        header: "<b>${point.x}</b>",
        format: "Composition: <b>${point.y}</b>",
      },
      enableAnimation: true,
      enableSmartLabels: true,
    };
  },
  provide: {
    accumulationchart: [
      AccumulationLegend,
      PieSeries,
      AccumulationDataLabel,
      AccumulationTooltip,
    ],
  },
});
</script>

```

{% previewsample "page.domainurl/code-snippet/chart/series/legend-reverse-cs1" %}

### Legend Shape

To change the legend icon shape, use the `legendShape` property in the `series`. By default, legend icon shape

is `seriesType`.

### APP.VUE

```
<template>
  <div id="app">
    <ejs-accumulationchart id="container"
:legendSettings='legendSettings'>
      <e-accumulation-series-collection>
        <e-accumulation-series :dataSource='seriesData' xName='x'
yName='y' legendShape='Rectangle'> </e-accumulation-series>
      </e-accumulation-series-collection>
    </ejs-accumulationchart>
  </div>
</template>
<script>
import Vue from "vue";
import { AccumulationChartPlugin, PieSeries, AccumulationLegend } from
"@syncfusion/ej2-vue-charts";
Vue.use(AccumulationChartPlugin);
export default {
  data() {
    return {
      seriesData: [
        { x: 'Jan', y: 3 }, { x: 'Feb', y: 3.5 },
        { x: 'Mar', y: 7 }, { x: 'Apr', y: 13.5 },
        { x: 'May', y: 19 }, { x: 'Jun', y: 23.5 },
        { x: 'Jul', y: 26 }, { x: 'Aug', y: 25 },
        { x: 'Sep', y: 21 }, { x: 'Oct', y: 15 }
      ],
      legendSettings:{ visible: true }
    };
  },
  provide: {
    accumulationchart: [PieSeries, AccumulationLegend]
  }
};
</script>
<style>
#container {
  height: 350px;
}
</style>
```

{% previewsample "page.domainurl/code-snippet/chart/series/pie-cs11" %}

### Legend Size

The legend size can be changed by using the `width` and `height` properties of the `legendSettings`.

### APP.VUE

```
<template>
```

```

<div id="app">
  <ejs-accumulationchart id="container"
:legendSettings='legendSettings'>
    <e-accumulation-series-collection>
      <e-accumulation-series :dataSource='seriesData' xName='x'
yName='y'> </e-accumulation-series>
    </e-accumulation-series-collection>
  </ejs-accumulationchart>
</div>
</template>
<script>
import Vue from "vue";
import { AccumulationChartPlugin, PieSeries, AccumulationLegend } from
"@syncfusion/ej2-vue-charts";
Vue.use(AccumulationChartPlugin);
export default {
  data() {
    return {
      seriesData: [
        { x: 'Jan', y: 3 }, { x: 'Feb', y: 3.5 },
        { x: 'Mar', y: 7 }, { x: 'Apr', y: 13.5 },
        { x: 'May', y: 19 }, { x: 'Jun', y: 23.5 },
        { x: 'Jul', y: 26 }, { x: 'Aug', y: 25 },
        { x: 'Sep', y: 21 }, { x: 'Oct', y: 15 }
      ],
      legendSettings:{ width: '150', height: '100', border: { width: 1,
color: 'pink' } }
    };
  },
  provide: {
    accumulationchart: [PieSeries, AccumulationLegend]
  }
};
</script>
<style>
#container {
  height: 350px;
}
</style>

```

{% previewsample "page.domainurl/code-snippet/chart/series/pie-cs12" %}

### Legend Item Size

You can customize the size of the legend items by using the `shapeHeight` and `shapeWidth` properties.

### APP.VUE

```

<template>
  <div id="app">
    <ejs-accumulationchart id="container"
:legendSettings='legendSettings'>
      <e-accumulation-series-collection>
        <e-accumulation-series :dataSource='seriesData' xName='x'
yName='y'> </e-accumulation-series>
      </e-accumulation-series-collection>
    </ejs-accumulationchart>
  </div>
</template>

```

```

    </div>
  </template>
  <script>
    import Vue from "vue";
    import { AccumulationChartPlugin, PieSeries, AccumulationLegend } from
    "@syncfusion/ej2-vue-charts";
    Vue.use(AccumulationChartPlugin);
    export default {
      data() {
        return {
          seriesData: [
            { x: 'Jan', y: 3 }, { x: 'Feb', y: 3.5 },
            { x: 'Mar', y: 7 }, { x: 'Apr', y: 13.5 },
            { x: 'May', y: 19 }, { x: 'Jun', y: 23.5 },
            { x: 'Jul', y: 26 }, { x: 'Aug', y: 25 },
            { x: 'Sep', y: 21 }, { x: 'Oct', y: 15 }
          ],
          legendSettings: { shapeHeight: 15, shapeWidth: 15 }
        };
      },
      provide: {
        accumulationchart: [PieSeries, AccumulationLegend]
      }
    };
  </script>
  <style>
    #container {
      height: 350px;
    }
  </style>

```

{% previewsample "page.domainurl/code-snippet/chart/series/pie-cs13" %}

### Paging for Legend

Paging will be enabled by default, when the legend items exceeds the legend bounds. You can view the each legend item by navigating between the pages using the navigation buttons.

### APP.VUE

```

<template>
  <div id="app">
    <ejs-accumulationchart id="container"
    :legendSettings='legendSettings'>
      <e-accumulation-series-collection>
        <e-accumulation-series :dataSource='seriesData' xName='x'
        yName='y'> </e-accumulation-series>
      </e-accumulation-series-collection>
    </ejs-accumulationchart>
  </div>
</template>
<script>
  import Vue from "vue";
  import { AccumulationChartPlugin, PieSeries, AccumulationLegend } from
  "@syncfusion/ej2-vue-charts";
  Vue.use(AccumulationChartPlugin);
  export default {

```

```

data() {
  return {
    seriesData: [
      { x: 'Jan', y: 3 }, { x: 'Feb', y: 3.5 },
      { x: 'Mar', y: 7 }, { x: 'Apr', y: 13.5 },
      { x: 'May', y: 19 }, { x: 'Jun', y: 23.5 },
      { x: 'Jul', y: 26 }, { x: 'Aug', y: 25 },
      { x: 'Sep', y: 21 }, { x: 'Oct', y: 15 }
    ],
    legendSettings:{ height: '150', width:'80' }
  };
},
provide: {
  accumulationchart: [PieSeries, AccumulationLegend]
}
};
</script>
<style>
  #container {
    height: 350px;
  }
</style>

```

{% previewsample "page.domainurl/code-snippet/chart/series/pie-cs14" %}

### Legend Text Wrap

When the legend text exceeds the container, the text can be wrapped by using `textWrap` Property. End user can also wrap the legend text based on the `maximumLabelWidth` property.

### APP.VUE

```

<template>
  <div id="app">
    <ejs-accumulationchart id="container"
      :legendSettings='legendSettings'>
      <e-accumulation-series-collection>
        <e-accumulation-series :dataSource='seriesData' xName='x'
          yName='y'> </e-accumulation-series>
      </e-accumulation-series-collection>
    </ejs-accumulationchart>
  </div>
</template>
<script>
import Vue from "vue";
import { AccumulationChartPlugin, PieSeries, AccumulationLegend } from
"@syncfusion/ej2-vue-charts";
Vue.use(AccumulationChartPlugin);
export default {
  data() {
    return {
      seriesData: [
        { 'x': 'Net-tution', y: 21, text: '21%' },
        { 'x': 'Private Gifts', y: 8, text: '8%' },
        { 'x': 'All Other', y: 9, text: '9%' },
        { 'x': 'Local Revenue', y: 4, text: '4%' },
        { 'x': 'State Revenue', y: 21, text: '21%' },
      ]
    }
  }
}

```



```

        { 'x': 'Federal Revenue', y: 16, text: '16%' },
        { 'x': 'Self-supporting Operations', y: 21, text: '21%'
    },
    ],
    xName: 'x', yName: 'y', startAngle: 0, endAngle: 360,
    innerRadius: '40%',
    type: 'Pie',
    legendSettings:{ visible:true, position:'Right', textWrap:'Wrap',
maximumLabelWidth:60, height:'44%', width:'64%' }
    };
},
provide: {
    accumulationchart: [PieSeries, AccumulationLegend]
}
};
</script>
<style>
    #container {
        height: 350px;
    }
</style>

```

{% previewsample "page.domainurl/code-snippet/chart/series/pie-cs15" %}

### Enable Animation

You can customize the animation while clicking legend by setting enableAnimation as true or false using enableAnimation property in Accumulation Chart.

### APP.VUE

```

<template>
    <div id="app">
        <ejs-accumulationchart id="container"
:legendSettings='legendSettings'>
            <e-accumulation-series-collection>
                <e-accumulation-series :dataSource='seriesData' xName='x'
yName='y' legendShape='Rectangle'> </e-accumulation-series>
            </e-accumulation-series-collection>
        </ejs-accumulationchart>
    </div>
</template>
<script>
import Vue from "vue";
import { AccumulationChartPlugin, PieSeries, AccumulationLegend } from
"@syncfusion/ej2-vue-charts";
Vue.use(AccumulationChartPlugin);
export default {
    data() {
        return {
            seriesData: [
                { x: 'Jan', y: 3 }, { x: 'Feb', y: 3.5 },
                { x: 'Mar', y: 7 }, { x: 'Apr', y: 13.5 },
                { x: 'May', y: 19 }, { x: 'Jun', y: 23.5 },
                { x: 'Jul', y: 26 }, { x: 'Aug', y: 25 },
                { x: 'Sep', y: 21 }, { x: 'Oct', y: 15 }
            ],

```

```

        legendSettings:{ visible: true },
        enableAnimation: true
    };
},
provide: {
    accumulationchart: [PieSeries, AccumulationLegend]
}
};
</script>
<style>
#container {
    height: 350px;
}
</style>

```

{% previewsample "page.domainurl/code-snippet/chart/series/pie-cs16" %}

### Legend Title

You can set title for legend using `title` property in `legendSettings`. You can also customize the `fontStyle`, `size`, `fontWeight`, `color`, `textAlignment`, `fontFamily`, `opacity` and `textOverflow` of legend title. `titlePosition` is used to set the legend position in `Top`, `Left` and `Right` position. `maximumTitleWidth` is used to set the width of the legend title. By default, it will be `100px`.

### APP.VUE

```

<template>
  <div id="app">
    <ejs-accumulationchart id="container"
      :legendSettings='legendSettings'>
      <e-accumulation-series-collection>
        <e-accumulation-series :dataSource='seriesData' xName='x'
          yName='y' radius='70%'> </e-accumulation-series>
      </e-accumulation-series-collection>
    </ejs-accumulationchart>
  </div>
</template>
<script>
import Vue from "vue";
import { AccumulationChartPlugin, PieSeries, AccumulationLegend } from
"@syncfusion/ej2-vue-charts";
Vue.use(AccumulationChartPlugin);
export default {
  data() {
    return {
      seriesData: [
        { x: 'Jan', y: 3 }, { x: 'Feb', y: 3.5 },
        { x: 'Mar', y: 7 }, { x: 'Apr', y: 13.5 },
        { x: 'May', y: 19 }, { x: 'Jun', y: 23.5 },
        { x: 'Jul', y: 26 }, { x: 'Aug', y: 25 },
        { x: 'Sep', y: 21 }, { x: 'Oct', y: 15 }
      ],
      legendSettings:{ visible: true, title: 'Months', position: 'Bottom' }
    };
  },
  provide: {

```

```

        accumulationchart: [PieSeries, AccumulationLegend]
    }
};
</script>
<style>
    #container {
        height: 350px;
    }
</style>

```

{% previewsample "page.domainurl/code-snippet/chart/series/pie-cs17" %}

### Arrow Page Navigation

By default, the page number will be enabled while legend paging. Now, you can disable that page number and also you can get left and right arrows for page navigation. You have to set `false` value to `enablePages` to get this support.

### APP.VUE

```

<template>
    <div id="app">
        <ejs-accumulationchart id="container"
        :legendSettings='legendSettings'>
            <e-accumulation-series-collection>
                <e-accumulation-series :dataSource='seriesData' xName='x'
                yName='y' radius='70%'> </e-accumulation-series>
            </e-accumulation-series-collection>
        </ejs-accumulationchart>
    </div>
</template>
<script>
import Vue from "vue";
import { AccumulationChartPlugin, PieSeries, AccumulationLegend } from
"@syncfusion/ej2-vue-charts";
Vue.use(AccumulationChartPlugin);
export default {
    data() {
        return {
            seriesData: [
                { x: 'Jan', y: 3 }, { x: 'Feb', y: 3.5 },
                { x: 'Mar', y: 7 }, { x: 'Apr', y: 13.5 },
                { x: 'May', y: 19 }, { x: 'Jun', y: 23.5 },
                { x: 'Jul', y: 26 }, { x: 'Aug', y: 25 },
                { x: 'Sep', y: 21 }, { x: 'Oct', y: 15 },
                { x: 'Nov', y: 15 }, { x: 'Dec', y: 15 }
            ],
            legendSettings:{ width: '260px', height: '50px', enablePages: false,
            position: 'Bottom' }
        };
    },
    provide: {
        accumulationchart: [PieSeries, AccumulationLegend]
    }
};
</script>
<style>

```

```
#container {
  height: 350px;
}
</style>
```

{% previewsample "page.domainurl/code-snippet/chart/series/pie-cs18" %}

### Legend Item Padding

The [itemPadding](#) property can be used to adjust the space between the legend items.

### APP.VUE

```
<template>
  <div id="app">
    <ejs-accumulationchart id="container"
:legendSettings='legendSettings'>
      <e-accumulation-series-collection>
        <e-accumulation-series :dataSource='seriesData' xName='x'
yName='y' radius='70%'> </e-accumulation-series>
      </e-accumulation-series-collection>
    </ejs-accumulationchart>
  </div>
</template>
<script>
import Vue from "vue";
import { AccumulationChartPlugin, PieSeries, AccumulationLegend } from
"@syncfusion/ej2-vue-charts";
Vue.use(AccumulationChartPlugin);
export default {
  data() {
    return {
      seriesData: [
        { x: 'Jan', y: 3 }, { x: 'Feb', y: 3.5 },
        { x: 'Mar', y: 7 }, { x: 'Apr', y: 13.5 },
        { x: 'May', y: 19 }, { x: 'Jun', y: 23.5 },
        { x: 'Jul', y: 26 }, { x: 'Aug', y: 25 },
        { x: 'Sep', y: 21 }, { x: 'Oct', y: 15 },
        { x: 'Nov', y: 15 }, { x: 'Dec', y: 15 }
      ],
      legendSettings:{ width: '260px', height: '50px', position: "Bottom",
itemPadding: 30 }
    };
  },
  provide: {
    accumulationchart: [PieSeries, AccumulationLegend]
  }
};
</script>
<style>
#container {
  height: 350px;
}
</style>
```

{% previewsample "page.domainurl/code-snippet/chart/series/pie-cs19" %}

## Center Label in Vue Accumulation chart component

## Center label

Using [centerLabel](#) it is now possible to place a label at the center of a pie or donut chart. To configure the default text rendered on the center label for the pie and doughnut charts, use the [text](#) property in the [centerLabel](#).

**APP.VUE**

```
<template>
  <div id="app">
    <ejs-accumulationchart id="container" :centerLabel="centerLabel">
      <e-accumulation-series-collection>
        <e-accumulation-series :dataSource='seriesData' xName='x'
yName='y' innerRadius='65%'> </e-accumulation-series>
      </e-accumulation-series-collection>
    </ejs-accumulationchart>
  </div>
</template>
<script>
import Vue from "vue";
import { AccumulationChartPlugin, PieSeries } from "@syncfusion/ej2-vue-
charts";
Vue.use(AccumulationChartPlugin);
export default {
  data() {
    return {
      seriesData: [
        { x: 'Chrome', y: 61.3, text: 'Chrome: 61.3%' },
        { x: 'Safari', y: 24.6, text: 'Safari: 24.6%' },
        { x: 'Edge', y: 5.0, text: 'Edge: 5.0%' },
        { x: 'Samsung Internet', y: 2.7, text: 'Samsung Internet: 2.7%' },
        { x: 'Firefox', y: 2.6, text: 'Firefox: 2.6%' },
        { x: 'Others', y: 3.6, text: 'Others: 3.6%' }
      ],
      centerLabel: {
        text : 'Mobile<br>Browsers<br>Statistics'
      }
    };
  },
  provide: {
    accumulationchart: [PieSeries]
  }
};
</script>
<style>
  #container {
    height: 350px;
  }
</style>
```

{% previewsample "page.domainurl/code-snippet/chart/series/pie-cs34" %}

### Hover text

The default text in the center label can be changed when the mouse pointer hovers over the pie and doughnut charts slice using the [hoverTextFormat](#) property.

### APP.VUE

```
<template>
  <div id="app">
    <ejs-accumulationchart id="container" :centerLabel="centerLabel">
      <e-accumulation-series-collection>
        <e-accumulation-series :dataSource='seriesData' xName='x'
yName='y' innerRadius='65%'> </e-accumulation-series>
      </e-accumulation-series-collection>
    </ejs-accumulationchart>
  </div>
</template>
<script>
import Vue from "vue";
import { AccumulationChartPlugin, PieSeries } from "@syncfusion/ej2-vue-
charts";
Vue.use(AccumulationChartPlugin);
export default {
  data() {
    return {
      seriesData: [
        { x: 'Chrome', y: 61.3, text: 'Chrome: 61.3%' },
        { x: 'Safari', y: 24.6, text: 'Safari: 24.6%' },
        { x: 'Edge', y: 5.0, text: 'Edge: 5.0%' },
        { x: 'Samsung Internet', y: 2.7, text: 'Samsung Internet: 2.7%' },
        { x: 'Firefox', y: 2.6, text: 'Firefox: 2.6%' },
        { x: 'Others', y: 3.6, text: 'Others: 3.6%' }
      ],
      centerLabel:{
        text : 'Mobile<br>Browsers<br>Statistics',
        hoverTextFormat: '$${point.x} <br> Browser Share <br>
${point.y}%'
      }
    };
  },
  provide: {
    accumulationchart: [PieSeries]
  }
};
</script>
<style>
  #container {
    height: 350px;
  }
</style>
```

{% previewsample "page.domainurl/code-snippet/chart/series/pie-cs35" %}

### Customization

Customize the center label text using the [textStyle](#) property.

**APP.VUE**

```

<template>
  <div id="app">
    <ejs-accumulationchart id="container" :centerLabel="centerLabel">
      <e-accumulation-series-collection>
        <e-accumulation-series :dataSource='seriesData' xName='x'
yName='y' innerRadius='65%'> </e-accumulation-series>
      </e-accumulation-series-collection>
    </ejs-accumulationchart>
  </div>
</template>
<script>
import Vue from "vue";
import { AccumulationChartPlugin, PieSeries } from "@syncfusion/ej2-vue-
charts";
Vue.use(AccumulationChartPlugin);
export default {
  data() {
    return {
      seriesData: [
        { x: 'Chrome', y: 61.3, text: 'Chrome: 61.3%' },
        { x: 'Safari', y: 24.6, text: 'Safari: 24.6%' },
        { x: 'Edge', y: 5.0, text: 'Edge: 5.0%' },
        { x: 'Samsung Internet', y: 2.7, text: 'Samsung Internet: 2.7%' },
        { x: 'Firefox', y: 2.6, text: 'Firefox: 2.6%' },
        { x: 'Others', y: 3.6, text: 'Others: 3.6%' }
      ],
      centerLabel: {
        text : 'Mobile<br>Browsers<br>Statistics',
        hoverTextFormat: '${point.x} <br> Browser Share <br>
${point.y}%',
        textStyle:{
          fontWeight: '900',
          size: '15px',
          color: 'red',
          fontFamily: 'Roboto',
          fontStyle: 'Italic'
        }
      }
    };
  },
  provide: {
    accumulationchart: [PieSeries]
  }
};
</script>
<style>
  #container {
    height: 350px;
  }
</style>

```

{% previewsample "page.domainurl/code-snippet/chart/series/pie-cs36" %}

## Title and sub title in Vue Accumulation chart component

### Title

Accumulation Chart can be given a title using [title](#) property, to show the information about the data plotted.

### APP.VUE

```
<template>
  <div id="app">
    <ejs-chart id="container" :title='title'
:primaryXAxis='primaryXAxis'>
      <e-series-collection>
        <e-series :dataSource='seriesData' type='Column'
xName='country' yName='gold' name='Gold'> </e-series>
      </e-series-collection>
    </ejs-chart>
  </div>
</template>
<script>
import Vue from "vue";
import { ChartPlugin, ColumnSeries, Category } from "@syncfusion/ej2-vue-
charts";
Vue.use(ChartPlugin);
export default {
  data() {
    return {
      seriesData: [
        { country: "USA", gold: 50 },
        { country: "China", gold: 40 },
        { country: "Japan", gold: 70 },
        { country: "Australia", gold: 60 },
        { country: "France", gold: 50 },
        { country: "Germany", gold: 40 },
        { country: "Italy", gold: 40 },
        { country: "Sweden", gold: 30 }
      ],
      primaryXAxis: {
        valueType: 'Category',
        title: 'Countries'
      },
      title: "Olympic Medals"
    };
  },
  provide: {
    chart: [ColumnSeries, Category]
  },
};
</script>
<style>
  #container {
    height: 350px;
  }
</style>
```

{% previewsample "page.domainurl/code-snippet/chart/chart-title-cs1" %}



### Title customization

Accumulation Chart can be customizing a title using [titleStyle](#) property, to show the information about the data plotted.

#### APP.VUE

```
<template>
  <div id="app">
    <ejs-chart id="container" :title='title' :titleStyle='titleStyle'
:primaryXAxis='primaryXAxis'>
      <e-series-collection>
        <e-series :dataSource='seriesData' type='Column'
xName='country' yName='gold' name='Gold'> </e-series>
      </e-series-collection>
    </ejs-chart>
  </div>
</template>
<script>
import Vue from "vue";
import { ChartPlugin, ColumnSeries, Category } from "@syncfusion/ej2-vue-
charts";
Vue.use(ChartPlugin);
export default {
  data() {
    return {
      seriesData: [
        { country: "USA", gold: 50 },
        { country: "China", gold: 40 },
        { country: "Japan", gold: 70 },
        { country: "Australia", gold: 60 },
        { country: "France", gold: 50 },
        { country: "Germany", gold: 40 },
        { country: "Italy", gold: 40 },
        { country: "Sweden", gold: 30 }
      ],
      primaryXAxis: {
        valueType: 'Category',
        title: 'Countries'
      },
      title: "Olympic Medals",
      titleStyle:{
        fontFamily: "Arial",
        fontStyle: 'italic',
        fontWeight: 'regular',
        color: "#E27F2D",
        size: '23px'
      }
    };
  },
  provide: {
    chart: [ColumnSeries, Category]
  },
};
</script>
<style>
#container {
  height: 350px;
```

```
}
</style>
```

```
{% previewsample "page.domainurl/code-snippet/chart/chart-title-cs2" %}
```

### SubTitle

Accumulation Chart can be given a subtitle using [subTitle](#) property, to show the information about the data plotted.

### APP.VUE

```
<template>
  <div id="app">
    <ejs-chart id="container" :title='title' :subTitle='subTitle'
:primaryXAxis='primaryXAxis'>
      <e-series-collection>
        <e-series :dataSource='seriesData' type='Column'
xName='country' yName='gold' name='Gold'> </e-series>
      </e-series-collection>
    </ejs-chart>
  </div>
</template>
<script>
import Vue from "vue";
import { ChartPlugin, ColumnSeries, Category } from "@syncfusion/ej2-vue-
charts";
Vue.use(ChartPlugin);
export default {
  data() {
    return {
      seriesData: [
        { country: "USA", gold: 50 },
        { country: "China", gold: 40 },
        { country: "Japan", gold: 70 },
        { country: "Australia", gold: 60 },
        { country: "France", gold: 50 },
        { country: "Germany", gold: 40 },
        { country: "Italy", gold: 40 },
        { country: "Sweden", gold: 30 }
      ],
      primaryXAxis: {
        valueType: 'Category',
        title: 'Countries'
      },
      title: "Olympic Medals",
      subTitle:'In the year of 2014'
    };
  },
  provide: {
    chart: [ColumnSeries, Category]
  },
};
</script>
<style>
#container {
  height: 350px;
```

```
}
</style>
```

```
{% previewsample "page.domainurl/code-snippet/chart/chart-title-cs3" %}
```

### SubTitle Customization

Accumulation Chart can be customizing a subtitle using [subTitleStyle](#) property.

### APP.VUE

```
<template>
  <div id="app">
    <ejs-chart id="container" :title='title' :subTitle='subTitle'
    :subTitleStyle='subTitleStyle' :primaryXAxis='primaryXAxis'>
      <e-series-collection>
        <e-series :dataSource='seriesData' type='Column'
        xName='country' yName='gold' name='Gold'> </e-series>
      </e-series-collection>
    </ejs-chart>
  </div>
</template>
<script>
import Vue from "vue";
import { ChartPlugin, ColumnSeries, Category } from "@syncfusion/ej2-vue-
charts";
Vue.use(ChartPlugin);
export default {
  data() {
    return {
      seriesData: [
        { country: "USA", gold: 50 },
        { country: "China", gold: 40 },
        { country: "Japan", gold: 70 },
        { country: "Australia", gold: 60 },
        { country: "France", gold: 50 },
        { country: "Germany", gold: 40 },
        { country: "Italy", gold: 40 },
        { country: "Sweden", gold: 30 }
      ],
      primaryXAxis: {
        valueType: 'Category',
        title: 'Countries'
      },
      title: "Olympic Medals",
      subTitle:'In the year of 2014',
      subTitleStyle:{
        fontFamily: "Arial",
        fontStyle: 'italic',
        fontWeight: 'regular',
        color: "#E27F2D"
      }
    };
  },
  provide: {
    chart: [ColumnSeries, Category]
  },
}
```

```
};
</script>
<style>
  #container {
    height: 350px;
  }
</style>
```

{% previewsample "page.domainurl/code-snippet/chart/chart-title-cs4" %}

## Chart print in Vue Accumulation chart component

### Print

The rendered chart can be printed directly from the browser by calling the public method print.

### APP.VUE

```
<template>
  <div id="app">
    <ejs-button id='print' @click.native='print'>Print</ejs-button>
    <ejs-accumulationchart ref="chart" id="container">
      <e-accumulation-series-collection>
        <e-accumulation-series :dataSource='seriesData' xName='x'
yName='y' radius='70%'> </e-accumulation-series>
      </e-accumulation-series-collection>
    </ejs-accumulationchart>
  </div>
</template>
<script>
import Vue from "vue";
import { AccumulationChartPlugin, PieSeries } from "@syncfusion/ej2-vue-
charts";
import { ButtonPlugin } from "@syncfusion/ej2-vue-buttons";
Vue.use(AccumulationChartPlugin);
Vue.use(ButtonPlugin);
export default {
  data() {
    return {
      seriesData: [
        { x: 'Jan', y: 3, text: 'Jan: 3' }, { x: 'Feb', y: 3.5,
text: 'Feb: 3.5' },
        { x: 'Mar', y: 7, text: 'Mar: 7' }, { x: 'Apr', y: 13.5,
text: 'Apr: 13.5' },
        { x: 'May', y: 19, text: 'May: 19' }, { x: 'Jun', y: 23.5,
text: 'Jun: 23.5' },
        { x: 'Jul', y: 26, text: 'Jul: 26' }, { x: 'Aug', y: 25,
text: 'Aug: 25' },
        { x: 'Sep', y: 21, text: 'Sep: 21' }, { x: 'Oct', y: 15,
text: 'Oct: 15' } ],
    };
  },
  provide: {
    accumulationchart: [PieSeries]
  },
  methods: {
    print: function() {
      this.$refs.chart.print();
    }
  }
}
```

```

    }
  }
};
</script>
<style>
  #container {
    height: 350px;
  }
</style>

```

{% previewsample "page.domainurl/code-snippet/chart/series/radius-cs1" %}

### Export

The rendered chart can be exported to **Image**(jpeg or png) or **SVG** or **PDF** format by using the export method.

Input parameters for this method are **Export Type** for **format** and **fileName** of result.

### APP.VUE

```

<template>
  <div id="app">
    <ejs-button id='togglebtn' @click.native='exportIcon'>Export</ejs-button>
    <ejs-accumulationchart ref="chart" id="container">
      <e-accumulation-series-collection>
        <e-accumulation-series :dataSource='seriesData' xName='x'
yName='y' radius='70%'> </e-accumulation-series>
      </e-accumulation-series-collection>
    </ejs-accumulationchart>
  </div>
</template>
<script>
import Vue from "vue";
import { AccumulationChartPlugin, PieSeries, Export } from "@syncfusion/ej2-vue-charts";
import { ButtonPlugin } from "@syncfusion/ej2-vue-buttons";
Vue.use(AccumulationChartPlugin);
Vue.use(ButtonPlugin);
export default {
  data() {
    return {
      seriesData: [
        { x: 'Jan', y: 3, text: 'Jan: 3' }, { x: 'Feb', y: 3.5,
text: 'Feb: 3.5' },
        { x: 'Mar', y: 7, text: 'Mar: 7' }, { x: 'Apr', y: 13.5,
text: 'Apr: 13.5' },
        { x: 'May', y: 19, text: 'May: 19' }, { x: 'Jun', y: 23.5,
text: 'Jun: 23.5' },
        { x: 'Jul', y: 26, text: 'Jul: 26' }, { x: 'Aug', y: 25,
text: 'Aug: 25' },
        { x: 'Sep', y: 21, text: 'Sep: 21' }, { x: 'Oct', y: 15,
text: 'Oct: 15' } ],
    };
  },
  provide: {

```

```

    accumulationchart: [PieSeries, Export]
  },
  methods: {
    exportIcon: function() {
      this.$refs.chart.export('PNG', 'export');
    }
  }
};
</script>
<style>
#container {
  height: 350px;
}
</style>

```

{% previewsample "page.domainurl/code-snippet/chart/series/radius-cs2" %}

## Appearance in Vue Accumulation chart component

### Custom Color Palette

You can customize the default color of series or points by providing a custom color palette of your choice by using the [palettes](#) property.

### APP.VUE

```

<template>
  <div id="app">
    <ejs-accumulationchart id="container">
      <e-accumulation-series-collection>
        <e-accumulation-series :dataSource='seriesData'
:palettes='palettes' xName='x' yName='y'> </e-accumulation-series>
      </e-accumulation-series-collection>
    </ejs-accumulationchart>
  </div>
</template>
<script>
import Vue from "vue";
import { AccumulationChartPlugin, PieSeries } from "@syncfusion/ej2-vue-
charts";
Vue.use(AccumulationChartPlugin);
export default {
  data() {
    return {
      seriesData: [
        { x: 'Jan', y: 3, fill: '#498fff', text: 'January' }, { x: 'Feb',
y: 3.5, fill: '#ffa060', text: 'February' },
        { x: 'Mar', y: 7, fill: '#ff68b6', text: 'March' }, { x: 'Apr',
y: 13.5, fill: '#81e2a1', text: 'April' }
      ],
      palettes: ["#E94649", "#F6B53F", "#6FAAB0",
"#FF33F3", "#228B22", "#3399FF"],
    };
  },
  provide: {
    accumulationchart: [PieSeries]
  }
};

```

```

</script>
<style>
  #container {
    height: 350px;
  }
</style>

```

```
{% previewsample "page.domainurl/code-snippet/chart/series/pie-cs5" %}
```

## Animation

### Fluid Animation

Fluid animation used to animate series with updated dataSource continues animation rather than animation whole series. You can customize animation for a particular series using `[animate]` method.

## APP.VUE

```

<template>
  <div id="app">
    <ejs-accumulationchart id="container" ref="pie" :title="title"
    :loaded="loaded">
      <e-accumulation-series-collection>
        <e-accumulation-series name="Revenue" :dataSource="data"
        xName="x" yName="y" :startAngle="startAngle"
        :endAngle="endAngle" innerRadius="40%"
        :dataLabel="dataLabel">
          </e-accumulation-series>
        </e-accumulation-series-collection>
      </ejs-accumulationchart>
    </div>
  </template>
<script>
import Vue from "vue";
import { AccumulationChartPlugin, PieSeries, AccumulationDataLabel } from
"@syncfusion/ej2-vue-charts";
Vue.use(AccumulationChartPlugin);
let count = 0;
let datasource1 = [
  { x: "Net-tution and Fees", y: 10 },
  { x: "Self-supporting Operations", y: 10 },
  { x: "Private Gifts", y: 13 },
  { x: "All Other", y: 14 },
  { x: "Local Revenue", y: 9 },
  { x: "State Revenue", y: 13 },
  { x: "Federal Revenue", y: 8 }
];
let datasource2 = [
  { x: "Net-tution and Fees", y: 120 },
  { x: "Self-supporting Operations", y: 31 },
  { x: "Private Gifts", y: 6 },
  { x: "All Other", y: 12 },
  { x: "Local Revenue", y: 25 },
  { x: "State Revenue", y: 11 },
  { x: "Federal Revenue", y: 12 }
];
let datasource3 = [
  { x: "Net-tution and Fees", y: 6 },

```

```

    { x: "Self-supporting Operations", y: 22 },
    { x: "Private Gifts", y: 11 },
    { x: "All Other", y: 15 },
    { x: "Local Revenue", y: 13 },
    { x: "State Revenue", y: 10 },
    { x: "Federal Revenue", y: 8 }
  ];
  let datasource4 = [
    { x: "Net-tution and Fees", y: 15 },
    { x: "Self-supporting Operations", y: 10 },
    { x: "Private Gifts", y: 18 },
    { x: "All Other", y: 20 },
    { x: "Local Revenue", y: 30 },
    { x: "State Revenue", y: 20 },
    { x: "Federal Revenue", y: 25 }
  ];
  let datasource5 = [
    { x: "Net-tution and Fees", y: 21 },
    { x: "Self-supporting Operations", y: 10 },
    { x: "Private Gifts", y: 15 },
    { x: "All Other", y: 15 },
    { x: "Local Revenue", y: 11 },
    { x: "State Revenue", y: 20 },
    { x: "Federal Revenue", y: 60 }
  ];
  export default {
    data() {
      return {
        data: [
          { x: "Net-tution and Fees", y: 21, text: "21%" },
          { x: "Self-supporting Operations", y: 21, text: "21%" },
          { x: "Private Gifts", y: 8, text: "8%" },
          { x: "All Other", y: 8, text: "8%" },
          { x: "Local Revenue", y: 4, text: "4%" },
          { x: "State Revenue", y: 21, text: "21%" },
          { x: "Federal Revenue", y: 16, text: "16%" }
        ],
        dataLabel: {
          visible: true,
          position: "Inside",
          font: {
            color: "white",
            fontWeight: "Bold",
            size: "14px"
          }
        },
        startAngle: 0,
        endAngle: 360,
        title: "Education Institutional Revenue"
      };
    },
    methods: {
      loaded: function(args) {
        this.$refs.pie.ej2Instances.loaded = null;
        setInterval(() => {
          if (count === 0) {
            this.$refs.pie.ej2Instances.series[0].dataSource = datasource1;

```



```

        this.$refs.pie.ej2Instances.animate();
        count++;
    } else if (count === 1) {
        this.$refs.pie.ej2Instances.series[0].dataSource = datasource2;
        this.$refs.pie.ej2Instances.animate();
        count++;
    } else if (count === 2) {
        this.$refs.pie.ej2Instances.series[0].dataSource = datasource3;
        this.$refs.pie.ej2Instances.animate();
        count++;
    } else if (count === 3) {
        this.$refs.pie.ej2Instances.series[0].dataSource = datasource4;
        this.$refs.pie.ej2Instances.animate();
        count++;
    } else if (count === 4) {
        this.$refs.pie.ej2Instances.series[0].dataSource = datasource5;
        this.$refs.pie.ej2Instances.animate();
        count = 0;
    }
    }, 3000);
    },
    },
    provide: {
        accumulationchart: [ PieSeries, AccumulationDataLabel]
    }
    };
</script>

```

{% previewsample "page.domainurl/code-snippet/chart/series/pie-cs6" %}

### Accessibility in Vue Accumulation chart component

The Accumulation chart component followed the accessibility guidelines and standards, including [ADA](#), [Section 508](#), [WCAG 2.2](#) standards, and [WCAG roles](#) that are commonly used to evaluate accessibility.

The accessibility compliance for the Accumulation chart component is outlined below.

| Accessibility Criteria | Compatibility |

| -- | -- |

| [WCAG 2.2](#) Support |  |

| [Section 508](#) Support |  |

| Screen Reader Support |  |

| Right-To-Left Support |  |

| Color Contrast |  |

| Mobile Device Support |  |

```

| Keyboard Navigation Support |  |

| Accessibility Checker Validation |  |

| Axe-core Accessibility Validation |  |

<style>
.post .post-content img {
display: inline-block;
margin: 0.5em 0;
}
</style>

<div> - All
features of the component meet the requirement.</div>

<div> - Some features of the component do not meet the requirement.</div>

<div> - The component does not meet the requirement.</div>

```

### WAI-ARIA attributes

The Accumulation chart component followed the [WAI-ARIA](#) patterns to meet the accessibility. The following ARIA attributes are used in the Accumulation chart component:

- `img` (role)
- `button` (role)
- `region` (role)
- `aria-label` (attribute)
- `aria-hidden` (attribute)
- `aria-pressed` (attribute)

### Keyboard interaction

The Accumulation chart component followed the [keyboard interaction](#) guideline, making it easy for people who use assistive technologies (AT) and those who completely rely on keyboard navigation. The following keyboard shortcuts are supported by the Accumulation chart component.

| **Press** | **To do this** |

| --- | --- |

| **Alt + J** | Moves the focus to the Accumulation chart element. |

| **Tab** | Moves the focus to the next element in the Accumulation chart. |

| **Shift + Tab** | Moves the focus to the previous element in the Accumulation chart. |

| **Down Arrow** | Moves the focus to the data point left side from the selected point. |

- | Up Arrow | Moves the focus to the data point right side from the selected point. |
- | Down/Left Arrow | Moves the focus to the legend left side from the selected legend. |
- | Up/Right Arrow | Moves the focus to the legend right side from the selected legend. |
- | Enter/Space | Toggles the visibility of the corresponding series. |
- | Ctrl + P | Prints the Accumulation chart. |

### Ensuring accessibility

The Accumulation chart component's accessibility levels are ensured through an [accessibility-checker](#) and [axe-core](#) software tools during automated testing.

The accessibility compliance of the Accumulation chart component is shown in the following sample. Open the [sample](#) in a new window to evaluate the accessibility of the Accumulation chart component with accessibility tools.

```
{% previewsample "https://ej2.syncfusion.com/accessibility/accumulation-chart.html" %}
```

See also

- [Accessibility in Syncfusion Vue components](#)

## AppBar

### Getting Started with the Vue AppBar Component in Vue 2

This article provides a step-by-step guide for setting up a Vue 2 project using [Vue-CLI](#) and integrating the Syncfusion Vue AppBar component using the [Composition API](#) / [Options API](#).

### Prerequisites

[System requirements for Syncfusion Vue UI components](#)

### Dependencies

The following list of dependencies are required to use the AppBar component in your application.

```
`js
|-- @syncfusion/ej2-vue-navigations
|-- @syncfusion/ej2-vue-base
|-- @syncfusion/ej2-navigations
|-- @syncfusion/ej2-base
`,`
```

### Setting up the Vue 2 project

To generate a Vue 2 project using Vue-CLI, use the [vue create](#) command. Follow these steps to install Vue CLI and create a new project:

```
`bash
npm install -g @vue/cli
vue create quickstart
cd quickstart
```

```
npm run serve
```

```
,
```

```
or
```

```
`bash
```

```
yarn global add @vue/cli
```

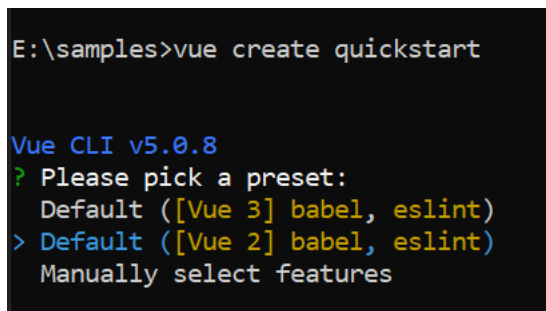
```
vue create quickstart
```

```
cd quickstart
```

```
yarn run serve
```

```
,
```

When creating a new project, choose the option `Default ([Vue 2] babel, eslint)` from the menu.



```
E:\samples>vue create quickstart

Vue CLI v5.0.8
? Please pick a preset:
  Default ([Vue 3] babel, eslint)
> Default ([Vue 2] babel, eslint)
  Manually select features
```

Once the `quickstart` project is set up with default settings, proceed to add Syncfusion components to the project.

#### Add Syncfusion Vue packages

Syncfusion packages are available at [npmjs.com](https://www.npmjs.com). To use Vue components, install the required npm package.

This article uses the [Vue Appbar component](#) as an example. Install the `@syncfusion/ej2-vue-navigations` package by running the following command:

```
`bash
```

```
npm install @syncfusion/ej2-vue-navigations --save
```

```
,
```

```
or
```

```
`bash
```

```
yarn add @syncfusion/ej2-vue-navigations
```

```
,
```

This example uses the [Vue Button component](#) inside the [Vue Appbar component](#). Install the `@syncfusion/ej2-vue-buttons` package by running the following command:

```
`bash
```

```
npm install @syncfusion/ej2-vue-buttons --save
```

`

or

`bash

yarn add @syncfusion/ej2-vue-buttons

`

### Import Syncfusion CSS styles

You can import themes for the Syncfusion Vue component in various ways, such as using CSS or SASS styles from npm packages, CDN, [CRG](#) and [Theme Studio](#). Refer to [themes topic](#) to know more about built-in themes and different ways to refer to themes in a Vue project.

In this article, the **Material** theme is applied using CSS styles, which are available in installed packages. The necessary **Material** CSS styles for the Appbar component and its dependents were imported into the `<style>` section of **src/App.vue** file.

#### ~/SRC/APP.VUE

```
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-buttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-navigations/styles/material.css";
</style>
```

### Add Syncfusion Vue component

Follow the below steps to add the Vue Appbar component using **Composition API** or **Options API**:

1\ First, import and register the Appbar component in the **script** section of the **src/App.vue** file. If you are using the **Composition API**, you should add the **setup** attribute to the **script** tag to indicate that Vue will be using the **Composition API**.

#### COMPOSITION API (~/SRC/APP.VUE)

```
<script setup>
import { AppBarComponent as EjsAppbar } from '@syncfusion/ej2-vue-navigations';
import { ButtonComponent as EjsButton } from '@syncfusion/ej2-vue-buttons';
</script>
```

#### OPTIONS API (~/SRC/APP.VUE)

```
<script>
import { AppBarComponent } from "@syncfusion/ej2-vue-navigations";
import { ButtonComponent } from "@syncfusion/ej2-vue-buttons";
export default {
  components: {
    'ejs-appbar': AppBarComponent,
    'ejs-button': ButtonComponent
  }
}
</script>
```

2.\ In the `template` section, define the AppBar component with the `cssClass` property.

### **(~/SRC/APP.VUE)**

```
<template>
<ejs-appbar colorMode="Primary">
  <ejs-button cssClass="e-inherit" iconCss="e-icons e-menu"></ejs-button>
  <span class="regular" style="margin:0 5px">Vue AppBar</span>
  <div class="e-appbar-spacer"></div>
  <ejs-button cssClass="e-inherit">FREE TRIAL</ejs-button>
</ejs-appbar>
</template>
```

Here is the summarized code for the above steps in the `src/App.vue` file:

### **COMPOSITION API (~/SRC/APP.VUE)**

```
<template>
  <ejs-appbar colorMode="Primary">
    <ejs-button cssClass="e-inherit" iconCss="e-icons e-menu"></ejs-button>
    <span class="regular" style="margin:0 5px">Vue AppBar</span>
    <div class="e-appbar-spacer"></div>
    <ejs-button cssClass="e-inherit">FREE TRIAL</ejs-button>
  </ejs-appbar>
</template>
<script setup>
import { AppBarComponent as EjsAppBar } from '@syncfusion/ej2-vue-
navigations';
import { ButtonComponent as EjsButton } from '@syncfusion/ej2-vue-buttons';
</script>
<style>
  @import "../node_modules/@syncfusion/ej2-base/styles/material.css";
  @import "../node_modules/@syncfusion/ej2-buttons/styles/material.css";
  @import "../node_modules/@syncfusion/ej2-navigations/styles/material.css";
</style>
```

### **OPTIONS API (~/SRC/APP.VUE)**

```
<template>
  <ejs-appbar colorMode="Primary">
    <ejs-button cssClass="e-inherit" iconCss="e-icons e-menu"></ejs-button>
    <span class="regular" style="margin:0 5px">Vue AppBar</span>
    <div class="e-appbar-spacer"></div>
    <ejs-button cssClass="e-inherit">FREE TRIAL</ejs-button>
  </ejs-appbar>
</template>
<script>
import { AppBarComponent } from "@syncfusion/ej2-vue-navigations";
import { ButtonComponent } from "@syncfusion/ej2-vue-buttons";
export default {
  components: {
    'ejs-appbar': AppBarComponent,
    'ejs-button': ButtonComponent
  },
  data: function () {
    return {};
  }
};
```

```
    },  
  };  
</script>  
<style>  
  @import "../node_modules/@syncfusion/ej2-base/styles/material.css";  
  @import "../node_modules/@syncfusion/ej2-buttons/styles/material.css";  
  @import "../node_modules/@syncfusion/ej2-navigations/styles/material.css";  
</style>
```

### Run the project

To run the project, use the following command:

```
`bash
```

```
npm run serve
```

```
,
```

or

```
`bash
```

```
yarn run serve
```

```
,
```

```
{% previewsample "page.domainurl/code-snippet/appbar/getting-started-cs5" %}
```

### Getting Started with Vue AppBar Component in Vue 3

This article provides a step-by-step guide for setting up a [Vite](#) project with a JavaScript environment and integrating the Syncfusion Vue AppBar component using the [Composition API](#) / [Options API](#).

The **Composition API** is a new feature introduced in Vue.js 3 that provides an alternative way to organize and reuse component logic. It allows developers to write components as functions that use smaller, reusable functions called composition functions to manage their properties and behavior.

The **Options API** is the traditional way of writing Vue.js components, where the component logic is organized into a series of options that define the component's properties and behavior. These options include data, methods, computed properties, watchers, lifecycle hooks, and more.

### Prerequisites

#### [System requirements for Syncfusion Vue UI components](#)

#### Setup the Vite project

A recommended approach for beginning with Vue is to scaffold a project using [Vite](#). To create a new Vite project, use one of the commands that are specific to either NPM or Yarn.

```
`bash
```

```
npm create vite@latest
```

```
,
```

or

```
`bash
```

```
yarn create vite
```

`

Using one of the above commands will lead you to set up additional configurations for the project as below:

1. Define the project name: We can specify the name of the project directly. Let's specify the name of the project as `my-project` for this article.

```
`bash
```

```
? Project name: » my-project
```

`

2. Select `Vue` as the framework. It will create a Vue 3 project.

```
`bash
```

```
? Select a framework: » - Use arrow-keys. Return to submit.
```

```
Vanilla
```

```
Vue
```

```
React
```

```
Preact
```

```
Lit
```

```
Svelte
```

```
Others
```

`

3. Choose `JavaScript` as framework variant to build this Vite project using JavaScript and Vue.

```
`bash
```

```
? Select a variant: » - Use arrow-keys. Return to submit.
```

```
JavaScript
```

```
TypeScript
```

```
Customize with create-vue ↗
```

```
Nuxt ↗
```

`

4. Upon completing the aforementioned steps to create the `my-project`, run the following command to install its dependencies:

```
`bash
```

```
cd my-project
```

```
npm install
```

`

or



```
`bash
cd my-project
yarn install
`
```

Now that `my-project` is ready to run with default settings, let's add Syncfusion Vue AppBar component to the project.

#### Add Syncfusion Vue packages

Syncfusion Vue component packages are available at [npmjs.com](https://www.npmjs.com). To use Syncfusion Vue components in the project, install the corresponding npm package.

This article uses the [Vue AppBar component](#) as an example. To use the `Vue AppBar` component in the project, the `@syncfusion/ej2-vue-navigations` package needs to be installed using the following command

```
`bash
npm install @syncfusion/ej2-vue-navigations --save
`
```

or

```
`bash
yarn add @syncfusion/ej2-vue-navigations
`
```

#### Import Syncfusion CSS styles

You can import themes for the Syncfusion Vue component in various ways, such as using CSS or SASS styles from npm packages, CDN, [CRG](#) and [Theme Studio](#). Refer to [themes topic](#) to know more about built-in themes and different ways to refer to themes in a Vue project.

In this article, `Material` theme is applied using CSS styles, which are available in installed packages. The necessary `Material` CSS styles for the AppBar component and its dependents were imported into the `<style>` section of the `src/App.vue` file.

```
`html
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-navigations/styles/material.css";
</style>
`
```

#### Adding Syncfusion Vue AppBar component in the application

Follow the below steps to add the Vue AppBar component using `Composition API` or `Options API`:

1.First, import and register the AppBar component and its child directives in the `script` section of the `src/App.vue` file. If you are using the `Composition API`, you should add the `setup` attribute to the `script` tag to indicate that Vue will be using the `Composition API`.

#### **COMPOSITION API (~SRC/APP.VUE)**

```
<script setup>
import { AppBarComponent as EjsAppBar } from '@syncfusion/ej2-vue-
navigations';
</script>
```

#### **OPTIONS API (~SRC/APP.VUE)**

```
<script>
import { AppBarComponent } from "@syncfusion/ej2-vue-navigations";
export default {
  name: "App",
  components: {
    "ejs-appbar": AppBarComponent,
  }
}
</script>
```

2.Add the component definition in template section.

```
`html
<template>
<ejs-appbar colorMode="Primary">
<ejs-button cssClass="e-inherit" iconCss="e-icons e-menu"></ejs-button>
<span class="regular" style="margin:0 5px">Vue AppBar</span>
<div class="e-appbar-spacer"></div>
<ejs-button cssClass="e-inherit">FREE TRIAL</ejs-button>
</ejs-appbar>
</template>
`
```

Here is the summarized code for the above steps in the `src/App.vue` file:

#### **COMPOSITION API (~SRC/APP.VUE)**

```
<template>
<ejs-appbar colorMode="Primary">
<ejs-button cssClass="e-inherit" iconCss="e-icons e-menu"></ejs-button>
<span class="regular" style="margin:0 5px">Vue AppBar</span>
<div class="e-appbar-spacer"></div>
<ejs-button cssClass="e-inherit">FREE TRIAL</ejs-button>
</ejs-appbar>
</template>
<script setup>
```

```
import { AppBarComponent as EjsAppBar } from '@syncfusion/ej2-vue-
navigations';
import { ButtonComponent as EjsButton } from "@syncfusion/ej2-vue-buttons";
</script>
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-buttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-
navigations/styles/material.css";
</style>
```

### OPTIONS API (~SRC/APP.VUE)

```
<template>
<ejs-appbar colorMode="Primary">
<ejs-button cssClass="e-inherit" iconCss="e-icons e-menu"></ejs-button>
<span class="regular" style="margin:0 5px">Vue AppBar</span>
<div class="e-appbar-spacer"></div>
<ejs-button cssClass="e-inherit">FREE TRIAL</ejs-button>
</ejs-appbar>
</template>
<script>
import { AppBarComponent } from '@syncfusion/ej2-vue-navigations';
export default {
name: "App",
components: {
"ejs-appbar": AppBarComponent
}
}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-buttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-
navigations/styles/material.css";
</style>
```

### Run the project

To run the project, use the following command:

```
`bash
```

```
npm run dev
```

```
,
```

or

```
`bash
```

```
yarn run dev
```

```
,
```

The output will appear as follows:



Refer the sample [Vue 3 using Composition API AppBar getting started](#)

For migrating from Vue 2 to Vue 3, refer to the [migration](#) documentation.

See also

- [Getting Started with Vue UI Components using Composition API and TypeScript](#)
- [Getting Started with Vue UI Components using Options API and TypeScript](#)

## Size and color in Vue AppBar component

### Size

The size of the AppBar can be set using the [mode](#) property. The available types of the AppBar are as follows:

- Regular AppBar
- Prominent AppBar
- Dense AppBar

### Regular AppBar

This mode is the default one in which the AppBar is displayed with the default height.

### APP.VUE

```
<template>
  <div class="control-container">
    <ejs-appbar colorMode="Primary">
      <ejs-button cssClass="e-inherit" iconCss="e-icons e-menu"></ejs-
button>
      <span class="regular">Regular AppBar</span>
      <div class="e-appbar-spacer"></div>
      <ejs-button cssClass="e-inherit">FREE TRIAL</ejs-button>
    </ejs-appbar>
  </div>
</template>
<script>
import Vue from "vue";
import { AppBarPlugin } from "@syncfusion/ej2-vue-navigations";
import { ButtonPlugin } from "@syncfusion/ej2-vue-buttons";
Vue.use(AppBarPlugin);
Vue.use(ButtonPlugin);
export default {
  data: function () {
    return {};
  }
};
</script>
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-buttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-navigations/styles/material.css";
.control-container .e-btn.e-inherit {
  margin: 0 3px;
}
</style>
```

```
{% previewsample "page.domainurl/code-snippet/appbar/getting-started-cs9" %}
```

### Prominent AppBar

This height mode can be set to the AppBar by setting **Prominent** to the property [mode](#). The prominent AppBar is displayed with a longer height and can be used for larger titles, images, or texts. It is also longer than the regular AppBar. In the following example, we have customized the prominent text using align-self and white-space CSS properties. You can change the prominent AppBar height if larger titles, images, or texts are used.

### APP.VUE

```
<template>
  <div class="control-container">
    <ejs-appbar colorMode="Primary" mode="Prominent" cssClass="prominent-
    appbar">
      <ejs-button cssClass="e-inherit" iconCss="e-icons e-menu"></ejs-
    button>
      <span class="prominent">AppBar Component with Prominent mode</span>
      <div class="e-appbar-spacer"></div>
      <ejs-button cssClass="e-inherit">FREE TRIAL</ejs-button>
    </ejs-appbar>
  </div>
</template>
<script>
import Vue from "vue";
import { AppBarPlugin } from "@syncfusion/ej2-vue-navigations";
import { ButtonPlugin } from "@syncfusion/ej2-vue-buttons";
Vue.use(AppBarPlugin);
Vue.use(ButtonPlugin);
export default {
  data: function () {
    return {};
  }
};
</script>
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-buttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-navigations/styles/material.css";
.prominent-appbar .prominent {
  align-self: center;
  white-space: break-spaces;
  text-align: inherit;
  font-size: 35px;
  line-height: 50px;
}
.prominent-appbar.e-appbar {
  background-image:
url("https://ej2.syncfusion.com/demos/src/appbar/images/prominent.png");
  background-size: 100% 350px;
  color: #ffffff;
  background-repeat: no-repeat;
  height: 350px;
}
.prominent-appbar .e-inherit.e-btn {
```

```

    background: transparent;
  }
  .prominent-appbar .e-inherit.e-btn:hover,
  .prominent-appbar .e-inherit.e-btn:focus,
  .prominent-appbar .e-inherit.e-btn:active,
  .prominent-appbar .e-inherit.e-btn.e-active,
  .prominent-appbar .e-inherit.e-css.e-btn:hover,
  .prominent-appbar .e-inherit.e-css.e-btn:focus
  .prominent-appbar .e-inherit.e-css.e-btn:active
  .prominent-appbar .e-inherit.e-css.e-btn.e-active {
    background: rgba(255, 255, 255, .08);
  }
</style>

```

{% previewsample "page.domainurl/code-snippet/appbar/getting-started-cs10" %}

### Dense AppBar

This height mode can be set to the AppBar by setting **Dense** to the property [mode](#). Dense AppBar is displayed with shorter height which is denser to accommodate all the AppBar content.

### APP.VUE

```

<template>
  <div class="control-container">
    <ejs-appbar colorMode="Primary" mode="Dense">
      <ejs-button cssClass="e-inherit" iconCss="e-icons e-menu"></ejs-
button>
      <span class="dense">Dense AppBar</span>
      <div class="e-appbar-spacer"></div>
      <ejs-button cssClass="e-inherit">FREE TRIAL</ejs-button>
    </ejs-appbar>
  </div>
</template>
<script>
import Vue from "vue";
import { AppBarPlugin } from "@syncfusion/ej2-vue-navigations";
import { ButtonPlugin } from "@syncfusion/ej2-vue-buttons";
Vue.use(AppBarPlugin);
Vue.use(ButtonPlugin);
export default {
  data: function () {
    return {};
  }
};
</script>
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-buttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-navigations/styles/material.css";
.control-container .e-btn.e-inherit {
  margin: 0 3px;
}
</style>

```

{% previewsample "page.domainurl/code-snippet/appbar/getting-started-cs11" %}

## Color

The background and font colors can be set using the [colorMode](#) property. The available types of background color for the AppBar are as follows:

- Light AppBar
- Dark AppBar
- Primary AppBar
- Inherit AppBar

### Light AppBar

This color mode is the default one in which the AppBar can be displayed with a light background and its corresponding font color.

### APP.VUE

```
<template>
  <div class="control-container">
    <ejs-appbar>
      <a href="https://www.syncfusion.com/vue-components" target="_blank"
rel="noopener" role="link" aria-label="Syncfusion vue controls">
        <div class="syncfusion-logo"></div>
      </a>
      <div class="e-appbar-spacer"></div>
      <ejs-button :isPrimary="true">FREE TRIAL</ejs-button>
    </ejs-appbar>
  </div>
</template>
<script>
import Vue from "vue";
import { AppBarPlugin } from "@syncfusion/ej2-vue-navigations";
import { ButtonPlugin } from "@syncfusion/ej2-vue-buttons";
Vue.use(AppBarPlugin);
Vue.use(ButtonPlugin);
export default {
  data: function () {
    return {};
  }
};
</script>
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-buttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-navigations/styles/material.css";
.control-container .syncfusion-logo {
  background:
url(https://cdn.syncfusion.com/blazor/images/demos/syncfusion-logo.svg);
  background-size: contain;
  background-repeat: no-repeat;
  height: 30px;
  width: 150px;
}
</style>
```

{% previewsample "page.domainurl/code-snippet/appbar/getting-started-cs12" %}

### Dark AppBar

This color mode can be set to the AppBar by setting **Dark** to the property [colorMode](#). A dark AppBar can be displayed with a dark background and its corresponding font color.

#### APP.VUE

```
<template>
  <div class="control-container">
    <ejs-appbar colorMode="Dark">
      <ejs-button cssClass="e-inherit" iconCss="e-icons e-menu"></ejs-
button>
      <div class="e-appbar-spacer"></div>
      <ejs-button cssClass="e-inherit">FREE TRIAL</ejs-button>
    </ejs-appbar>
  </div>
</template>
<script>
  import Vue from "vue";
  import { AppBarPlugin } from "@syncfusion/ej2-vue-navigations";
  import { ButtonPlugin } from "@syncfusion/ej2-vue-buttons";
  Vue.use(AppBarPlugin);
  Vue.use(ButtonPlugin);
  export default {
    data: function () {
      return {};
    }
  };
</script>
<style>
  @import "../node_modules/@syncfusion/ej2-base/styles/material.css";
  @import "../node_modules/@syncfusion/ej2-buttons/styles/material.css";
  @import "../node_modules/@syncfusion/ej2-navigations/styles/material.css";
  .control-container .e-btn.e-inherit {
    margin: 0 3px;
  }
</style>
```

{% previewsample "page.domainurl/code-snippet/appbar/getting-started-cs13" %}

### Primary AppBar

This color mode can be set to the AppBar by setting **Primary** to the property [colorMode](#). The primary AppBar can be displayed with primary colors.

#### APP.VUE

```
<template>
  <div class="control-container">
    <ejs-appbar colorMode="Primary">
      <ejs-button cssClass="e-inherit" iconCss="e-icons e-menu"></ejs-
button>
      <div class="e-appbar-spacer"></div>
      <ejs-button cssClass="e-inherit">FREE TRIAL</ejs-button>
    </ejs-appbar>
  </div>
</template>
<script>
```



```

import Vue from "vue";
import { AppBarPlugin } from "@syncfusion/ej2-vue-navigations";
import { ButtonPlugin } from "@syncfusion/ej2-vue-buttons";
Vue.use(AppBarPlugin);
Vue.use(ButtonPlugin);
export default {
  data: function () {
    return {};
  }
};
</script>
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-buttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-navigations/styles/material.css";
.control-container .e-btn.e-inherit {
  margin: 0 3px;
}
</style>

```

{% previewsample "page.domainurl/code-snippet/appbar/getting-started-cs14" %}

### *Inherit AppBar*

This color mode can be set to the AppBar by setting **Inherit** to the property [colorMode](#). The AppBar inherits the background and font color from its parent element.

### **APP.VUE**

```

<template>
  <div class="control-container">
    <ejs-appbar colorMode="Inherit">
      <a href="https://www.syncfusion.com/vue-components" target="_blank"
        rel="noopener" role="link" aria-label="Syncfusion vue controls">
        <div class="syncfusion-logo"></div>
      </a>
      <div class="e-appbar-spacer"></div>
      <ejs-button :isPrimary="true">FREE TRIAL</ejs-button>
    </ejs-appbar>
  </div>
</template>
<script>
import Vue from "vue";
import { AppBarPlugin } from "@syncfusion/ej2-vue-navigations";
import { ButtonPlugin } from "@syncfusion/ej2-vue-buttons";
Vue.use(AppBarPlugin);
Vue.use(ButtonPlugin);
export default {
  data: function () {
    return {};
  }
};
</script>
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-buttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-navigations/styles/material.css";

```

```
.control-container .syncfusion-logo {
  background:
url (https://cdn.syncfusion.com/blazor/images/demos/syncfusion-logo.svg);
  background-size: contain;
  background-repeat: no-repeat;
  height: 30px;
  width: 150px;
}
</style>
```

{% previewsample "page.domainurl/code-snippet/appbar/getting-started-cs15" %}

### Position in Vue AppBar component

The position of the AppBar can be set using the [position](#) and [isSticky](#) property. The AppBar provides the following options for setting its position:

- Top AppBar
- Bottom AppBar
- Sticky AppBar

### Top AppBar

The top AppBar is the default one in which it positions the AppBar at the top of the content.

### APP.VUE

```
<template>
  <div class="control-container">
    <ejs-appbar colorMode="Primary">
      <ejs-button cssClass="e-inherit" iconCss="e-icons e-menu"></ejs-
button>
      <div class="e-appbar-spacer"></div>
      <ejs-button cssClass="e-inherit">FREE TRIAL</ejs-button>
    </ejs-appbar>
    <div class="appbar-content" style="font-size: 12px">
      <p>
        The AppBar also known as action bar or nav bar displays
        information and actions related to the current application screen. It is
        used to show branding, screen titles, navigation, and actions. The control
        supports height mode, color mode, positioning, and more.
      </p>
      <p>
        The AppBar control provides flexible ways to configure the look
        and feel of the bar to match your requirement.
      </p>
      <p>
        Developers can control the appearance and behaviors of the
        AppBar using a rich set of APIs.
      </p>
      <p>
        The AppBar component supports built-in themes such as Material,
        Bootstrap, Fabric (Office 365), Tailwind CSS, and high contrast. Users can
        customize these built-in themes or create new themes to achieve their
        desired look and feel by either simply overriding SASS variables or using
our Theme Studio application.
      </p>
    </div>
  </div>
```

```

    </div>
  </div>
</template>
<script>
  import Vue from "vue";
  import { AppBarPlugin } from "@syncfusion/ej2-vue-navigations";
  import { ButtonPlugin } from "@syncfusion/ej2-vue-buttons";
  Vue.use(AppBarPlugin);
  Vue.use(ButtonPlugin);
  export default {
    data: function () {
      return {};
    }
  };
</script>
<style>
  @import "../node_modules/@syncfusion/ej2-base/styles/material.css";
  @import "../node_modules/@syncfusion/ej2-buttons/styles/material.css";
  @import "../node_modules/@syncfusion/ej2-navigations/styles/material.css";
  .control-container {
    height: 220px;
    margin: 0 auto;
    width: 500px;
    overflow-y: scroll;
  }
  .control-container .e-btn.e-inherit {
    margin: 0 3px;
  }
</style>

```

{% previewsample "page.domainurl/code-snippet/appbar/getting-started-cs6" %}

### Bottom AppBar

This position can be set to the AppBar by setting **Bottom** to the property [position](#). The bottom AppBar positions the AppBar at the bottom of the content.

### APP.VUE

```

<template>
  <div class="control-container">
    <ejs-appbar colorMode="Primary" position="Bottom">
      <ejs-button cssClass="e-inherit" iconCss="e-icons e-menu"></ejs-
button>
      <div class="e-appbar-spacer"></div>
      <ejs-button cssClass="e-inherit">FREE TRIAL</ejs-button>
    </ejs-appbar>
    <div class="appbar-content" style="font-size: 12px">
      <p>
        The AppBar also known as action bar or nav bar displays
        information and actions related to the current application screen. It is
        used to show branding, screen titles, navigation, and actions. The control
        supports height mode, color mode, positioning, and more.
      </p>
      <p>
        The AppBar control provides flexible ways to configure the look
        and feel of the bar to match your requirement.
      </p>
    </div>
  </div>

```

```

    </p>
    <p>
        Developers can control the appearance and behaviors of the
        AppBar using a rich set of APIs.
    </p>
    <p>
        The AppBar component supports built-in themes such as Material,
        Bootstrap, Fabric (Office 365), Tailwind CSS, and high contrast. Users can
        customize these built-in themes or create new themes to achieve their
        desired look and feel by either simply overriding SASS variables or using
        our Theme Studio application.
    </p>
</div>
</div>
</template>
<script>
    import Vue from "vue";
    import { AppBarPlugin } from "@syncfusion/ej2-vue-navigations";
    import { ButtonPlugin } from "@syncfusion/ej2-vue-buttons";
    Vue.use(AppBarPlugin);
    Vue.use(ButtonPlugin);
    export default {
        data: function () {
            return {};
        }
    };
</script>
<style>
    @import "../node_modules/@syncfusion/ej2-base/styles/material.css";
    @import "../node_modules/@syncfusion/ej2-buttons/styles/material.css";
    @import "../node_modules/@syncfusion/ej2-navigations/styles/material.css";
    .control-container {
        height: 300px;
        width: 500px;
        margin: 0 auto;
        position: relative;
    }
    .control-container .e-btn.e-inherit {
        margin: 0 3px;
    }
</style>

```

{% previewsample "page.domainurl/code-snippet/appbar/getting-started-cs7" %}

### Sticky AppBar

This position can be set to the AppBar by setting `true` to the property `isSticky`. AppBar will be sticky while scrolling the AppBar content.

### APP.VUE

```

<template>
    <div class="control-container">
        <ejs-appbar colorMode="Primary" :isSticky="true">
            <ejs-button cssClass="e-inherit" iconCss="e-icons e-menu"></ejs-
            button>
            <div class="e-appbar-spacer"></div>

```

```

    <ejs-button cssClass="e-inherit">FREE TRIAL</ejs-button>
  </ejs-appbar>
  <div class="appbar-content" style="font-size: 12px">
    <p>
      The AppBar also known as action bar or nav bar displays
      information and actions related to the current application screen. It is
      used to show branding, screen titles, navigation, and actions. The control
      supports height mode, color mode, positioning, and more.
    </p>
    <p>
      The AppBar control provides flexible ways to configure the look
      and feel of the bar to match your requirement.
    </p>
    <p>
      Developers can control the appearance and behaviors of the
      AppBar using a rich set of APIs.
    </p>
    <p>
      The AppBar component supports built-in themes such as Material,
      Bootstrap, Fabric (Office 365), Tailwind CSS, and high contrast. Users can
      customize these built-in themes or create new themes to achieve their
      desired look and feel by either simply overriding SASS variables or using
our Theme Studio application.
    </p>
  </div>
</div>
</template>
<script>
  import Vue from "vue";
  import { AppBarPlugin } from "@syncfusion/ej2-vue-navigations";
  import { ButtonPlugin } from "@syncfusion/ej2-vue-buttons";
  Vue.use(AppBarPlugin);
  Vue.use(ButtonPlugin);
  export default {
    data: function () {
      return {};
    }
  };
</script>
<style>
  @import "../node_modules/@syncfusion/ej2-base/styles/material.css";
  @import "../node_modules/@syncfusion/ej2-buttons/styles/material.css";
  @import "../node_modules/@syncfusion/ej2-navigations/styles/material.css";
  .control-container {
    height: 220px;
    margin: 0 auto;
    width: 500px;
    overflow-y: scroll;
  }
  .control-container .e-btn.e-inherit {
    margin: 0 3px;
  }
</style>

```

```
{% previewsample "page.domainurl/code-snippet/appbar/getting-started-cs8" %}
```

## Design in Vue AppBar component

### Spacer

**Spacer** is used to provide spacing between the AppBar contents which gives additional space to the content layout.

The following example depicts the code to provide spacing between the home and pan buttons in the AppBar:

#### APP.VUE

```
<template>
  <div class="control-container">
    <ejs-appbar colorMode="Primary">
      <ejs-button cssClass="e-inherit" iconCss="e-icons e-home"></ejs-
button>
      <div class="e-appbar-spacer"></div>
      <ejs-button cssClass="e-inherit" iconCss="e-icons e-cut"></ejs-button>
      <div class="e-appbar-spacer"></div>
      <ejs-button cssClass="e-inherit" iconCss="e-icons e-pan"></ejs-button>
    </ejs-appbar>
  </div>
</template>
<script>
import Vue from "vue";
import { AppBarPlugin } from "@syncfusion/ej2-vue-navigations";
import { ButtonPlugin } from "@syncfusion/ej2-vue-buttons";
Vue.use(AppBarPlugin);
Vue.use(ButtonPlugin);
export default {
  data: function () {
    return {};
  }
};
</script>
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-buttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-navigations/styles/material.css";
</style>
```

{% previewsample "page.domainurl/code-snippet/appbar/getting-started-cs1" %}

### Separator

**Separator** shows a vertical line to visually group or separate the AppBar contents.

The following example depicts the code to provide a vertical line between a group of buttons in the AppBar.

#### APP.VUE

```
<template>
  <div class="control-container">
    <ejs-appbar colorMode="Primary">
      <ejs-button cssClass="e-inherit" iconCss="e-icons e-cut"></ejs-button>
      <ejs-button cssClass="e-inherit" iconCss="e-icons e-copy"></ejs-
button>
```

```

        <ejs-button cssClass="e-inherit" iconCss="e-icons e-paste"></ejs-
button>
        <div class="e-appbar-separator"></div>
        <ejs-button cssClass="e-inherit" iconCss="e-icons e-bold"></ejs-
button>
        <ejs-button cssClass="e-inherit" iconCss="e-icons e-underline"></ejs-
button>
        <ejs-button cssClass="e-inherit" iconCss="e-icons e-italic"></ejs-
button>
        <div class="e-appbar-separator"></div>
        <ejs-button cssClass="e-inherit" iconCss="e-icons e-align-left"></ejs-
button>
        <ejs-button cssClass="e-inherit" iconCss="e-icons e-align-
right"></ejs-button>
        <ejs-button cssClass="e-inherit" iconCss="e-icons e-align-
center"></ejs-button>
        <ejs-button cssClass="e-inherit" iconCss="e-icons e-justify"></ejs-
button>
    </ejs-appbar>
</div>
</template>
<script>
    import Vue from "vue";
    import { AppBarPlugin } from "@syncfusion/ej2-vue-navigations";
    import { ButtonPlugin } from "@syncfusion/ej2-vue-buttons";
    Vue.use(AppBarPlugin);
    Vue.use(ButtonPlugin);
    export default {
        data: function () {
            return {};
        }
    };
</script>
<style>
    @import "../node_modules/@syncfusion/ej2-base/styles/material.css";
    @import "../node_modules/@syncfusion/ej2-buttons/styles/material.css";
    @import "../node_modules/@syncfusion/ej2-navigations/styles/material.css";
    .control-container .e-btn.e-inherit {
        margin: 0 3px;
    }
</style>

```

{% previewsample "page.domainurl/code-snippet/appbar/getting-started-cs2" %}

### Media Query

Media Query is used to adjusting the AppBar for different screen sizes. Resize the screen to observe the responsive layout of the AppBar.

### APP.VUE

```

<template>
    <div class="control-container">
        <ejs-appbar colorMode="Primary">
            <ejs-button cssClass="e-inherit" iconCss="e-icons e-menu"></ejs-
button>
            <ejs-button cssClass="e-inherit">Home</ejs-button>

```

```

    <ejs-button cssClass="e-inherit">About</ejs-button>
    <ejs-button cssClass="e-inherit">Products</ejs-button>
    <ejs-button cssClass="e-inherit">Contacts</ejs-button>
    <div class="e-appbar-spacer"></div>
    <div class="e-appbar-separator"></div>
    <ejs-button cssClass="e-inherit">Login</ejs-button>
  </ejs-appbar>
</div>
</template>
<script>
import Vue from "vue";
import { AppBarPlugin } from "@syncfusion/ej2-vue-navigations";
import { ButtonPlugin } from "@syncfusion/ej2-vue-buttons";
Vue.use(AppBarPlugin);
Vue.use(ButtonPlugin);
export default {
  data: function () {
    return {};
  }
};
</script>
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-buttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-navigations/styles/material.css";
.control-container .e-btn.e-inherit {
  margin: 0 3px;
}
@media screen and (max-width: 1024px) {
  .control-container .e-appbar {
    flex-flow: row wrap;
    height: auto;
    gap: 8px;
  }
  .control-container {
    width: 350px;
  }
}
@media screen and (max-width: 480px) {
  .control-container {
    width: 200px;
    margin: 0 2px;
  }
}
</style>

```

{% previewsample "page.domainurl/code-snippet/appbar/getting-started-cs3" %}

### Designing AppBar with Menu

AppBar is rendered with a Menu component in its AppBar header area. Menu component's styles are inherited from the AppBar component using the `e-inherit` CSS class.

### APP.VUE

```

<template>
  <div class="control-container">

```



```

    <ejs-appbar colorMode="Primary">
      <ejs-button cssClass="e-inherit" iconCss="e-icons e-menu"></ejs-
button>
      <ejs-menu cssClass="e-inherit" :items="companyMenuItems"></ejs-menu>
      <ejs-menu cssClass="e-inherit" :items="productMenuItems"></ejs-menu>
      <ejs-menu cssClass="e-inherit" :items="aboutMenuItems"></ejs-menu>
      <ejs-menu cssClass="e-inherit" :items="carrerMenuItems"></ejs-menu>
      <div class="e-appbar-spacer"></div>
      <ejs-button cssClass="e-inherit">Login</ejs-button>
    </ejs-appbar>
  </div>
</template>
<script>
  import Vue from "vue";
  import { AppBarPlugin, MenuPlugin } from "@syncfusion/ej2-vue-
navigations";
  import { ButtonPlugin } from "@syncfusion/ej2-vue-buttons";
  Vue.use(AppBarPlugin);
  Vue.use(MenuPlugin);
  Vue.use(ButtonPlugin);
  export default {
    data: function () {
      return {
        companyMenuItems: [
          {
            text : 'Company',
            items: [
              { text: 'About Us' },
              { text: 'Customers' },
              { text: 'Blog' },
              { text: 'Careers' }
            ]
          }
        ],
        productMenuItems: [
          {
            text : 'Products',
            items: [
              { text: 'Developer' },
              { text: 'Analytics' },
              { text: 'Reporting' },
              { text: 'Help Desk' }
            ]
          }
        ],
        aboutMenuItems: [
          {
            text : 'About Us'
          }
        ],
        carrerMenuItems: [
          {
            text : 'Carrers'
          }
        ]
      };
    }
  }

```

```

    };
</script>
<style>
  @import "../node_modules/@syncfusion/ej2-base/styles/material.css";
  @import "../node_modules/@syncfusion/ej2-buttons/styles/material.css";
  @import "../node_modules/@syncfusion/ej2-navigations/styles/material.css";
  .control-container .e-btn.e-inherit {
    margin: 0 3px;
  }
</style>

```

{% previewsample "page.domainurl/code-snippet/appbar/getting-started-cs4" %}

### Designing AppBar with Buttons

The AppBar is rendered with a Button and DropDownButton component in its AppBar header area.

Button and DropDownButton components' styles are inherited from the AppBar component using the `e-inherit` CSS class.

### APP.VUE

```

<template>
  <div class="control-container">
    <ejs-appbar colorMode="Primary">
      <ejs-button cssClass="e-inherit" iconCss="e-icons e-menu"></ejs-button>
      <ejs-dropdownbutton cssClass="e-inherit"
      :items="productDropDownButtonItem">Products</ejs-dropdownbutton>
      <div class="e-appbar-spacer"></div>
      <ejs-button cssClass="e-inherit">Login</ejs-button>
    </ejs-appbar>
  </div>
</template>
<script>
  import Vue from "vue";
  import { AppBarPlugin } from "@syncfusion/ej2-vue-navigations";
  import { DropDownButtonPlugin } from "@syncfusion/ej2-vue-splitbuttons";
  import { ButtonPlugin } from "@syncfusion/ej2-vue-buttons";
  Vue.use(AppBarPlugin);
  Vue.use(DropDownButtonPlugin);
  Vue.use(ButtonPlugin);
  export default {
    data: function () {
      return {
        productDropDownButtonItem: [
          { text: 'Developer' },
          { text: 'Analytics' },
          { text: 'Reporting' },
          { text: 'E-Signature' },
          { text: 'Help Desk' }
        ]
      };
    }
  };
</script>
<style>
  @import "../node_modules/@syncfusion/ej2-base/styles/material.css";

```

```

@import "../node_modules/@syncfusion/ej2-buttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-navigations/styles/material.css";
@import "../node_modules/@syncfusion/ej2-splitbuttons/styles/material.css";
.control-container .e-btn.e-inherit {
    margin: 0 3px;
}
</style>

```

{% previewsample "page.domainurl/code-snippet/appbar/buttons-cs1" %}

### Designing AppBar with SideBar

The AppBar is rendered with the SideBar component below the AppBar. Click on the menu icon to expand/collapse the Sidebar. In the following sample, the `toggle` method has been used to show or hide the Sidebar on the AppBar button click.

#### APP.VUE

```

<template>
  <div id="wrapper" class="control-section">
    <div id="reswrapper">
      <div>
        <ejs-appbar>
          <ejs-button id="button" cssClass="e-inherit" iconCss="e-icons e-menu"></ejs-button>
          <div class="e-folder">
            <div class="e-folder-name">Navigation Pane</div>
          </div>
        </ejs-appbar>
      </div>
      <ejs-sidebar id="sideTree" class="sidebar-treeview"
        ref="sidebarTreeviewInstance" width="220px" target=".main-sidebar-content"
        mediaQuery="(min-width: 600px)" :isOpen="true">
        <div class='main-menu'>
          <div class="table-content">
            <ejs-textbox id="resSearch"
              placeholder="Search..."></ejs-textbox>
            <p class="main-menu-header">TABLE OF CONTENTS</p>
          </div>
          <div>
            <ejs-treeview id='mainTree' cssClass="main-treeview"
              :fields="fields" expandOn='Click'>
            </ejs-treeview>
          </div>
        </div>
      </ejs-sidebar>
      <div class="main-sidebar-content" id="main-text">
        <div class="sidebar-content">
          <div class="sidebar-heading"> Responsive Sidebar with
            Treeview</div>
          <p class="paragraph-content">
            This is a graphical aid for visualising and categorising
            the site, in the style of an expandable and collapsable treeview component.
            It auto-expands to display the node(s), if any,
            corresponding to the currently viewed title, highlighting that node(s)

```

and its ancestors. Load-on-demand when expanding nodes is available where supported (most graphical browsers), falling back to a full-page reload. MediaWiki-supported caching, aside from squid, has been considered so that unnecessary re-downloads of content are avoided where possible. The complete expanded/collapsed state of the treeview persists across page views in most situations.

```

        </p>
      </div>
    </div>
  </div>
</template>
<script>
  import Vue from "vue";
  import { AppBarPlugin, SidebarPlugin, TreeViewPlugin } from
"@syncfusion/ej2-vue-navigations";
  import { TextBoxPlugin } from '@syncfusion/ej2-vue-inputs';
  import { ButtonPlugin } from "@syncfusion/ej2-vue-buttons";
  Vue.use(AppBarPlugin);
  Vue.use(SidebarPlugin);
  Vue.use(TreeViewPlugin);
  Vue.use(TextBoxPlugin);
  Vue.use(ButtonPlugin);
  export default {
    mounted () {
      var button = document.getElementById('button');
      button.addEventListener('click', toggle);
      function toggle() {
        let sidebar = document.getElementById("sideTree").ej2_instances[0];
        sidebar.toggle();
      }
    },
    data () {
      var dataSource = [
        {
          nodeId: '01', nodeText: 'Installation',
        },
        {
          nodeId: '02', nodeText: 'Deployment',
        },
        {
          nodeId: '03', nodeText: 'Quick Start',
        },
        {
          nodeId: '04', nodeText: 'Components',
          nodeChild: [
            { nodeId: '04-01', nodeText: 'Calendar' },
            { nodeId: '04-02', nodeText: 'DatePicker' },
            { nodeId: '04-03', nodeText: 'DateTimePicker' },
            { nodeId: '04-04', nodeText: 'DateRangePicker' },
            { nodeId: '04-05', nodeText: 'TimePicker' },
            { nodeId: '04-06', nodeText: 'SideBar' }
          ]
        }
      ]
    }
  };

```

```

        return {
            fields: { dataSource: dataSource, id: 'nodeId', text: 'nodeText',
child: 'nodeChild' }
        }
    }
}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-buttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-navigations/styles/material.css";
@import "../node_modules/@syncfusion/ej2-inputs/styles/material.css";
.e-appbar .e-folder {
    margin: 0 5px;
}
#wrapper.control-section {
    padding: 15px 0px;
}
#wrapper .sidebar-treeview {
    z-index: 20 !important;
}
#wrapper .main-sidebar-content {
    height: 380px;
}
.sidebar-treeview .main-menu .main-menu-header {
    color: #656a70;
    padding: 15px 15px 15px 0;
    font-size: 14px;
    width: 13em;
    margin: 0;
}
#main-text .sidebar-heading {
    font-size: 16px;
}
.sidebar-treeview .table-content {
    padding: 20px 14px;
    height: 8em;
}
#main-text .sidebar-content .line {
    width: 100%;
    height: 1px;
    border-bottom: 1px dashed #ddd;
    margin: 40px 0;
}
#main-text .sidebar-content {
    padding: 15px;
    font-size: 14px;
}
#main-text .paragraph-content {
    padding: 15px 0;
    font-weight: normal;
    font-size: 14px;
}
.e-folder {
    text-align: center;
    font-weight: 500;
    font-size: 16px

```

```

}
.e-folder-name {
  margin-top: 1px;
}
.sidebar-treeview .e-treeview .e-icon-collapsible,
.sidebar-treeview .e-treeview .e-icon-expandable {
  float: right;
  margin: 3px;
}
.sidebar-treeview .e-treeview,
.sidebar-treeview .e-treeview .e-ul {
  padding: 0;
  margin: 0;
}
.sidebar-treeview .e-treeview .e-text-content {
  padding-left: 14px;
}
#wrapper .e-appbar {
  border-bottom: 1px solid #eaeaea0;
}
.sidebar-treeview {
  border-right: 1px solid #eaeaea0;
}
#reswrapper{
  border: 1px solid #d7d7d7;
}
</style>

```

{% previewsample "page.domainurl/code-snippet/appbar/sidebar-cs1" %}

### Style and appearance in Vue AppBar component

To modify the AppBar appearance, you need to override the default CSS of the AppBar component. Please find the list of CSS classes and their corresponding sections in the AppBar component. Also, you have an option to create your own custom theme for the controls using our [Theme Studio](#).

| CSS Class | Purpose of Class |

|-----|-----|

| .e-appbar | To customize the appbar. |

| .e-appbar.e-prominent | To customize the prominent appbar. |

| .e-appbar.e-dense | To customize the dense appbar. |

| .e-appbar.e-light | To customize the light appbar. |

| .e-appbar.e-dark | To customize the dark appbar. |

| .e-appbar.e-primary | To customize the dark appbar. |

| .e-appbar.e-inherit | To customize the inherit appbar. |

Note: You can change the prominent AppBar height if larger titles, images, or texts are used.

### CssClass

CssClass is used for AppBar customization based on the custom class. In the example below, the AppBar background and color are customized using the [cssClass](#) property.

**APP.VUE**

```

<template>
  <div class="control-container">
    <ejs-appbar colorMode="Primary" cssClass="custom-appbar">
      <ejs-button cssClass="e-inherit" iconCss="e-icons e-home"></ejs-
button>
    </ejs-appbar>
  </div>
</template>
<script>
  import Vue from "vue";
  import { AppBarPlugin } from "@syncfusion/ej2-vue-navigations";
  import { ButtonPlugin } from "@syncfusion/ej2-vue-buttons";
  Vue.use(AppBarPlugin);
  Vue.use(ButtonPlugin);
  export default {
    data: function () {
      return {};
    }
  };
</script>
<style>
  @import "../node_modules/@syncfusion/ej2-base/styles/material.css";
  @import "../node_modules/@syncfusion/ej2-buttons/styles/material.css";
  @import "../node_modules/@syncfusion/ej2-navigations/styles/material.css";
  .control-container .e-appbar.custom-appbar {
    background: #ff0000;
    color: #fff;
  }
</style>

```

{% previewsample "page.domainurl/code-snippet/appbar/getting-started-cs16" %}

**HtmlAttributes**

It can be used for additional inline attributes by specifying as inline attributes or by specifying htmlAttributes directive. In the code example below, the aria-label of the AppBar is customized by specifying as attributes.

**APP.VUE**

```

<template>
  <div class="control-container">
    <ejs-appbar colorMode="Primary" aria-label="appbar">
      <ejs-button cssClass="e-inherit" iconCss="e-icons e-home"></ejs-
button>
    </ejs-appbar>
  </div>
</template>
<script>
  import Vue from "vue";
  import { AppBarPlugin } from "@syncfusion/ej2-vue-navigations";
  import { ButtonPlugin } from "@syncfusion/ej2-vue-buttons";
  Vue.use(AppBarPlugin);
  Vue.use(ButtonPlugin);
  export default {

```

```

    data: function () {
      return {};
    }
  };
</script>
<style>
  @import "../node_modules/@syncfusion/ej2-base/styles/material.css";
  @import "../node_modules/@syncfusion/ej2-buttons/styles/material.css";
  @import "../node_modules/@syncfusion/ej2-navigations/styles/material.css";
</style>

```

```
{% previewsample "page.domainurl/code-snippet/appbar/getting-started-cs17" %}
```

## AutoComplete

### Getting Started with the Vue Auto complete Component in Vue 2

This article provides a step-by-step guide for setting up a Vue 2 project using [Vue-CLI](#) and integrating the Syncfusion Vue Auto complete component using the [Composition API](#) / [Options API](#)[Link to the Video](#).

#### Prerequisites

[System requirements for Syncfusion Vue UI components](#)

#### Dependencies

The list of dependencies required to use the AutoComplete component in your application is given below:

```

`javascript
|-- @syncfusion/ej2-vue-dropdowns
|-- @syncfusion/ej2-base
|-- @syncfusion/ej2-dropdowns
|-- @syncfusion/ej2-data
|-- @syncfusion/ej2-inputs
|-- @syncfusion/ej2-lists
|-- @syncfusion/ej2-navigations
|-- @syncfusion/ej2-popups
|-- @syncfusion/ej2-buttons
`

```

#### Setting up the Vue 2 project

To generate a Vue 2 project using Vue-CLI, use the [vue create](#) command. Follow these steps to install Vue CLI and create a new project:

```

`bash
npm install -g @vue/cli
vue create quickstart
cd quickstart

```



```
npm run serve
```

```
,
```

```
or
```

```
`bash
```

```
yarn global add @vue/cli
```

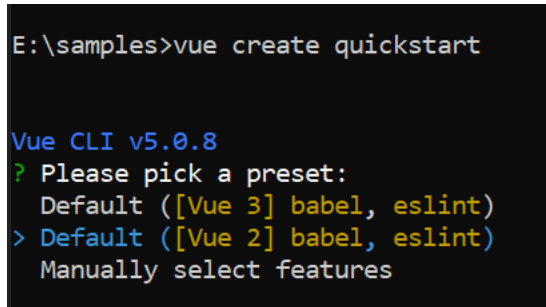
```
vue create quickstart
```

```
cd quickstart
```

```
yarn run serve
```

```
,
```

When creating a new project, choose the option `Default ([Vue 2] babel, eslint)` from the menu.



```
E:\samples>vue create quickstart

Vue CLI v5.0.8
? Please pick a preset:
  Default ([Vue 3] babel, eslint)
> Default ([Vue 2] babel, eslint)
  Manually select features
```

Once the `quickstart` project is set up with default settings, proceed to add Syncfusion components to the project.

### Add Syncfusion Vue packages

Syncfusion packages are available at [npmjs.com](https://www.npmjs.com). To use Vue components, install the required npm package.

This article uses the [Vue Auto complete component](#) as an example. Install the `@syncfusion/ej2-vue-dropdowns` package by running the following command:

```
`bash
```

```
npm install @syncfusion/ej2-vue-dropdowns --save
```

```
,
```

```
or
```

```
`bash
```

```
yarn add @syncfusion/ej2-vue-dropdowns
```

```
,
```

### Import Syncfusion CSS styles

You can import themes for the Syncfusion Vue component in various ways, such as using CSS or SASS styles from npm packages, CDN, [CRG](#) and [Theme Studio](#). Refer to [themes topic](#) to know more about built-in themes and different ways to refer to themes in a Vue project.

In this article, the **Material** theme is applied using CSS styles, which are available in installed packages. The necessary **Material** CSS styles for the Auto complete component and its dependents were imported into the `<style>` section of **src/App.vue** file.

#### ~/SRC/APP.VUE

```
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-inputs/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-dropdowns/styles/material.css";
</style>
```

#### Add Syncfusion Vue component

Follow the below steps to add the Vue Auto complete component using **Composition API** or **Options API**:

1\ First, import and register the Auto complete component in the **script** section of the **src/App.vue** file. If you are using the **Composition API**, you should add the **setup** attribute to the **script** tag to indicate that Vue will be using the **Composition API**.

#### COMPOSITION API (~/SRC/APP.VUE)

```
<script setup>
import { AutoCompleteComponent as EjsAutocomplete } from '@syncfusion/ej2-vue-dropdowns';
</script>
```

#### OPTIONS API (~/SRC/APP.VUE)

```
<script>
import { AutoCompleteComponent } from '@syncfusion/ej2-vue-dropdowns';
export default {
  components: {
    'ejs-autocomplete': AutoCompleteComponent
  }
}
</script>
```

#### Binding data source

After initialization, populate the AutoComplete with data using the **dataSource** property. Here, an array of string values is passed to the AutoComplete component.

#### COMPOSITION API (~/SRC/APP.VUE)

```
<template>
<div id="app">
<ejs-autocomplete :dataSource='sportsData' :placeholder="waterMark"></ejs-autocomplete>
</div>
</template>
<script setup>
import { AutoCompleteComponent as EjsAutocomplete } from '@syncfusion/ej2-vue-dropdowns';
const waterMark = 'e.g. Basketball';
```

```
const sportsData = ['American Football', 'Badminton', 'Basketball',
  'Cricket',
  'Football', 'Golf', 'Gymnastics',
  'Hockey', 'Rugby', 'Snooker', 'Tennis'
];
</script>
```

### OPTIONS API (~SRC/APP.VUE)

```
</template>
<script>
import { AutoCompleteComponent } from '@syncfusion/ej2-vue-dropdowns';
export default {
  components: {
    'ejs-autocomplete': AutoCompleteComponent
  },
  name: 'app',
  data () {
    return {
      waterMark : 'e.g. Basketball',
      sportsData: ['American Football', 'Badminton', 'Basketball', 'Cricket',
        'Football', 'Golf', 'Gymnastics',
        'Hockey', 'Rugby', 'Snooker', 'Tennis'
      ]
    }
  }
}
</script>
```

Here is the summarized code for the above steps in the **src/App.vue** file:

### COMPOSITION API (~SRC/APP.VUE)

```
<template>
  <div id="app">
    <ejs-autocomplete :dataSource='sportsData'
    :placeholder="waterMark"></ejs-autocomplete>
  </div>
</template>
<script setup>
import { AutoCompleteComponent as EjsAutocomplete } from '@syncfusion/ej2-vue-dropdowns';
const waterMark = 'e.g. Basketball';
const sportsData = ['American Football', 'Badminton', 'Basketball',
  'Cricket',
    'Football', 'Golf', 'Gymnastics',
    'Hockey', 'Rugby', 'Snooker', 'Tennis'
  ];
</script>
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-inputs/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-dropdowns/styles/material.css";
#app {
  color: #008cff;
  height: 40px;
}
```

```
    left: 35%;
    position: absolute;
    top: 35%;
    width: 30%;
  }
</style>
```

### OPTIONS API (~SRC/APP.VUE)

```
<template>
  <div id="app">
    <ejs-autocomplete :dataSource='sportsData'
:placeholder="waterMark"></ejs-autocomplete>
  </div>
</template>
<script>
import { AutoCompleteComponent } from '@syncfusion/ej2-vue-dropdowns';
export default {
  components: {
    'ejs-autocomplete': AutoCompleteComponent
  },
  name: 'app',
  data () {
    return {
      waterMark : 'e.g. Basketball',
      sportsData: ['American Football', 'Badminton', 'Basketball',
'Cricket',
                  'Football', 'Golf', 'Gymnastics',
                  'Hockey', 'Rugby', 'Snooker', 'Tennis'
    ]
  }
}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-inputs/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-dropdowns/styles/material.css";
#app {
  color: #008cff;
  height: 40px;
  left: 35%;
  position: absolute;
  top: 35%;
  width: 30%;
}
</style>
```

### Run the project

To run the project, use the following command:

```
`bash
```

```
npm run serve
```

```
`
```

or

```
`bash
```

```
yarn run serve
```

```
,
```

```
{% previewsample "page.domainurl/code-snippet/auto-complete/getting-started-cs11" %}
```

### Custom values

The AutoComplete allows the user to give input as custom value which is not required to present in predefined set of values. By default, this support is enabled by [allowCustom](#) property. The custom value will be sent to post back handler when a form is about to be submitted.

### COMPOSITION API (~SRC/APP.VUE)

```
<template>
  <div id="app">
    <ejs-autocomplete :dataSource='sportsData' :placeholder="waterMark"
  ></ejs-autocomplete>
  </div>
</template>
<script setup>
import { AutoCompleteComponent as EjsAutocomplete } from '@syncfusion/ej2-
vue-dropdowns';
const waterMark = 'Find a game';
const sportsData = ['Badminton', 'Basketball', 'Cricket',
                    'Football', 'Golf', 'Gymnastics',
                    'Hockey', 'Rugby', 'Snooker', 'Tennis'
                    ];
</script>
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-inputs/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-dropdowns/styles/material.css";
#app {
  color: #008cff;
  height: 40px;
  left: 35%;
  position: absolute;
  top: 35%;
  width: 30%;
}
</style>
```

### OPTIONS API (~SRC/APP.VUE)

```
<template>
  <div id="app">
    <ejs-autocomplete :dataSource='sportsData' :placeholder="waterMark"
  ></ejs-autocomplete>
  </div>
</template>
<script>
import { AutoCompleteComponent } from '@syncfusion/ej2-vue-dropdowns';
export default {
```

```

components: {
  'ejs-autocomplete': AutoCompleteComponent
},
name: 'app',
data () {
  return {
    waterMark : 'Find a game',
    allowCustom: true,
    sportsData: ['Badminton', 'Basketball', 'Cricket',
                  'Football', 'Golf', 'Gymnastics',
                  'Hockey', 'Rugby', 'Snooker', 'Tennis'
                ]
  }
}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-inputs/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-dropdowns/styles/material.css";
#app {
  color: #008cff;
  height: 40px;
  left: 35%;
  position: absolute;
  top: 35%;
  width: 30%;
}
</style>

```

{% previewsample "page.domainurl/code-snippet/auto-complete/getting-started-cs12" %}

### Configure the suggestion list

By default, suggestion list width automatically adjusts according to the AutoComplete input element's width, and the height of the suggestion list has '300px'. The height and width of the popup list can also be customized using the [popupHeight](#) and [popupWidth](#) property respectively. In the following sample, suggestion list's width and height are configured.

### COMPOSITION API (~SRC/APP.VUE)

```

<template>
  <div id="app">
    <ejs-autocomplete :dataSource='sportsData' :popupHeight='height'
    :popupWidth='width' :placeholder="waterMark" ></ejs-autocomplete>
  </div>
</template>
<script setup>
import { AutoCompleteComponent as EjsAutocomplete } from '@syncfusion/ej2-vue-dropdowns';
const waterMark = 'Find a game';
const height = '250px';
const width = '250px';
const sportsData = ['Badminton', 'Basketball', 'Cricket',
                     'Football', 'Golf', 'Gymnastics',
                     'Hockey', 'Rugby', 'Snooker', 'Tennis'
                    ];

```

```

</script>
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-inputs/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-dropdowns/styles/material.css";
#app {
  color: #008cff;
  height: 40px;
  left: 35%;
  position: absolute;
  top: 35%;
  width: 30%;
}
</style>

```

### OPTIONS API (~SRC/APP.VUE)

```

<template>
  <div id="app">
    <ejs-autocomplete :dataSource='sportsData' :popupHeight='height'
:popupWidth='width' :placeholder="waterMark" ></ejs-autocomplete>
  </div>
</template>
<script>
import { AutoCompleteComponent } from '@syncfusion/ej2-vue-dropdowns';
export default {
  components: {
    'ejs-autocomplete': AutoCompleteComponent
  },
  name: 'app',
  data () {
    return {
      waterMark : 'Find a game',
      allowCustom: true,
      height: '250px',
      width: '250px',
      sportsData: ['Badminton', 'Basketball', 'Cricket',
        'Football', 'Golf', 'Gymnastics',
        'Hockey', 'Rugby', 'Snooker', 'Tennis'
      ]
    }
  }
}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-inputs/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-dropdowns/styles/material.css";
#app {
  color: #008cff;
  height: 40px;
  left: 35%;
  position: absolute;
  top: 35%;
  width: 30%;
}

```

```
</style>
```

```
{% previewsample "page.domainurl/code-snippet/auto-complete/getting-started-cs13" %}
```

**Note:** You can refer to our [Vue AutoComplete](#) feature tour page for its groundbreaking feature representations. You can also explore our [Vue AutoComplete example](#) to know how to render and configure the autocomplete.

See Also

- [How to bind the data](#)

## Getting Started with the Vue AutoComplete Component in Vue 3

This article provides a step-by-step guide for setting up a [Vite](#) project with a JavaScript environment and integrating the Syncfusion Vue AutoComplete component using the [Composition API](#) / [Options API](#).

The **Composition API** is a new feature introduced in Vue.js 3 that provides an alternative way to organize and reuse component logic. It allows developers to write components as functions that use smaller, reusable functions called composition functions to manage their properties and behavior.

The **Options API** is the traditional way of writing Vue.js components, where the component logic is organized into a series of options that define the component's properties and behavior. These options include data, methods, computed properties, watchers, lifecycle hooks, and more.

### Prerequisites

#### [System requirements for Syncfusion Vue UI components](#)

#### Set up the Vite project

A recommended approach for beginning with Vue is to scaffold a project using [Vite](#). To create a new Vite project, use one of the commands that are specific to either NPM or Yarn.

```
`bash
```

```
npm create vite@latest
```

```
,
```

or

```
`bash
```

```
yarn create vite
```

```
,
```

Using one of the above commands will lead you to set up additional configurations for the project as below:

1. Define the project name: We can specify the name of the project directly. Let's specify the name of the project as **my-project** for this article.

```
`bash
```

```
? Project name: » my-project
```

```
,
```



2. Select **Vue** as the framework. It will create a Vue 3 project.

```
`bash
```

? Select a framework: » - Use arrow-keys. Return to submit.

Vanilla

Vue

React

Preact

Lit

Svelte

Others

,

3. Choose **JavaScript** as the framework variant to build this Vite project using JavaScript and Vue.

```
`bash
```

? Select a variant: » - Use arrow-keys. Return to submit.

JavaScript

TypeScript

Customize with create-vue ↗

Nuxt ↗

,

4. Upon completing the aforementioned steps to create the **my-project**, run the following command to install its dependencies:

```
`bash
```

```
cd my-project
```

```
npm install
```

,

or

```
`bash
```

```
cd my-project
```

```
yarn install
```

,

Now that **my-project** is ready to run with default settings, let's add Syncfusion components to the project.

### Add Syncfusion Vue packages

Syncfusion Vue component packages are available at [npmjs.com](https://www.npmjs.com). To use Syncfusion Vue components in the project, install the corresponding npm package.

This article uses the [Vue AutoComplete component](#) as an example. To use the Vue AutoComplete component in the project, the `@syncfusion/ej2-vue-dropdowns` package needs to be installed using the following command:

```
`bash
```

```
npm install @syncfusion/ej2-vue-dropdowns --save
```

```
,
```

or

```
`bash
```

```
yarn add @syncfusion/ej2-vue-dropdowns
```

```
,
```

### Import Syncfusion CSS styles

You can import themes for the Syncfusion Vue component in various ways, such as using CSS or SASS styles from npm packages, CDN, [CRG](#) and [Theme Studio](#). Refer to [themes topic](#) to know more about built-in themes and different ways to refer to themes in a Vue project.

In this article, `Material` theme is applied using CSS styles, which are available in installed packages. The necessary `Material` CSS styles for the AutoComplete component and its dependents were imported into the `<style>` section of `src/App.vue` file.

#### ~/SRC/APP.VUE

```
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-inputs/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-dropdowns/styles/material.css";
</style>
```

The order of importing CSS styles should be in line with its dependency graph.

### Add Syncfusion Vue component

Follow the below steps to add the Vue AutoComplete component using `Composition API` or `Options API`:

1.First, import and register the AutoComplete component in the `script` section of the `src/App.vue` file. If you are using the `Composition API`, you should add the `setup` attribute to the `script` tag to indicate that Vue will be using the `Composition API`.

#### COMPOSITION API (~/SRC/APP.VUE)

```
<script setup>
import { AutoCompleteComponent as EjsAutocomplete } from "@syncfusion/ej2-
vue-dropdowns";
</script>
```

**OPTIONS API (~SRC/APP.VUE)**

```
<script>
import { AutoCompleteComponent } from "@syncfusion/ej2-vue-dropdowns";
//Component registration
export default {
  name: "App",
  components: {
    'ejs-autocomplete' : AutoCompleteComponent,
  }
}
</script>
```

2. In the **template** section, define the AutoComplete component with the [dataSource](#) property and column definitions.

**~/SRC/APP.VUE**

```
<template>
<div class="control_wrapper">
<ejs-autocomplete :dataSource='sportsData' :placeholder="waterMark"></ejs-
autocomplete>
</div>
</template>
```

Here is the summarized code for the above steps in the **src/App.vue** file:

**COMPOSITION API (~SRC/APP.VUE)**

```
<template>
<div class="control_wrapper">
<ejs-autocomplete :dataSource='data[0].sportsData'
:placeholder="data[0].waterMark"></ejs-autocomplete>
</div>
</template>
<script setup>
import { AutoCompleteComponent as EjsAutocomplete } from "@syncfusion/ej2-
vue-dropdowns";
const data = [{ waterMark : 'e.g. Basketball',
sportsData: ['American Football', 'Badminton', 'Basketball', 'Cricket',
'Football', 'Golf', 'Gymnastics',
'Hockey', 'Rugby', 'Snooker', 'Tennis'
]}]
</script>
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-inputs/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-dropdowns/styles/material.css";
</style>
```

**OPTIONS API (~SRC/APP.VUE)**

```
<template>
<div class="control_wrapper">
<ejs-autocomplete :dataSource='sportsData' :placeholder="waterMark"></ejs-
autocomplete>
```

```
</div>
</template>
<script>
import { AutoCompleteComponent } from "@syncfusion/ej2-vue-dropdowns";
//Component registration
export default {
  name: 'App',
  components: {
    "ejs-autocomplete": AutoCompleteComponent
  },
  data () {
    return {
      waterMark : 'e.g. Basketball',
      sportsData: ['American Football', 'Badminton', 'Basketball', 'Cricket',
        'Football', 'Golf', 'Gymnastics',
        'Hockey', 'Rugby', 'Snooker', 'Tennis'
      ]
    }
  }
}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-inputs/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-dropdowns/styles/material.css";
</style>
```

### Run the project

To run the project, use the following command:

```
`bash
```

```
npm run dev
```

```
,
```

or

```
`bash
```

```
yarn run dev
```

```
,
```

### Custom values

The AutoComplete allows the user to give input as custom value which is not required to present in predefined set of values. By default, this support is enabled by [allowCustom](#) property. The custom value will be sent to post back handler when a form is about to be submitted.

### COMPOSITION API (~SRC/APP.VUE)

```
<template>
<div id="app">
<ejs-autocomplete :dataSource='data[0].sportsData'
:placeholder="data[0].waterMark" :allowcuston="data[0].allowcustom"></ejs-
autocomplete>
</div>
</template>
```

```

<script setup>
import { AutoCompleteComponent as EjsAutocomplete } from "@syncfusion/ej2-vue-dropdowns";
const data = [{ waterMark : 'Find a game',
sportsData: ['Badminton', 'Basketball', 'Cricket',
'Football', 'Golf', 'Gymnastics',
'Hockey', 'Rugby', 'Snooker', 'Tennis'],
allowcuston: true }]
</script>
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-inputs/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-dropdowns/styles/material.css";
#app {
color: #008c8f;
height: 40px;
left: 35%;
position: absolute;
top: 35%;
width: 30%;
}
</style>

```

### OPTIONS API (~SRC/APP.VUE)

```

<template>
<div id="app">
<ejs-autocomplete :dataSource='sportsData' :placeholder="waterMark"
:allowcuston="allowcuston"></ejs-autocomplete>
</div>
</template>
<script>
import { AutoCompleteComponent } from "@syncfusion/ej2-vue-dropdowns";
//Component registration
export default {
name: 'App',
components: {
"ejs-autocomplete": AutoCompleteComponent
},
data () {
return {
allowcuston: true,
waterMark : 'Find a game',
sportsData: ['Badminton', 'Basketball', 'Cricket',
'Football', 'Golf', 'Gymnastics',
'Hockey', 'Rugby', 'Snooker', 'Tennis'
]
}
}
}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-inputs/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-dropdowns/styles/material.css";
#app {

```

```

color: #008cff;
height: 40px;
left: 35%;
position: absolute;
top: 35%;
width: 30%;
}
</style>

```

### Configure the suggestion list

By default, suggestion list width automatically adjusts according to the AutoComplete input element's width, and the height of the suggestion list has '300px'. The height and width of the popup list can also be customized using the [popupHeight](#) and [popupWidth](#) property respectively. In the following sample, suggestion list's width and height are configured.

#### COMPOSITION API (~SRC/APP.VUE)

```

<template>
<div id="app">
<ejs-autocomplete :dataSource='data[0].sportsData'
:popupHeight='data[0].height' :popupWidth='data[0].width'
:placeholder="data[0].waterMark" ></ejs-autocomplete>
</div>
</template>
<script setup>
import { AutoCompleteComponent as EjsAutocomplete } from "@syncfusion/ej2-
vue-dropdowns";
const data = [{ waterMark : 'Find a game',
height: '250px',
width: '250px',
sportsData: ['Badminton', 'Basketball', 'Cricket',
'Football', 'Golf', 'Gymnastics',
'Hockey', 'Rugby', 'Snooker', 'Tennis']}]
</script>
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-inputs/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-dropdowns/styles/material.css";
#app {
color: #008cff;
height: 40px;
left: 35%;
position: absolute;
top: 35%;
width: 30%;
}
</style>

```

#### OPTIONS API (~SRC/APP.VUE)

```

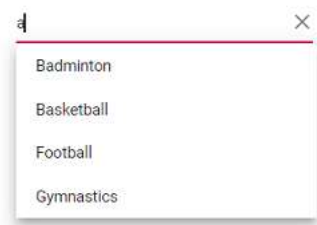
<template>
<div id="app">
<ejs-autocomplete :dataSource='sportsData' :popupHeight='height'
:popupWidth='width' :placeholder="waterMark" ></ejs-autocomplete>
</div>

```

```

</template>
<script>
import { AutoCompleteComponent } from "@syncfusion/ej2-vue-dropdowns";
//Component registration
export default {
  name: 'App',
  components: {
    "ejs-autocomplete": AutoCompleteComponent
  },
  data () {
    return {
      waterMark : 'Find a game',
      allowCustom: true,
      height: '250px',
      width: '250px',
      sportsData: ['Badminton', 'Basketball', 'Cricket',
        'Football', 'Golf', 'Gymnastics',
        'Hockey', 'Rugby', 'Snooker', 'Tennis'
      ]
    }
  }
}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-inputs/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-dropdowns/styles/material.css";
#app {
  color: #008cff;
  height: 40px;
  left: 35%;
  position: absolute;
  top: 35%;
  width: 30%;
}
</style>

```



See Also

- [How to bind the data](#)

### Data binding in Vue Auto complete component

The AutoComplete loads the data either from local data sources or remote data services using the [dataSource](#) property. It supports the data type of array or DataManager. The AutoComplete also

supports different kind of data services such as OData, OData V4, Web API and data formats such as XML, JSON, JSONP with the help of DataManager Adaptors.

| Fields | Type | Description |

|-----|-----|-----|

| value | number or string | Specifies the hidden data value mapped to each list item that should contain a unique value. |

| groupBy | string | Specifies the category under which the list item has to be grouped. |

| iconCss | string | Specifies the icon class of each list item |

While binding complex data to AutoComplete, fields should be mapped correctly. Otherwise, the selected item remains undefined.

### Bind to local data

Local data can be represented in two ways, they are as follows:

#### Array of string

The AutoComplete has support to load array of primitive data such as strings and numbers.

### APP.VUE

```
<template>
  <div id="app">
    <ejs-autocomplete :dataSource='sportsData' :placeholder="waterMark"
  ></ejs-autocomplete>
  </div>
</template>
<script>
import Vue from 'vue';
import { AutoCompletePlugin } from '@syncfusion/ej2-vue-dropdowns';
Vue.use(AutoCompletePlugin);
export default {
  name: 'app',
  data () {
    return {
      waterMark : 'Find a game',
      allowCustom: true,
      sportsData: ['Badminton', 'Basketball', 'Cricket',
        'Football', 'Golf', 'Gymnastics',
        'Hockey', 'Rugby', 'Snooker', 'Tennis'
      ]
    }
  }
}
</script>
<style>
@import "../../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../../node_modules/@syncfusion/ej2-inputs/styles/material.css";
@import "../../node_modules/@syncfusion/ej2-vue-
dropdowns/styles/material.css";
#app {
  color: #008cff;
  height: 40px;
  left: 35%;
}
```



```

    position: absolute;
    top: 35%;
    width: 30%;
  }
</style>

```

{% previewsample "page.domainurl/code-snippet/auto-complete/getting-started-cs2" %}

### Array of object

The AutoComplete can generate its list items through an array of complex data. For this, the appropriate columns should be mapped to the `fields` property.

In the following example, Game column from complex data have been mapped to the value field.

### APP.VUE

```

<template>
  <div id="app">
    <ejs-autocomplete :dataSource='sportsData' :fields='fields'
    :placeholder="waterMark" ></ejs-autocomplete>
  </div>
</template>
<script>
import Vue from 'vue';
import { AutoCompletePlugin } from '@syncfusion/ej2-vue-dropdowns';
Vue.use(AutoCompletePlugin);
export default {
  name: 'app',
  data () {
    return {
      waterMark : 'Find a game',
      sportsData: [
        { Id: 'Game1', Game: 'Badminton' },
        { Id: 'Game2', Game: 'Basketball' },
        { Id: 'Game3', Game: 'Cricket' },
        { Id: 'Game4', Game: 'Football' },
        { Id: 'Game5', Game: 'Golf' },
        { Id: 'Game6', Game: 'Hockey' },
        { Id: 'Game7', Game: 'Rugby' },
        { Id: 'Game8', Game: 'Snooker' }
      ],
      fields: { value: 'Game' }
    }
  }
}
</script>
<style>
@import "../../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../../node_modules/@syncfusion/ej2-inputs/styles/material.css";
@import "../../node_modules/@syncfusion/ej2-vue-
dropdowns/styles/material.css";
#app {
  color: #008c8f;
  height: 40px;
  left: 35%;
  position: absolute;

```

```

    top: 35%;
    width: 30%;
  }
</style>

```

```
{% previewsample "page.domainurl/code-snippet/auto-complete/getting-started-cs3" %}
```

### Array of complex object

The AutoComplete can generate its list items through an array of complex data. For this, the appropriate columns should be mapped to the `fields` property. In the following example, `Country.Name` column from complex data have been mapped to the value field.

### APP.VUE

```

<template>
  <div id="app">
    <ejs-autocomplete :dataSource='countriesData' :fields='fields'
    :placeholder="waterMark" ></ejs-autocomplete>
  </div>
</template>
<script>
import Vue from 'vue';
import { AutoCompletePlugin } from '@syncfusion/ej2-vue-dropdowns';
Vue.use(AutoCompletePlugin);
export default {
  name: 'app',
  data () {
    return {
      waterMark : 'Find a country',
      countriesData: [
        { Country: { Name: 'Australia' }, Code: { Id: 'AU' } },
        { Country: { Name: 'Bermuda' }, Code: { Id: 'BM' } },
        { Country: { Name: 'Canada' }, Code: { Id: 'CA' } },
        { Country: { Name: 'Cameroon' }, Code: { Id: 'CM' } },
        { Country: { Name: 'Denmark' }, Code: { Id: 'DK' } },
        { Country: { Name: 'France' }, Code: { Id: 'FR' } },
        { Country: { Name: 'Finland' }, Code: { Id: 'FI' } },
        { Country: { Name: 'Germany' }, Code: { Id: 'DE' } },
        { Country: { Name: 'Greenland' }, Code: { Id: 'GL' } },
        { Country: { Name: 'Hong Kong' }, Code: { Id: 'HK' } },
        { Country: { Name: 'India' }, Code: { Id: 'IN' } },
        { Country: { Name: 'Italy' }, Code: { Id: 'IT' } },
        { Country: { Name: 'Japan' }, Code: { Id: 'JP' } },
        { Country: { Name: 'Mexico' }, Code: { Id: 'MX' } },
        { Country: { Name: 'Norway' }, Code: { Id: 'NO' } },
        { Country: { Name: 'Poland' }, Code: { Id: 'PL' } },
        { Country: { Name: 'Switzerland' }, Code: { Id: 'CH' } },
        { Country: { Name: 'United Kingdom' }, Code: { Id: 'GB' } },
        { Country: { Name: 'United States' }, Code: { Id: 'US' } }
      ],
      fields: { value: 'Country.Name' }
    }
  }
}
</script>
<style>

```

```
@import "../../../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../../../node_modules/@syncfusion/ej2-inputs/styles/material.css";
@import "../../../node_modules/@syncfusion/ej2-vue-
dropdowns/styles/material.css";
#app {
  color: #008cff;
  height: 40px;
  left: 35%;
  position: absolute;
  top: 10%;
  width: 30%;
}
</style>
```

{% previewsample "page.domainurl/code-snippet/auto-complete/getting-started-cs4" %}

### Bind to remote data

The AutoComplete supports retrieval of data from remote data services with the help of **DataManager** component. The Query property is used to fetch data from the database and bind it to the AutoComplete. The following sample displays the first 6 contacts from the Customers table of the Northwind data service.

### APP.VUE

```
<template>
  <div id="app">
    <ejs-autocomplete :dataSource='data' :fields='fields'
    sortOrder='sortOrder' :query='query' :placeholder="waterMark" ></ejs-
    autocomplete>
  </div>
</template>
<script>
import Vue from 'vue';
import { AutoCompletePlugin } from '@syncfusion/ej2-vue-dropdowns';
import { Query, DataManager, ODataV4Adaptor } from '@syncfusion/ej2-data';
Vue.use(AutoCompletePlugin);
var remoteData = new DataManager({
  url: 'https://services.odata.org/V4/Northwind/Northwind.svc/Customers',
  adaptor: new ODataV4Adaptor,
  crossDomain: true
});
export default {
  name: 'app',
  data () {
    return {
      waterMark : 'Find a customer',
      data: remoteData,
      fields: { value: 'ContactName' },
      query: new Query().select(['ContactName', 'CustomerID']),
      sortOrder: 'Ascending'
    }
  }
}
</script>
<style>
@import "../../../node_modules/@syncfusion/ej2-base/styles/material.css";
```

```
@import "../node_modules/@syncfusion/ej2-inputs/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-
dropdowns/styles/material.css";
#app {
  color: #008cff;
  height: 40px;
  left: 35%;
  position: absolute;
  top: 12%;
  width: 30%;
}
</style>
```

{% previewsample "page.domainurl/code-snippet/auto-complete/getting-started-cs5" %}

See Also

- [How to load data using template](#)
- [How to group the data using header](#)
- [How to filter the bound data](#)

## Templates in Vue Auto complete component

The AutoComplete has been provided with several options to customize each list items, group title, header and footer elements. It uses the Essential JS 2 Template engine to compile and render the elements properly.

### Item template

The content of each list item within the AutoComplete can be customized with the help of [itemTemplate](#) property.

In the following sample, each list item is split into two columns to display relevant data's.

### APP.VUE

```
<template>
  <div id="app">
    <div class='autocomplete'>
      <ejs-autocomplete id='employees' :query='query' :dataSource='data'
      :fields='fields' :placeholder='waterMark' :sortOrder='sortOrder'
      :itemTemplate='iTemplate' popupHeight="450px"></ejs-autocomplete>
    </div>
  </div>
</template>
<script>
import Vue from 'vue';
import { AutoCompletePlugin } from '@syncfusion/ej2-vue-dropdowns';
import { Query, DataManager, ODataV4Adaptor } from '@syncfusion/ej2-data';
Vue.use(AutoCompletePlugin);
var itemVue = Vue.component("itemTemplate", {
  template: `<span><span class='name'>{{data.FirstName}}</span><span class
='city'>{{data.City}}</span></span>`,
  data() {
    return {
      data: {}
    };
  }
});
```

```

    }
  });
  var remoteData = new DataManager({
    url: 'https://services.odata.org/V4/Northwind/Northwind.svc',
    adaptor: new ODataV4Adaptor,
    crossDomain: true
  });
  export default {
    name: 'app',
    data () {
      return {
        fields: { value: 'FirstName' },
        waterMark: 'Find an employee',
        sortOrder: 'Ascending',
        data: remoteData,
        iTemplate: function(e) {
          return {
            template: itemVue
          };
        },
        query: new Query().from('Employees').select(['FirstName', 'City',
        'EmployeeID']).take(6),
      }
    }
  }
</script>
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-inputs/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-dropdowns/styles/material.css";
#app {
  color: #008cff;
  height: 40px;
  position: absolute;
  width: 90%;
  top: 10%;
}
.autocomplete {
  width: 30%;
  margin: 0 auto;
}
.city{
  right: 15px;
  position: absolute;
}
</style>

```

{% previewsample "page.domainurl/code-snippet/auto-complete/getting-started-cs20" %}

### Group template

The group header title under which appropriate sub-items are categorized can also be customize with the help of [groupTemplate](#) property. This template is common for both inline and floating group header template.

In the following sample, employees are grouped according to their city.

**APP.VUE**

```

<template>
  <div id="app">
    <div class='autocomplete'>
      <ejs-autocomplete id='employees' :query='query' :dataSource='data'
:fields='fields' :placeholder='waterMark' :sortOrder='sortOrder'
:groupTemplate='gTemplate' popupHeight="450px"></ejs-autocomplete>
    </div>
  </div>
</template>
<script>
import Vue from 'vue';
import { AutoCompletePlugin } from '@syncfusion/ej2-vue-dropdowns';
import { Query, Predicate, DataManager, ODataV4Adaptor } from
'@syncfusion/ej2-data';
Vue.use(AutoCompletePlugin);
var groupTemplate = Vue.component("groupTemplate", {
  template: `<strong>{{data.City}}</strong>`,
  data() {
    return {
      data: {}
    };
  }
});
var remoteData = new DataManager({
  url: 'https://services.odata.org/V4/Northwind/Northwind.svc',
  adaptor: new ODataV4Adaptor,
  crossDomain: true
});
export default {
  name: 'app',
  data () {
    return {
      fields: { value: 'FirstName', groupBy:'City' },
      waterMark: 'Find an employee',
      sortOrder: 'Ascending',
      data: remoteData,
      gTemplate: function(e) {
        return {
          template: groupTemplate
        };
      },
      query: new Query().from('Employees').select(['FirstName',
'City', 'EmployeeID']).take(6).where(new Predicate('City',
'equal','london').or('City','equal','seattle')),
    }
  }
}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-inputs/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-dropdowns/styles/material.css";
#app {
  color: #008cff;
  height: 40px;

```

```

    position: absolute;
    width: 90%;
    top: 10%;
  }
  .autocomplete {
    width: 30%;
    margin: 0 auto;
  }
</style>

```

{% previewsample "page.domainurl/code-snippet/auto-complete/getting-started-cs21" %}

### Header template

The header element is shown statically at the top of the suggestion list items within the AutoComplete, and any custom element can be placed as a header element using [headerTemplate](#) property.

In the following sample, the list items and its headers are designed and displayed as two columns similar to multiple columns of the grid.

### APP.VUE

```

<template>
  <div id="app">
    <div class='autocomplete'>
      <ejs-autocomplete id='employees' :query='query' :dataSource='data'
      :fields='fields' :placeholder='waterMark' :headerTemplate='hTemplate'
      :sortOrder='sortOrder' :itemTemplate='iTemplate' popupHeight="450px"></ejs-
      autocomplete>
    </div>
  </div>
</template>
<script>
import Vue from 'vue';
import { AutoCompletePlugin } from '@syncfusion/ej2-vue-dropdowns';
import { Query, DataManager, ODataV4Adaptor } from '@syncfusion/ej2-data';
Vue.use(AutoCompletePlugin);
var headerVue = Vue.component("headerTemplate", {
  template: `<span class='head'><span class='name'>Name</span><span
class='city'>City</span></span>`,
  data() {
    return {
      data: {}
    };
  }
});
var itemVue = Vue.component("itemTemplate", {
  template: `<span class='item'><span class='name'>
{{data.FirstName}}</span><span class='city'>{{data.City}}</span></span>`,
  data() {
    return {
      data: {}
    };
  }
});
var remoteData = new DataManager({
  url: 'https://services.odata.org/V4/Northwind/Northwind.svc',

```

```

    adaptor: new ODataV4Adaptor,
    crossDomain: true
  });
  export default {
    name: 'app',
    data () {
      return {
        fields: { value: 'FirstName' },
        waterMark: 'Find an employee',
        sortOrder: 'Ascending',
        data: remoteData,
        iTemplate: function(e) {
          return {
            template: itemVue
          };
        },
        hTemplate: function(e) {
          return {
            template: headerVue
          };
        },
        query: new Query().from('Employees').select(['FirstName',
'City', 'EmployeeID']).take(6),
      }
    }
  }
</script>
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-inputs/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-
dropdowns/styles/material.css";
#app {
  color: #008cff;
  height: 40px;
  position: absolute;
  width: 90%;
  top: 10%;
}
.autocomplete {
  width: 30%;
  margin: 0 auto;
}
.city{
  right: 15px;
  position: absolute;
}
</style>

```

{% previewsample "page.domainurl/code-snippet/auto-complete/getting-started-cs22" %}

### Footer template

The AutoComplete has options to show a footer element at the bottom of the list items in the suggestion list. Here, you can place any custom element as a footer element using [footerTemplate](#) property.



In the following sample, footer element displays the total number of list items present in the AutoComplete.

#### APP.VUE

```
<template>
  <div id="app">
    <div class='autocomplete'>
      <ejs-autocomplete ref='instance' :footerTemplate='fTemplate'
id='employees' :dataSource='data' :placeholder='waterMark'
:open='onopen'></ejs-autocomplete>
    </div>
  </div>
</template>
<script>
import Vue from 'vue';
import { AutoCompletePlugin } from '@syncfusion/ej2-vue-dropdowns';
Vue.use(AutoCompletePlugin);
var footVue = Vue.component("footerTemplate", {
  template: `<span class='foot'></span>`,
  data() {
    return {
      data: {}
    };
  }
});
export default {
  name: 'app',
  mounted() {
    document.getElementsByClassName('e-
autocomplete')[0].addEventListener('keyup', (e) => {
      this.onopen();
    })
  },
  data () {
    return {
      waterMark: 'Find a game',
      sortOrder: 'Ascending',
      data: ['Badminton', 'Basketball', 'Cricket',
        'Football', 'Golf', 'Gymnastics',
        'Hockey', 'Rugby', 'Snooker', 'Tennis'
      ],
      fTemplate: function(e) {
        return {
          template: footVue
        };
      },
    },
  },
  methods: {
    onopen: function() {
      var count = this.$refs.instance.getItems().length;
      var ele = document.getElementsByClassName('foot')[0];
      ele.innerHTML = "Total list item: " + count;
    }
  }
}
```

```

</script>
<style>
@import
"https://ej2.syncfusion.com/vue/documentation/../../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../../node_modules/@syncfusion/ej2-inputs/styles/material.css";
@import "../../node_modules/@syncfusion/ej2-vue-
dropdowns/styles/material.css";
#app {
  color: #008cff;
  height: 40px;
  position: absolute;
  width: 90%;
  top: 10%;
}
.autocomplete {
  width: 30%;
  margin: 0 auto;
}
.foot {
  text-indent: 1.2em;
  display: block;
  font-size: 15px;
  line-height: 40px;
  border-top: 1px solid #e0e0e0;
}
</style>

```

{% previewsample "page.domainurl/code-snippet/auto-complete/getting-started-cs23" %}

### No records template

The AutoComplete is provided with support to custom design the popup list content when no data is found and no matches found on search with the help of [noRecordsTemplate](#) property.

In the following sample, popup list content displays the notification of no data available.

### APP.VUE

```

<template>
  <div id="app">
    <div class='autocomplete'>
      <ejs-autocomplete ref='instance' id='employees' :dataSource='data'
      :placeholder='waterMark' :noRecordsTemplate='nTemplate'></ejs-autocomplete>
    </div>
  </div>
</template>
<script>
import Vue from 'vue';
import { AutoCompletePlugin } from '@syncfusion/ej2-vue-dropdowns';
Vue.use(AutoCompletePlugin);
var nVue = Vue.component("noRecordsTemplate", {
  template: `<span class="norecord"> NO DATA AVAILABLE</span>`,
  data() {
    return {
      data: {}
    };
  }
});

```

```

    }
  });
  export default {
    name: 'app',
    data () {
      return {
        waterMark: 'Find an item',
        data: [],
        nTemplate: function(e) {
          return {
            template: nVue
          };
        },
      }
    }
  }
}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-inputs/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-
dropdowns/styles/material.css";
#app {
  color: #008cff;
  height: 40px;
  position: absolute;
  width: 90%;
  top: 10%;
}
.autocomplete {
  width: 30%;
  margin: 0 auto;
}
</style>

```

{% previewsample "page.domainurl/code-snippet/auto-complete/getting-started-cs24" %}

### Action failure template

There is also an option to custom design the popup list content when the data fetch request fails at the remote server. This can be achieved using the `actionFailureTemplate` property.

In the following sample, when the data fetch request fails, the AutoComplete displays the notification.

### APP.VUE

```

<template>
  <div id="app">
    <div class='autocomplete'>
      <ejs-autocomplete ref='instance' :fields='fields' :query='query'
id='employees' :dataSource='data' :placeholder='waterMark'
:actionFailureTemplate='actionTemplate'></ejs-autocomplete>
    </div>
  </div>
</template>
<script>
import Vue from 'vue';

```

```

import { AutoCompletePlugin } from '@syncfusion/ej2-vue-dropdowns';
import { Query, DataManager, ODataV4Adaptor } from '@syncfusion/ej2-data';
Vue.use(AutoCompletePlugin);
var actionVue = Vue.component("actionFailureTemplate", {
  template: `<span class="action-failure"> Data fetch got failed</span>`,
  data() {
    return {
      data: {}
    };
  }
});
var remoteData = new DataManager({
  url: 'https://services.odata.org/V4/Northwind/Northwind.svcs',
  adaptor: new ODataV4Adaptor,
  crossDomain: true
});
export default {
  name: 'app',
  data () {
    return {
      waterMark: 'Find an employee',
      data: remoteData,
      actionTemplate: function(e) {
        return {
          template: actionVue
        };
      },
      query: new
Query().from('Employees').select(['FirstName']).take(6),
      fields: { value: 'FirstName' }
    }
  }
}
</script>
<style>
@import "../../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../../node_modules/@syncfusion/ej2-inputs/styles/material.css";
@import "../../node_modules/@syncfusion/ej2-vue-
dropdowns/styles/material.css";
#app {
  color: #008cff;
  height: 40px;
  position: absolute;
  width: 90%;
  top: 10%;
}
.autocomplete {
  width: 30%;
  margin: 0 auto;
}
</style>

```

{% previewsample "page.domainurl/code-snippet/auto-complete/getting-started-cs25" %}

## See Also

- [How to achieve filtering](#)
- [How to group the data using header](#)
- [How to show the list items with icon](#)

## Virtualization in AutoComplete Component

AutoComplete virtualization is a technique used to efficiently render extensive lists of items while minimizing the impact on performance. This method is particularly advantageous when dealing with large datasets because it ensures that only a fixed number of DOM (Document Object Model) elements are created. When scrolling through the list, existing DOM elements are reused to display relevant data instead of generating new elements for each item. This recycling process is managed internally.

During virtual scrolling, the data retrieved from the data source depends on the popup height and the calculation of the list item height. Enabling the [enableVirtualization](#) option in a AutoComplete activates this virtualization technique.

When fetching data from the data source, the [actionBegin](#) event is triggered before data retrieval begins. Then, the [actionComplete](#) event is triggered once the data is successfully fetched.

When the `enableVirtualization` property is enabled, the `skip` and `take` properties provided by the user within the Query class at the initial state or during the `actionBegin` or `actionComplete` events will not be considered, since it is internally managed and calculated based on certain dimensions with respect to the popup height.

## Binding local data

The AutoComplete can generate its list items through an array of complex data. For this, the appropriate columns should be mapped to the [fields](#) property. When using virtual scrolling, the list updates based on the scroll offset value, triggering a request to fetch more data from the server. As the data is being fetched, the `actionBegin` event occurs before the data retrieval starts. Once the data retrieval is successful, the `actionComplete` event is triggered, indicating that the data fetch process is complete.

In the following example, `text` column from complex data have been mapped to the `value` field.

**APP.VUE**

```
<template>
<div id="app">
  <div id="wrapper1">
    <ejs-autocomplete id='autocomplete' :dataSource='itemData'
placeholder='e.g Item 1' :fields='fields' :enableVirtualization='true'
popupHeight="200px"></ejs-autocomplete>
  </div>
</div>
</template>
<script>
import { AutoCompleteComponent, VirtualScroll } from "@syncfusion/ej2-vue-
dropdowns";
let records = [];
function dataSource() {
  for (let i = 1; i <= 150; i++) {
    let item = {
      id: 'id' + i,
```

```

        text: "Item " + i,
    };
    records.push(item);
}
}
dataSource();
//Component registration
export default {
    name: 'App',
    components: {
        "ejs-autocomplete": AutoCompleteComponent
    },
    data () {
        return {
            itemData: records,
            fields: { value: 'text' }
        }
    },
    provide: {
        autocomplete: [VirtualScroll]
    }
}
</script>
<style>
@import "../../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../../node_modules/@syncfusion/ej2-inputs/styles/material.css";
@import "../../node_modules/@syncfusion/ej2-vue-
dropdowns/styles/material.css";
@import "../../node_modules/@syncfusion/ej2-
notifications/styles/material.css";
#wrapper1{
    min-width: 250px;
    float: left;
    margin-left: 350px;
}
#wrapper2{
    min-width: 250px;
    float: right;
    margin-right: 100px;
}
</style>

```

{% previewsample "page.domainurl/code-snippet/auto-complete/virtual-scroll" %}

### Binding remote data

The AutoComplete supports retrieval of data from remote data services with the help of **DataManager** component. When using remote data, it initially fetches all the data from the server, triggering the **actionBegin** and **actionComplete** events, and then stores the data locally. During virtual scrolling, additional data is retrieved from the locally stored data, triggering the **actionBegin** and **actionComplete** events at that time as well.

The following sample displays the OrderId from the **Orders** Data Service.

### APP.VUE

```

<template>
<div id="app">
  <div id="wrapper1">
    <ejs-autocomplete id='autocomplete' :dataSource='itemData'
placeholder='e.g Item 1' :fields='fields' :enableVirtualization='true'
popupHeight="200px"></ejs-autocomplete>
  </div>
</div>
</template>
<script>
import { AutoCompleteComponent, VirtualScroll } from "@syncfusion/ej2-vue-
dropdowns";
import { Query, DataManager, WebApiAdaptor } from '@syncfusion/ej2-data';
var remoteData = new DataManager({
  url: 'https://services.syncfusion.com/js/production/api/orders',
  adaptor: new WebApiAdaptor,
  crossDomain: true
});
//Component registration
export default {
  name: 'App',
  components: {
    "ejs-autocomplete": AutoCompleteComponent
  },
  data () {
    return {
      itemData: remoteData,
      fields: { text: 'OrderID', value: 'OrderID' },
    },
    provide: {
      autocomplete: [VirtualScroll]
    }
  }
}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-inputs/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-
dropdowns/styles/material.css";
@import "../node_modules/@syncfusion/ej2-
notifications/styles/material.css";
#wrapper1{
  min-width: 250px;
  float: left;
  margin-left: 350px;
}
#wrapper2{
  min-width: 250px;
  float: right;
  margin-right: 100px;
}
</style>

```

{% previewsample "page.domainurl/code-snippet/auto-complete/virtual-scroll-remote" %}

## Grouping

The AutoComplete component supports grouping with Virtualization. It allows you to organize elements into groups based on different categories. Each item in the list can be classified using the [groupBy](#) field in the data table. After grouping, virtualization works similarly to local data binding, providing a seamless user experience. When the data source is bound to remote data, an initial request is made to retrieve all data for the purpose of grouping. Subsequently, the grouped data works in the same way as local data binding virtualization, enhancing performance and responsiveness.

The following sample shows the example for Grouping with Virtualization.

### APP.VUE

```
<template>
<div id="app">
  <div id="wrapper1">
    <ejs-autocomplete id='autocomplete' :dataSource='itemData'
placeholder='e.g Item 1' :fields='fields' :enableVirtualization='true'
popupHeight="200px"></ejs-autocomplete>
  </div>
</div>
</template>
<script>
import { AutoCompleteComponent, VirtualScroll } from "@syncfusion/ej2-vue-
dropdowns";
let records = [];
function dataSource() {
  for (var i = 1; i <= 150; i++) {
    var item = {};
    item.id = 'id' + i;
    item.text = "Item ".concat(i);
    var randomGroup = Math.floor(Math.random() * 4) + 1;
    switch (randomGroup) {
      case 1:
        item.group = 'Group A';
        break;
      case 2:
        item.group = 'Group B';
        break;
      case 3:
        item.group = 'Group C';
        break;
      case 4:
        item.group = 'Group D';
        break;
      default:
        break;
    }
    records.push(item);
  }
}
dataSource();
//Component registration
export default {
  name: 'App',
  components: {
    "ejs-autocomplete": AutoCompleteComponent
```



```

    },
    data () {
      return {
        itemData: records,
        fields: { groupBy: 'group', value: 'text' }
      }
    },
    provide: {
      autocomplete: [VirtualScroll]
    }
  }
</script>
<style>
@import ".../node_modules/@syncfusion/ej2-base/styles/material.css";
@import ".../node_modules/@syncfusion/ej2-inputs/styles/material.css";
@import ".../node_modules/@syncfusion/ej2-vue-
dropdowns/styles/material.css";
@import ".../node_modules/@syncfusion/ej2-
notifications/styles/material.css";
#wrapper1{
  min-width: 250px;
  float: left;
  margin-left: 350px;
}
#wrapper2{
  min-width: 250px;
  float: right;
  margin-right: 100px;
}
</style>

```

{% previewsample "page.domainurl/code-snippet/auto-complete/virtual-scroll-group" %}

### Grouping in Vue Auto complete component

The AutoComplete supports wrapping nested elements into a group based on different categories. The category of each list item can be mapped through the [Link to the Video](#) field in the data table. The group header is displayed as both inline and fixed headers. The fixed group header content is updated dynamically on scrolling the suggestion list with its category value.

To group the Vue AutoComplete items, you can check on this video:

In the following sample, vegetables are grouped according on its category using groupBy field.

#### APP.VUE

```

<template>
  <div id="app">
    <ejs-autocomplete :dataSource='vegetableData' :fields='fields'
:placeholder="waterMark" ></ejs-autocomplete>
  </div>
</template>
<script>
import Vue from 'vue';
import { AutoCompletePlugin } from '@syncfusion/ej2-vue-dropdowns';
Vue.use(AutoCompletePlugin);
export default {

```

```

name: 'app',
data () {
  return {
    waterMark : 'Find a vegetable',
    vegetableData: [
      { Vegetable: 'Cabbage', Category: 'Leafy and Salad', Id: 'item1' },
      { Vegetable: 'Spinach', Category: 'Leafy and Salad', Id: 'item2' },
      { Vegetable: 'Wheat grass', Category: 'Leafy and Salad', Id: 'item3' },
      { Vegetable: 'Yarrow', Category: 'Leafy and Salad', Id: 'item4' },
      { Vegetable: 'Pumpkins', Category: 'Leafy and Salad', Id: 'item5' },
      { Vegetable: 'Chickpea', Category: 'Beans', Id: 'item6' },
      { Vegetable: 'Green bean', Category: 'Beans', Id: 'item7' },
      { Vegetable: 'Horse gram', Category: 'Beans', Id: 'item8' },
      { Vegetable: 'Garlic', Category: 'Bulb and Stem', Id: 'item9' },
      { Vegetable: 'Nopal', Category: 'Bulb and Stem', Id: 'item10' },
      { Vegetable: 'Onion', Category: 'Bulb and Stem', Id: 'item11' }
    ],
    fields: { groupBy: 'Category', value: 'Vegetable' }
  }
}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-inputs/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-
dropdowns/styles/material.css";
#app {
  color: #008cff;
  height: 40px;
  left: 35%;
  position: absolute;
  top: 10%;
  width: 30%;
}
</style>

```

{% previewsample "page.domainurl/code-snippet/auto-complete/getting-started-cs14" %}

See Also

- [Group Template support to AutoComplete](#)[Link to the Video](#).

### Filtering in Vue Auto complete component

The AutoComplete has built-in support to filter data items. The filter operation starts as soon as you start typing characters in the component.

To filter the Vue AutoComplete items, you can check on this video:

#### Change the filter type

Determines on which filter type, the component needs to be considered on search action. The available [filterType](#) and its supported data types are

Filter Type	Description	Supported Types
-------------	-------------	-----------------

|-----|-----|-----|

| **StartsWith** | Checks whether a value begins with the specified value. | String |

| **EndsWith** | Checks whether a value ends with specified value. | String |

| **Contains** | Checks whether a value contains with specified value. | String |

The following examples shows the data filtering is done with **StartsWith** type.

### APP.VUE

```
<template>
  <div id="app">
    <div class='autocomplete'>
      <ejs-autocomplete :dataSource='data' :fields='fields'
sortOrder='sortOrder' :filterType='filterType' :query='query'
:placeholder="waterMark" ></ejs-autocomplete>
    </div>
  </div>
</template>
<script>
import Vue from 'vue';
import { AutoCompletePlugin } from '@syncfusion/ej2-vue-dropdowns';
import { Query, DataManager, ODataV4Adaptor } from '@syncfusion/ej2-data';
Vue.use(AutoCompletePlugin);
var remoteData = new DataManager({
  url: 'https://services.odata.org/V4/Northwind/Northwind.svc/Customers',
  adaptor: new ODataV4Adaptor,
  crossDomain: true
});
export default {
  name: 'app',
  data () {
    return {
      waterMark : 'Find a customer',
      data: remoteData,
      fields: { value: 'ContactName' },
      query: new Query().select(['ContactName']),
      sortOrder: 'Ascending',
      filterType: 'StartsWith'
    }
  }
}
</script>
<style>
@import "../../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../../node_modules/@syncfusion/ej2-inputs/styles/material.css";
@import "../../node_modules/@syncfusion/ej2-vue-
dropdowns/styles/material.css";
#app {
  color: #008c8f;
  height: 40px;
  position: absolute;
  top: 10%;
  width: 90%;
}
.autocomplete {
```

```
width: 30%;
margin: 0 auto;
}
</style>
```

{% previewsample "page.domainurl/code-snippet/auto-complete/getting-started-cs6" %}

### Filter item count

You can specify the filter suggestion item count through [suggestionCount](#) property of AutoComplete.

The following example, to restrict the suggestion list item counts as 5.

### APP.VUE

```
<template>
  <div id="app">
    <div class='autocomplete'>
      <ejs-autocomplete :dataSource='data' :fields='fields'
sortOrder='sortOrder'
      :query='query' :suggestionCount='suggestionCount'
:placeholder="waterMark"></ejs-autocomplete>
    </div>
  </div>
</template>
<script>
import Vue from 'vue';
import { AutoCompletePlugin } from '@syncfusion/ej2-vue-dropdowns';
import { Query, DataManager, ODataV4Adaptor } from '@syncfusion/ej2-data';
Vue.use(AutoCompletePlugin);
var remoteData = new DataManager({
  url: 'https://services.odata.org/V4/Northwind/Northwind.svc/Customers',
  adaptor: new ODataV4Adaptor,
  crossDomain: true
});
export default {
  name: 'app',
  data () {
    return {
      waterMark : 'Find a customer',
      data: remoteData,
      fields: { value: 'ContactName' },
      query: new Query().select(['ContactName', 'CustomerID']),
      sortOrder: 'Ascending',
      suggestionCount: 5
    }
  }
}
</script>
<style>
@import "../../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../../node_modules/@syncfusion/ej2-inputs/styles/material.css";
@import "../../node_modules/@syncfusion/ej2-vue-
dropdowns/styles/material.css";
#app {
  color: #008cff;
  height: 40px;
  position: absolute;
```

```

    top: 10%;
    width: 90%;
  }
  .autocomplete {
    width: 30%;
    margin: 0 auto;
  }
</style>

```

{% previewsample "page.domainurl/code-snippet/auto-complete/getting-started-cs7" %}

### Limit the minimum filter character

You can set the limit for the character count to filter the data on the AutoComplete. This can be done by set the [minLength](#) property to AutoComplete.

In the following example, the remote request doesn't fetch the search data, until the search key contains three characters.

### APP.VUE

```

<template>
  <div id="app">
    <div class='autocomplete'>
      <ejs-autocomplete :dataSource='data' :fields='fields'
sortOrder='sortOrder'
      :query='query' :filterType='filterType' :minLength='minLength'
      :placeholder="waterMark"></ejs-autocomplete>
    </div>
  </div>
</template>
<script>
import Vue from 'vue';
import { AutoCompletePlugin } from '@syncfusion/ej2-vue-dropdowns';
import { Query, DataManager, ODataV4Adaptor } from '@syncfusion/ej2-data';
Vue.use(AutoCompletePlugin);
var remoteData = new DataManager({
  url: 'https://services.odata.org/V4/Northwind/Northwind.svc/Customers',
  adaptor: new ODataV4Adaptor,
  crossDomain: true
});
export default {
  name: 'app',
  data () {
    return {
      waterMark : 'Find a customer',
      data: remoteData,
      fields: { value: 'ContactName' },
      query: new Query().select(['ContactName', 'CustomerID']),
      sortOrder: 'Ascending',
      filterType: 'StartsWith',
      minLength: 3
    }
  }
}
</script>
<style>

```

```

@import "../../../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../../../node_modules/@syncfusion/ej2-inputs/styles/material.css";
@import "../../../node_modules/@syncfusion/ej2-vue-
dropdowns/styles/material.css";
#app {
  color: #008cff;
  height: 40px;
  position: absolute;
  top: 10%;
  width: 90%;
}
.autocomplete {
  width: 30%;
  margin: 0 auto;
}
</style>

```

{% previewsample "page.domainurl/code-snippet/auto-complete/getting-started-cs8" %}

### Case sensitive filtering

Data items can be filtered either with or without case sensitivity using the **DataManager**. This can be done by setting the [ignoreCase](#) property of AutoComplete.

The following sample depicts how to filter the data with case-sensitive.

### APP.VUE

```

<template>
  <div id="app">
    <div class='autocomplete'>
      <ejs-autocomplete :dataSource='data' :fields='fields'
      sortOrder='sortOrder' :query='query' :filterType='filterType'
      :ignoreCase='ignoreCase' :placeholder="waterMark" ></ejs-autocomplete>
    </div>
  </div>
</template>
<script>
import Vue from 'vue';
import { AutoCompletePlugin } from '@syncfusion/ej2-vue-dropdowns';
import { Query, DataManager, ODataV4Adaptor } from '@syncfusion/ej2-data';
Vue.use(AutoCompletePlugin);
var remoteData = new DataManager({
  url: 'https://services.odata.org/V4/Northwind/Northwind.svc/Customers',
  adaptor: new ODataV4Adaptor,
  crossDomain: true
});
export default {
  name: 'app',
  data () {
    return {
      waterMark : 'Find a customer',
      data: remoteData,
      fields: { value: 'ContactName' },
      query: new Query().select(['ContactName']),
      sortOrder: 'Ascending',
      filterType: 'StartsWith',

```

```

        ignoreCase: false
      }
    }
  }
</script>
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-inputs/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-
dropdowns/styles/material.css";
#app {
  color: #008cff;
  height: 40px;
  position: absolute;
  top: 10%;
  width: 90%;
}
.autocomplete {
  width: 30%;
  margin: 0 auto;
}
</style>

```

{% previewsample "page.domainurl/code-snippet/auto-complete/getting-started-cs9" %}

### Diacritics Filtering

An AutoComplete supports diacritics filtering which will ignore the diacritics and makes it easier to filter the results in international characters lists when the [ignoreAccent](#) is enabled.

In the following sample, data with diacritics are bound as dataSource for AutoComplete.

### APP.VUE

```

<template>
  <div id="app">
    <div class='autocomplete'>
      <ejs-autocomplete :dataSource='data' :ignoreAccent='ignoreAccent'
        :placeholder="waterMark" ></ejs-autocomplete>
    </div>
  </div>
</template>
<script>
import Vue from 'vue';
import { AutoCompletePlugin } from '@syncfusion/ej2-vue-dropdowns';
Vue.use(AutoCompletePlugin);
export default {
  name: 'app',
  data () {
    return {
      data: [
        'Aeróbics',
        'Aeróbics en Agua',
        'Aerografía',
        'Aeromodelaje',
        'Águilas',
        'Ajedrez',

```

```

        'Ala Delta',
        'Álbumes de Música',
        'Alusivos',
        'Análisis de Escritura a Mano'
    ],
    ignoreAccent: true,
    waterMark: 'e.g: aero'
}
}
}
</script>
<style>
@import ".../node_modules/@syncfusion/ej2-base/styles/material.css";
@import ".../node_modules/@syncfusion/ej2-inputs/styles/material.css";
@import ".../node_modules/@syncfusion/ej2-vue-
dropdowns/styles/material.css";
#app {
    color: #008cff;
    height: 40px;
    position: absolute;
    top: 10%;
    width: 90%;
}
.autocomplete {
    width: 30%;
    margin: 0 auto;
}
</style>

```

{% previewsample "page.domainurl/code-snippet/auto-complete/getting-started-cs10" %}

See Also

- [How to achieve autofill while filtering](#)
- [How to group the data using header](#)
- [How to highlight the search data](#)

## Localization in Vue Auto complete component

The Localization library allows you to localize static text content of the noRecordsTemplate and actionFailureTemplate properties according to the culture currently assigned to the AutoComplete.

| Locale key | en-US (default) |

|-----|-----|

| noRecordsTemplate | No Records Found |

| actionFailureTemplate | The Request Failed |

### Loading translations

To load translation object to your application, use load function of the L10n class. In the following sample, French culture is set to the AutoComplete and no data is loaded.

Hence, the noRecordsTemplate property displays its text in French culture initially and if the sample is run offline, the actionFailureTemplate property displays its text appropriately.



**APP.VUE**

```

<template>
  <div id="app">
    <ejs-autocomplete :dataSource='data' :locale='locale' :fields='fields'
    sortOrder='sortOrder' :query='query' :placeholder="waterMark" ></ejs-
    autocomplete>
  </div>
</template>
<script>
import Vue from 'vue';
import { AutoCompletePlugin } from '@syncfusion/ej2-vue-dropdowns';
import { loadCldr, L10n } from '@syncfusion/ej2-base';
import { Query, DataManager, ODataV4Adaptor } from '@syncfusion/ej2-data';
Vue.use(AutoCompletePlugin);
var remoteData = new DataManager({
  url: 'https://services.odata.org/V4/Northwind/Northwind.svc/Customers',
  adaptor: new ODataV4Adaptor,
  crossDomain: true
});
L10n.load({
  'fr-BE': {
    'dropdowns': {
      noRecordsTemplate: "Aucun enregistrement trouvé",
      actionFailureTemplate: "Modèle d'échec d'action"
    }
  },
});
export default {
  name: 'app',
  data () {
    return {
      waterMark : 'Trouver un client',
      query: new Query().select(['ContactName', 'CustomerID']),
      data: remoteData,
      locale: 'fr-BE',
      fields: { value: 'ContactName' },
      sortOrder: 'Ascending'
    }
  }
}
</script>
<style>
@import ".../node_modules/@syncfusion/ej2-base/styles/material.css";
@import ".../node_modules/@syncfusion/ej2-inputs/styles/material.css";
@import ".../node_modules/@syncfusion/ej2-vue-
dropdowns/styles/material.css";
#app {
  color: #008cff;
  height: 40px;
  left: 35%;
  position: absolute;
  top: 15%;
  width: 30%;
}
</style>

```

```
{% previewsample "page.domainurl/code-snippet/auto-complete/getting-started-cs18" %}
```

See Also

- [Accessibility](#)
- [How to bind the data to the autocomplete](#)

### Style in Vue Auto complete component

The following content provides the exact CSS structure that can be used to modify the control's appearance based on the user preference.

#### Customizing the appearance of wrapper element

Use the following CSS to customize the appearance of wrapper element.

```
,
.e-ddl.e-input-group.e-control-wrapper .e-input {
font-size: 20px;
font-family: emoji;
color: #ab3243;
background: #32a5ab;
}
,
```

#### Customizing the dropdown icon's color

Use the following CSS to customize the dropdown icon's color.

```
,
.e-ddl.e-input-group .e-input-group-icon,.e-ddl.e-input-group.e-control-wrapper .e-input-group-
icon:hover {
color: #bb233d;
font-size: 13px;
}
,
```

#### Customizing the focus color

Use the following CSS to customize the focusing color of input element.

```
,
.e-ddl.e-input-group.e-control-wrapper.e-input-focus::before, .e-ddl.e-input-group.e-control-wrapper.e-
input-focus::after {
background: #c000ff;
}
,
```

### Customizing the outline theme's focus color

Use the following CSS to customize the focusing color of outline theme.

```
,
.e-outline.e-input-group.e-input-focus:hover:not(.e-success):not(.e-warning):not(.e-error):not(.e-disabled):not(.e-float-icon-left),.e-outline.e-input-group.e-input-focus.e-control-wrapper:hover:not(.e-success):not(.e-warning):not(.e-error):not(.e-disabled):not(.e-float-icon-left),.e-outline.e-input-group.e-input-focus:not(.e-success):not(.e-warning):not(.e-error):not(.e-disabled),.e-outline.e-input-group.e-control-wrapper.e-input-focus:not(.e-success):not(.e-warning):not(.e-error):not(.e-disabled) {
border-color: #b1bd15;
box-shadow: inset 1px 1px #b1bd15, inset -1px 0 #b1bd15, inset 0 -1px #b1bd15;
}
,
```

### Customizing the disabled component's text color

Use the following CSS to customize the text color when the component is disabled.

```
,
.e-input-group.e-control-wrapper .e-input[disabled] {
-webkit-text-fill-color: #0d9133;
}
,
```

### Customizing the float label element's focusing color

Use the following CSS to customize the focusing color of float label element.

```
,
.e-float-input.e-input-group:not(.e-float-icon-left) .e-float-line::before,.e-float-input.e-control-wrapper.e-input-group:not(.e-float-icon-left) .e-float-line::before,.e-float-input.e-input-group:not(.e-float-icon-left) .e-float-line::after,.e-float-input.e-control-wrapper.e-input-group:not(.e-float-icon-left) .e-float-line::after {
background-color: #2319b8;
}
.e-ddl.e-input-group.e-control-wrapper.e-float-input.e-input-focus .e-float-text.e-label-top, .e-float-input.e-control-wrapper:not(.e-error).e-input-focus input ~ label.e-float-text {
color: #2319b8;
}
,
```

### Customizing the color of the placeholder text

Use the following CSS to customize the text color of placeholder.

```
,
.e-ddl.e-input-group input.e-input::placeholder {
```

```
color: red;
}
```

### Customizing the text selection color

Use the following CSS to customize the selection color of text and background.

```
.e-ddl.e-input-group input.e-input::selection {
color: red;
background: yellow;
}
```

### Customizing the background color of focus, hover, and active item's

Use the following CSS to customize the background color of focus, hover and active item's.

```
.e-dropdownbase .e-list-item.e-item-focus, .e-dropdownbase .e-list-item.e-active, .e-dropdownbase .e-list-item.e-active.e-hover, .e-dropdownbase .e-list-item.e-hover {
background-color: #1f9c99;
color: #2319b8;
}
```

### Customizing the appearance of pop-up element

Use the following CSS to customize the appearance of popup element.

```
.e-dropdownbase .e-list-item, .e-dropdownbase .e-list-item.e-item-focus {
background-color: #29c2b8;
color: #207cd9;
font-family: emoji;
min-height: 29px;
}
```

### Adding mandatory asterisk to placeholder and float label

You can add a mandatory asterisk(\*) to placeholder and float label using `<b>.e-input-group.e-control-wrapper.e-float-input .e-float-text::after</b>` class.

### APP.VUE

```
<template>
```

```

    <div id="app">
      <ejs-autocomplete :dataSource='sportsData' :placeholder="waterMark"
      :floatLabelType= "auto"></ejs-autocomplete>
    </div>
  </template>
  <script>
import Vue from 'vue';
import { AutoCompletePlugin } from '@syncfusion/ej2-vue-dropdowns';
Vue.use(AutoCompletePlugin);
export default {
  name: 'app',
  data () {
    return {
      waterMark : 'e.g. Basketball',
      sportsData: ['American Football', 'Badminton', 'Basketball',
'Cricket',
                    'Football', 'Golf', 'Gymnastics',
                    'Hockey', 'Rugby', 'Snooker', 'Tennis'
                ]
    }
  }
}
  </script>
  <style>
@import "../../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../../node_modules/@syncfusion/ej2-inputs/styles/material.css";
@import "../../node_modules/@syncfusion/ej2-vue-
dropdowns/styles/material.css";
#app {
  color: #008cff;
  height: 40px;
  left: 35%;
  position: absolute;
  top: 35%;
  width: 30%;
}
.e-input-group.e-control-wrapper.e-float-input .e-float-text::after {
  content: '*';
  color: red;
}
  </style>

```

{% previewsample "page.domainurl/code-snippet/auto-complete/getting-started-cs19" %}

### Accessibility in Vue Auto complete component

The AutoComplete component followed the accessibility guidelines and standards, including [ADA](#), [Section 508](#), [WCAG 2.2](#) standards, and [WCAG roles](#) that are commonly used to evaluate accessibility.

The accessibility compliance for the AutoComplete component is outlined below.

| Accessibility Criteria | Compatibility |

| -- | -- |

| [WCAG 2.2 Support](#) |  |

| [Section 508 Support](#) |  |

| [Screen Reader Support](#) |  |

| [Right-To-Left Support](#) |  |

| [Color Contrast](#) |  |

| [Mobile Device Support](#) |  |

| [Keyboard Navigation Support](#) |  |

| [Accessibility Checker Validation](#) |  |

| [Axe-core Accessibility Validation](#) |  |

<style>

```
.post .post-content img {
display: inline-block;
margin: 0.5em 0;
}
```

</style>

<div> - All features of the component meet the requirement.</div>

<div> - Some features of the component do not meet the requirement.</div>

<div> - The component does not meet the requirement.</div>

### WAI-ARIA attributes

The AutoComplete component has been designed, keeping in mind the WAI-ARIA specifications, and applies the WAI-ARIA roles, states, and properties along with keyboard support. This component is characterized by complete keyboard interaction support and ARIA accessibility support that makes it easy for people who use assistive technologies (AT) or those who completely rely on keyboard navigation.

### ARIA attributes

The AutoComplete component uses the **combobox** role and each list item has an **option** role. The following ARIA Attributes denote the AutoComplete state.

| **Property** | **Functionalities** |

| --- | --- |

| aria-haspopup | Indicates whether the AutoComplete input element has a suggestion list or not. |

| aria-expanded | Indicates whether the suggestion list has expanded or not. |

| aria-selected | Indicates the selected option from the list. |

| aria-readonly | Indicates the readonly state of the AutoComplete element. |

| aria-disabled | Indicates whether the AutoComplete component is in a disabled state or not. |

| aria-activedescendent | This attribute holds the ID of the active list item to focus its descendant child element. |

| aria-owns | This attribute contains the ID of the suggestion list to indicate popup as a child element. |

| aria-autocomplete | This attribute contains the 'both' to a list of options shows and the currently selected suggestion also shows inline. |

### Keyboard interaction

You can use the following key shortcuts to access the AutoComplete without interruptions.

| **Keyboard shortcuts** | **Actions** |

| --- | --- |

| Arrow Down | In popup hidden state, opens the suggestion list. In popup open state, selects the first item when no item selected else selects the item next to the currently selected item. |

| Arrow Up | In popup hidden state, opens the suggestion list. In popup open state, selects the last item when no item selected else selects the item previous to the currently selected one. |

| Page Down | Scrolls down to the next page and selects the first item when popup list opens. |

| Page Up | Scrolls up to previous page and select the first item when popup list open. |

| Enter | Selects the focused item and set to AutoComplete component. |

| Tab | Focuses on the next tab indexed element when the popup is closed. Otherwise, closes the popup list and remains the focus in component suppose if it is in an open state. |

| Shift + tab | Focuses the previous tab indexed element when the popup is closed. Otherwise, closes the popup list and remains the focus in component suppose if it is in an open state. |

| Alt + Down | Opens the popup list. |

| Alt + Up | In popup hidden state, opens the popup list. In popup open state, closes the popup list. |

| Esc(Escape) | Closes the popup list when it is in an open state then remove the selection. |

| Home | Cursor moves to before of first character in input. |

| End | Cursor moves to next of last character in input. |

In the below sample, focus the AutoComplete component using alt+t keys.

### APP.VUE

```

<template>
  <div id="app">
    <ejs-autocomplete ref='instance' :dataSource='sportsData'
:fields='fields' :placeholder="waterMark" ></ejs-autocomplete>
  </div>
</template>
<script>
import Vue from 'vue';
import { AutoCompletePlugin } from '@syncfusion/ej2-vue-dropdowns';
Vue.use(AutoCompletePlugin);
export default {
  name: 'app',
  data () {
    return {
      waterMark : 'Find a game',
      sportsData: [
        { Id: 'Game1', Game: 'Badminton' },
        { Id: 'Game2', Game: 'Basketball' },
        { Id: 'Game3', Game: 'Cricket' },
        { Id: 'Game4', Game: 'Football' },
        { Id: 'Game5', Game: 'Golf' },
        { Id: 'Game6', Game: 'Hockey' },
        { Id: 'Game7', Game: 'Rugby' },
        { Id: 'Game8', Game: 'Snooker' }
      ],
      fields: { value: 'Game' }
    }
  },
  methods: {
    keylistener: function(evt) {
      if (event.altKey && event.keyCode === 84 /* t */) {
        // press alt+t to focus the control.
        this.$refs.instance.focusIn();
      }
    },
    created: function() {
      document.addEventListener('keyup', this.keylistener);
    },
  }
}
</script>
<style>
@import "../../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../../node_modules/@syncfusion/ej2-inputs/styles/material.css";
@import "../../node_modules/@syncfusion/ej2-vue-
dropdowns/styles/material.css";
#app {
  color: #008cff;
  height: 40px;
  left: 35%;
  position: absolute;
  top: 35%;
  width: 30%;
}
</style>

```



```
{% previewsample "page.domainurl/code-snippet/auto-complete/getting-started-cs1" %}
```

### Ensuring accessibility

The AutoComplete component's accessibility levels are ensured through an [accessibility-checker](#) and [axe-core](#) software tools during automated testing.

The accessibility compliance of the AutoComplete component is shown in the following sample. Open the [sample](#) in a new window to evaluate the accessibility of the AutoComplete component with accessibility tools.

```
{% previewsample "https://ej2.syncfusion.com/accessibility/auto-complete.html" %}
```

See also

- [Accessibility in Syncfusion Vue components](#)

### Two way binding in Vue Auto complete component

Two-way binding can be achieved by using the `v-model` directive in Vue. In the following sample, when you change the value in one AutoComplete component, v-model will automatically update the value in the other AutoComplete.

The following example demonstrates how to set the `two-way-binding` in the AutoComplete.

#### APP.VUE

```
<template>
<div id="app">
  <div id="wrapper1">
    <ejs-autocomplete id='first':dataSource='sportsData'
:fields='fields' :placeholder="waterMark" v-model="value" ></ejs-
autocomplete>
  </div>
  <div id="wrapper2">
    <ejs-autocomplete id='second' :dataSource='sportsData'
:fields='fields' :placeholder="waterMark" v-model="value" ></ejs-
autocomplete>
  </div>
</div>
</template>
<script>
import Vue from 'vue';
import { AutoCompletePlugin } from '@syncfusion/ej2-vue-dropdowns';
Vue.use(AutoCompletePlugin);
export default {
  name: 'app',
  data () {
    return {
      waterMark : 'Select a game',
      value: null,
      sportsData: [
        { Id: 'Game1', Game: 'Badminton' },
        { Id: 'Game2', Game: 'Basketball' },
        { Id: 'Game3', Game: 'Cricket' },
        { Id: 'Game4', Game: 'Football' },
        { Id: 'Game5', Game: 'Golf' },
        { Id: 'Game6', Game: 'Hockey' },
      ],
    }
  }
}
```

```

    { Id: 'Game7', Game: 'Rugby' },
    { Id: 'Game8', Game: 'Snooker' }
  ],
  fields: { value: 'Game' }
}
}
</script>
<style>
@import "../../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../../node_modules/@syncfusion/ej2-inputs/styles/material.css";
@import "../../node_modules/@syncfusion/ej2-vue-
dropdowns/styles/material.css";
#wrapper1{
  min-width: 250px;
  float: left;
  margin-left: 100px;
}
#wrapper2{
  min-width: 250px;
  float: right;
  margin-right: 100px;
}
</style>

```

{% previewsample "page.domainurl/code-snippet/auto-complete/two-way-cs1" %}

## How To

### Autofill in Vue Auto complete component

The AutoComplete supports the autofill behavior with the help of [autofill](#) property. Whenever you change the input value, the AutoComplete will autocomplete your data by matching the typed character. Suppose, if no matches found then, AutoComplete doesn't suggest any item.

In the below sample, showcase that how to work autofill with AutoComplete.

### APP.VUE

```

<template>
  <div id="app">
    <ejs-autocomplete :autofill='autofill' :dataSource='searchData'
    :fields='fields' :placeholder="waterMark" ></ejs-autocomplete>
  </div>
</template>
<script>
import Vue from 'vue';
import { AutoCompletePlugin } from '@syncfusion/ej2-vue-dropdowns';
Vue.use(AutoCompletePlugin);
export default {
  name: 'app',
  data () {
    return {
      waterMark : 'Find a country',
      searchData: [
        { Name: 'Australia', Code: 'AU' },
        { Name: 'Bermuda', Code: 'BM' },
        { Name: 'Canada', Code: 'CA' },

```

```

    { Name: 'Cameroon', Code: 'CM' },
    { Name: 'Denmark', Code: 'DK' },
    { Name: 'France', Code: 'FR' },
    { Name: 'Finland', Code: 'FI' },
    { Name: 'Germany', Code: 'DE' },
    { Name: 'Greenland', Code: 'GL' },
    { Name: 'Hong Kong', Code: 'HK' },
    { Name: 'India', Code: 'IN' },
    { Name: 'Italy', Code: 'IT' },
    { Name: 'Japan', Code: 'JP' },
    { Name: 'Mexico', Code: 'MX' },
    { Name: 'Norway', Code: 'NO' },
    { Name: 'Poland', Code: 'PL' },
    { Name: 'Switzerland', Code: 'CH' },
    { Name: 'United Kingdom', Code: 'GB' },
    { Name: 'United States', Code: 'US' }
  ],
  fields: { value: 'Name' },
  autofill: true
}
}
}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-inputs/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-
dropdowns/styles/material.css";
#app {
  color: #008c9f;
  height: 40px;
  left: 35%;
  position: absolute;
  top: 15%;
  width: 30%;
}
</style>

```

{% previewsample "page.domainurl/code-snippet/auto-complete/getting-started-cs15" %}

### Icon support in Vue Auto complete component

You can render icons to the list items by mapping the [iconCss](#) field. This iconCss field create a span in the list item with mapped class name to allow styling as per your need.

In the following sample, the icon classes are mapped with iconCss field.

### APP.VUE

```

<template>
  <div id="app">
    <ejs-autocomplete :dataSource='sortFormatData' :fields='fields'
:placeholder="waterMark" ></ejs-autocomplete>
  </div>
</template>
<script>
import Vue from 'vue';

```

```

import { AutoCompletePlugin } from '@syncfusion/ej2-vue-dropdowns';
Vue.use(AutoCompletePlugin);
export default {
  name: 'app',
  data () {
    return {
      waterMark : 'Find a format',
      sortFormatData: [
        { Class: 'sort', Type: 'Sort A to Z', Id: '1' },
        { Class: 'filter', Type: 'Filter', Id: '2' },
        { Class: 'clear', Type: 'Clear', Id: '3' }
      ],
      fields: { value: 'Type', iconCss: 'Class' }
    }
  }
}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-inputs/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-dropdowns/styles/material.css";
#app {
  color: #008cff;
  height: 40px;
  left: 35%;
  position: absolute;
  top: 35%;
  width: 30%;
}
#container {
  visibility: hidden;
}
#loader {
  color: #008cff;
  height: 40px;
  width: 30%;
  position: absolute;
  top: 45%;
  left: 45%;
}
.e-list-icon{
  line-height: 1.3;
  padding-right: 10px;
  text-indent: 5px;
}
.sort:before {
  content: '\e890';
  font-family: 'e-icons';
  font-size: 15px;
}
.filter:before {
  content: '\e7ee';
  font-family: 'e-icons';
  font-size: 15px;
  opacity: 0.78;
}
.clear:before {

```

```

        content: '\e7fc';
        font-family: 'e-icons';
        font-size: 15px;
    }
</style>

```

{% previewsample "page.domainurl/code-snippet/auto-complete/getting-started-cs17" %}

### Custom search in Vue Auto complete component

The AutoComplete has built-in support to [highlight](#) the searched characters on suggested list items when enabled the highlight property.

In the below sample, customized the matched character in suggestion list by `e-highlight` class.

#### APP.VUE

```

<template>
  <div id="app">
    <ejs-autocomplete :highlight='highlight' :dataSource='searchData'
    :fields='fields' :placeholder="waterMark" ></ejs-autocomplete>
  </div>
</template>
<script>
import Vue from 'vue';
import { AutoCompletePlugin } from '@syncfusion/ej2-vue-dropdowns';
Vue.use(AutoCompletePlugin);
export default {
  name: 'app',
  data () {
    return {
      waterMark : 'Find a country',
      searchData: [
        { Name: 'Australia', Code: 'AU' },
        { Name: 'Bermuda', Code: 'BM' },
        { Name: 'Canada', Code: 'CA' },
        { Name: 'Cameroon', Code: 'CM' },
        { Name: 'Denmark', Code: 'DK' },
        { Name: 'France', Code: 'FR' },
        { Name: 'Finland', Code: 'FI' },
        { Name: 'Germany', Code: 'DE' },
        { Name: 'Greenland', Code: 'GL' },
        { Name: 'Hong Kong', Code: 'HK' },
        { Name: 'India', Code: 'IN' },
        { Name: 'Italy', Code: 'IT' },
        { Name: 'Japan', Code: 'JP' },
        { Name: 'Mexico', Code: 'MX' },
        { Name: 'Norway', Code: 'NO' },
        { Name: 'Poland', Code: 'PL' },
        { Name: 'Switzerland', Code: 'CH' },
        { Name: 'United Kingdom', Code: 'GB' },
        { Name: 'United States', Code: 'US' }
      ],
      fields: { value: 'Name' },
      highlight: true
    }
  }
}

```

```
}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-inputs/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-
dropdowns/styles/material.css";
#app {
  color: #008cff;
  height: 40px;
  left: 35%;
  position: absolute;
  top: 15%;
  width: 30%;
}
</style>
```

{% previewsample "page.domainurl/code-snippet/auto-complete/getting-started-cs16" %}

## Avatar

### Getting Started with the Vue Avatar Component in Vue 2

This article provides a step-by-step guide for setting up a Vue 2 project using [Vue-CLI](#) and integrating the Syncfusion Vue Avatar component using the [Composition API](#) / [Options API](#)[Link to the Video](#).

To get start quickly with Vue Avatar component, you can check on this video:

Prerequisites

[System requirements for Syncfusion Vue UI components](#)

### Setting up the Vue 2 project

To generate a Vue 2 project using Vue-CLI, use the [vue create](#) command. Follow these steps to install Vue CLI and create a new project:

```
`bash
```

```
npm install -g @vue/cli
```

```
vue create quickstart
```

```
cd quickstart
```

```
npm run serve
```

```
,
```

or

```
`bash
```

```
yarn global add @vue/cli
```

```
vue create quickstart
```

```
cd quickstart
```

```
yarn run serve
```

```
,
```

When creating a new project, choose the option `Default ([Vue 2] babel, eslint)` from the menu.

```
E:\samples>vue create quickstart

Vue CLI v5.0.8
? Please pick a preset:
  Default ([Vue 3] babel, eslint)
> Default ([Vue 2] babel, eslint)
  Manually select features
```

Once the `quickstart` project is set up with default settings, proceed to add Syncfusion components to the project.

#### Add Syncfusion Vue packages

Syncfusion packages are available at [npmjs.com](https://www.npmjs.com). To use Vue components, install the required npm package.

This article uses the [Vue Avatar component](#) as an example. Install the `@syncfusion/ej2-layouts` package by running the following command:

```
`bash
```

```
npm install @syncfusion/ej2-layouts --save
```

```
,
```

```
or
```

```
`bash
```

```
yarn add @syncfusion/ej2-layouts
```

```
,
```

#### Import Syncfusion CSS styles

You can import themes for the Syncfusion Vue component in various ways, such as using CSS or SASS styles from npm packages, CDN, [CRG](#) and [Theme Studio](#). Refer to [themes topic](#) to know more about built-in themes and different ways to refer to themes in a Vue project.

In this article, the `Material` theme is applied using CSS styles, which are available in installed packages. The necessary `Material` CSS styles for the Avatar component and its dependents were imported into the `<style>` section of `src/App.vue` file.

#### ~/SRC/APP.VUE

```
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-layouts/styles/material.css";
</style>
```

We can also use [CRG](#) to generate combined component styles.

### Using Avatar in Vue Application

Add an HTML span element with `e-avatar` class into the `<template>` section of the `App.vue` file in `src` directory.

#### ~/SRC/APP.VUE

```
<template>
<div id="app">
<span class="e-avatar">GR</span>
</div>
</template>
```

Here is the summarized code for the above steps in the `src/App.vue` file:

#### ~/SRC/APP.VUE

```
<template>
  <div id='element'>
    <span class="e-avatar e-avatar-xlarge"></span>
    <span class="e-avatar e-avatar-large"></span>
    <span class="e-avatar"></span>
    <span class="e-avatar e-avatar-small"></span>
    <span class="e-avatar e-avatar-xsmall"></span>
  </div>
</template>
<style>
  @import "../node_modules/@syncfusion/ej2-base/styles/material.css";
  @import "../node_modules/@syncfusion/ej2-layouts/styles/material.css";
  #element {
    display: block;
    width: 300px;
    margin: 130px auto;
    border-radius: 3px;
    justify-content: center;
  }
  .e-avatar {
    background-image: url(../pic01.png);
    margin: 2px;
  }
</style>
```

### Run the project

To run the project, use the following command:

```
`bash
```

```
npm run serve
```

```
,
```

or

```
`bash
```

```
yarn run serve
```

```
,
```



```
{% previewsample "page.domainurl/code-snippet/avatar/getting-started-cs1" %}
```

See Also

[Types of Avatar](#)

## Getting Started with the Vue Avatar Component in Vue 3

This article provides a step-by-step guide for setting up a [Vite](#) project with a JavaScript environment and integrating the Syncfusion Vue Avatar component using the [Composition API](#) / [Options API](#).

The **Composition API** is a new feature introduced in Vue.js 3 that provides an alternative way to organize and reuse component logic. It allows developers to write components as functions that use smaller, reusable functions called composition functions to manage their properties and behavior.

The **Options API** is the traditional way of writing Vue.js components, where the component logic is organized into a series of options that define the component's properties and behavior. These options include data, methods, computed properties, watchers, lifecycle hooks, and more.

Prerequisites

[System requirements for Syncfusion Vue UI components](#)

### Set up the Vite project

A recommended approach for beginning with Vue is to scaffold a project using [Vite](#). To create a new Vite project, use one of the commands that are specific to either NPM or Yarn.

```
`bash
```

```
npm create vite@latest
```

```
,
```

or

```
`bash
```

```
yarn create vite
```

```
,
```

Using one of the above commands will lead you to set up additional configurations for the project as below:

1. Define the project name: We can specify the name of the project directly. Let's specify the name of the project as **my-project** for this article.

```
`bash
```

```
? Project name: » my-project
```

```
,
```

2. Select **Vue** as the framework. It will create a Vue 3 project.

```
`bash
```

```
? Select a framework: » - Use arrow-keys. Return to submit.
```

Vanilla

Vue

React

Preact

Lit

Svelte

Others

,

3. Choose **JavaScript** as the framework variant to build this Vite project using JavaScript and Vue.

```
`bash
```

? Select a variant: » - Use arrow-keys. Return to submit.

JavaScript

TypeScript

Customize with create-vue ↗

Nuxt ↗

,

4. Upon completing the aforementioned steps to create the **my-project**, run the following command to install its dependencies:

```
`bash
```

```
cd my-project
```

```
npm install
```

,

or

```
`bash
```

```
cd my-project
```

```
yarn install
```

,

Now that **my-project** is ready to run with default settings, let's add Syncfusion components to the project.

#### Add Syncfusion Vue packages

Syncfusion Vue component packages are available at [npmjs.com](https://www.npmjs.com). To use Syncfusion Vue components in the project, install the corresponding npm package.

This article uses the [Vue Avatar component](#) as an example. To use the Vue Avatar component in the project, the **@syncfusion/ej2-vue-layouts** package needs to be installed using the following command:

```
`bash
```

```
npm install @syncfusion/ej2-vue-layouts --save
```

`

or

`bash

yarn add @syncfusion/ej2-vue-layouts

`

### Import Syncfusion CSS styles

You can import themes for the Syncfusion Vue component in various ways, such as using CSS or SASS styles from npm packages, CDN, [CRG](#) and [Theme Studio](#). Refer to [themes topic](#) to know more about built-in themes and different ways to refer to themes in a Vue project.

In this article, **Material** theme is applied using CSS styles, which are available in installed packages. The necessary **Material** CSS styles for the Avatar component and its dependents were imported into the `<style>` section of **src/App.vue** file.

#### ~/SRC/APP.VUE

```
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-layouts/styles/material.css";
</style>
```

### Add Syncfusion Vue component

Follow the below steps to add the Vue Avatar component using **Composition API** or **Options API**:

1.Add an HTML span element with **e-avatar** class into the section of the **App.vue** file in **src** directory.

#### ~/SRC/APP.VUE

```
<template>
<div id='element'>
<span class="e-avatar e-avatar-xlarge"></span>
<span class="e-avatar e-avatar-large"></span>
<span class="e-avatar"></span>
<span class="e-avatar e-avatar-small"></span>
<span class="e-avatar e-avatar-xsmall"></span>
</div>
</template>
```

Here is the summarized code for the above steps in the **src/App.vue** file:

#### ~/SRC/APP.VUE

```
<template>
<div id='element'>
<span class="e-avatar e-avatar-xlarge"></span>
<span class="e-avatar e-avatar-large"></span>
<span class="e-avatar"></span>
<span class="e-avatar e-avatar-small"></span>
<span class="e-avatar e-avatar-xsmall"></span>
</div>
</template>
<style>
```

```
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-layouts/styles/material.css";
#element {
  display: block;
  width: 300px;
  margin: 130px auto;
  border-radius: 3px;
  justify-content: center;
}
/* Add your custom avatar image. */
.e-avatar {
  background-image:
  url(https://ej2.syncfusion.com/vue/documentation/samples/avatar/media-formats-cs1/pic01.png);
  margin: 2px;
}
</style>
```

### Run the project

To run the project, use the following command:

```
`bash
```

```
npm run dev
```

```
`
```

or

```
`bash
```

```
yarn run dev
```

```
`
```

The output will appear as follows:



For migrating from Vue 2 to Vue 3, refer to the [migration](#) documentation.

See also

- [Getting Started with Vue UI Components using Composition API and TypeScript](#)

- [Getting Started with Vue UI Components using Options API and TypeScript](#)

## Types in Vue Avatar component

This section explains different types of avatar.

### Avatar size

The Essential JS 2 Avatar has the following predefined sizes that can be used with the `.e-avatar` class to change the appearance of the avatar.

Class Name	Description
<code>:-----:</code>	<code>:-----:</code>
<code>e-avatar-xlarge</code>	Displays xlarge size avatar.
<code>e-avatar-large</code>	Displays apply large size avatar.
<code>e-avatar</code>	Displays apply default size avatar.
<code>e-avatar-small</code>	Displays apply small size avatar.
<code>e-avatar-xsmall</code>	Displays apply xsmall size avatar.

### APP.VUE

```
<template>
  <div id='element'>
    <span class="e-avatar e-avatar-xlarge"></span>
    <span class="e-avatar e-avatar-large"></span>
    <span class="e-avatar"></span>
    <span class="e-avatar e-avatar-small"></span>
    <span class="e-avatar e-avatar-xsmall"></span>
  </div>
</template>
<script>
import Vue from "vue";
export default {
  data() {
    return {};
  }
}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-layouts/styles/material.css";
#element {
  display: block;
  width: 300px;
  margin: 100px auto;
  border-radius: 3px;
}
.e-avatar {
  background-image: url(./pic01.png);
}
</style>
```

{% previewsample "page.domainurl/code-snippet/avatar/size-cs1" %}

## Avatar types

The types of Essential JS 2 avatar are:

- Default
- Circle

### Default

The default style of the avatar is rectangular shape with rounded corners, which can be applied from adding the modifier class `.e-avatar` to the target element.

#### APP.VUE

```
<template>
  <div id='element'>
    <span class="e-avatar e-avatar-xlarge">RT</span>
    <span class="e-avatar e-avatar-large">RT</span>
    <span class="e-avatar">RT</span>
    <span class="e-avatar e-avatar-small">RT</span>
    <span class="e-avatar e-avatar-xsmall">RT</span>
  </div>
</template>
<script>
import Vue from "vue";
export default {
  data() {
    return {};
  }
}
</script>
<style>
  @import "../node_modules/@syncfusion/ej2-base/styles/material.css";
  @import "../node_modules/@syncfusion/ej2-layouts/styles/material.css";
  #element {
    display: block;
    width: 260px;
    margin: 100px auto;
    border-radius: 3px;
  }
</style>
```

{% previewsample "page.domainurl/code-snippet/avatar/default-cs1" %}

### Circle

The circle avatar style can be applied by adding the modifier class `.e-avatar-circle` to the target element.

#### APP.VUE

```
<template>
  <div id='element'>
    <span class="e-avatar e-avatar-xlarge e-avatar-circle">SJ</span>
    <span class="e-avatar e-avatar-large e-avatar-circle">SJ</span>
    <span class="e-avatar e-avatar-circle">SJ</span>
    <span class="e-avatar e-avatar-small e-avatar-circle">SJ</span>
    <span class="e-avatar e-avatar-xsmall e-avatar-circle">SJ</span>
  </div>
</template>
```

```

    </div>
</template>
<script>
import Vue from "vue";
export default {
  data() {
    return {};
  }
}
</script>
<style>
  @import "../node_modules/@syncfusion/ej2-base/styles/material.css";
  @import "../node_modules/@syncfusion/ej2-layouts/styles/material.css";
  #element {
    display: block;
    width: 300px;
    margin: 100px auto;
    border-radius: 3px;
  }
</style>

```

{% previewsample "page.domainurl/code-snippet/avatar/circle-cs1" %}

## How To

### Avatar customization in Vue Avatar component

#### Color customization

The avatar comes with default background color (grey). This can be easily customized to desired color by adding custom class or directly selecting the avatar class from the CSS.

#### APP.VUE

```

<template>
  <div id='element'>
    <span class="e-avatar e-avatar-xlarge e-avatar-circle
green">AJ</span>
    <span class="e-avatar e-avatar-xlarge e-avatar-circle
violet">JK</span>
    <span class="e-avatar e-avatar-xlarge e-avatar-circle
rose">EL</span>
    <span class="e-avatar e-avatar-xlarge e-avatar-circle
blue">SR</span>
    <span class="e-avatar e-avatar-xlarge e-avatar-circle red">PD</span>
  </div>
</template>
<script>
import Vue from "vue";
export default {
  data() {
    return {};
  }
}
</script>
<style>
  @import "../node_modules/@syncfusion/ej2-base/styles/material.css";
  @import "../node_modules/@syncfusion/ej2-layouts/styles/material.css";

```

```

#element {
  display: flex;
  width: 400px;
  margin: 100px auto;
  border-radius: 3px;
  justify-content: center;
}
.e-avatar {
  margin: 2px;
}
.e-avatar.green {
  background-color: #87eb87;
}
.e-avatar.violet {
  background-color: #ee82ee;
}
.e-avatar.blue {
  background-color: #7171e4;
}
.e-avatar.red {
  background-color: #d86e6e;
}
.e-avatar.rose {
  background-color: #bc8f8f;
}
}
</style>

```

{% previewsample "page.domainurl/code-snippet/avatar/color-cs1" %}

### *Customize avatar sizes*

Even though the avatar comes with five predefined sizes, sometimes it's not enough. So, the avatar is designed in such a way that the width and height will be relative to font-size. By changing the font-size of the avatar element, you can change the width and height automatically.

### **APP.VUE**

```

<template>
  <div id='element'>
    <span class="e-avatar">26px</span>
    <span class="e-avatar">24px</span>
    <span class="e-avatar">22px</span>
    <span class="e-avatar">20px</span>
    <span class="e-avatar">18px</span>
  </div>
</template>
<script>
import Vue from "vue";
export default {
  data() {
    return {};
  }
}
</script>
<style>
  @import "../node_modules/@syncfusion/ej2-base/styles/material.css";
  @import "../node_modules/@syncfusion/ej2-layouts/styles/material.css";

```



```

#element {
  display: block;
  width: 400px;
  margin: 100px auto;
  border-radius: 3px;
}
#element :nth-child(5) {
  font-size: 18px;
}
#element :nth-child(4) {
  font-size: 20px;
}
#element :nth-child(3) {
  font-size: 22px;
}
#element :nth-child(2) {
  font-size: 24px;
}
#element :nth-child(1) {
  font-size: 26px;
}
</style>

```

{% previewsample "page.domainurl/code-snippet/avatar/custom-size-cs1" %}

#### Use various media in avatar

Avatars can be used with a wide variety of media formats like SVG, font-icons, images, letters, words, etc. Some of them are given below.

#### APP.VUE

```

<template>
  <div id='element'>
    <div class="sample_container avatar-types">
      <div class="avatar-block">
        <!-- Card Component -->
        <div class="e-card e-avatar-showcase">
          <div class="e-card-content">
            <!-- XLarge Circle Avatar Component -->
            <div class="e-avatar e-avatar-xlarge e-avatar-circle">
              
            </div>
          </div>
          <div class="e-card-content">
            <div>Image</div>
          </div>
        </div>
      </div>
      <div class="avatar-block">
        <!-- Card Component -->
        <div class="e-card e-avatar-showcase">
          <div class="e-card-content">
            <!-- XLarge Circle Avatar Component -->

```

```

        <div class="e-avatar e-avatar-xlarge e-avatar-
circle">
            <div class="svg_icons chrome"></div>
        </div>
    </div>
    <div class="e-card-content">
        <div>SVG</div>
    </div>
</div>
<div class="avatar-block">
    <!-- Card Component -->
    <div class="e-card e-avatar-showcase">
        <div class="e-card-content">
            <!-- XLarge Circle Avatar Component -->
            <div class="e-avatar e-avatar-xlarge e-avatar-
circle">GR</div>
        </div>
        <div class="e-card-content">
            <div>Initial</div>
        </div>
    </div>
</div>
<div class="avatar-block">
    <!-- Card Component -->
    <div class="e-card e-avatar-showcase">
        <div class="e-card-content">
            <!-- XLarge Circle Avatar Component -->
            <div class="e-avatar e-avatar-xlarge e-avatar-
circle">
                <div class="e-people icons"></div>
            </div>
        </div>
        <div class="e-card-content">
            <div>Font Icon</div>
        </div>
    </div>
</div>
<div class="avatar-block">
    <!-- Card Component -->
    <div class="e-card e-avatar-showcase">
        <div class="e-card-content">
            <!-- XLarge Circle Avatar Component -->
            <div class="e-avatar e-avatar-xlarge e-avatar-
circle">User</div>
        </div>
        <div class="e-card-content">
            <div>Word</div>
        </div>
    </div>
</div>
<div class="avatar-block">
    <!-- Card Component -->
    <div class="e-card e-avatar-showcase">
        <div class="e-card-content">
            <!-- XLarge Circle Avatar Component -->

```

```

        <div class="e-avatar e-avatar-xlarge e-avatar-circle
custom">
            <div class="e-people icons"></div>
        </div>
    </div>
    <div class="e-card-content">
        <div>Custom</div>
    </div>
</div>
</div>
</template>
<script>
import Vue from "vue";
export default {
    data() {
        return {};
    }
}
</script>
<style>
    @import "../node_modules/@syncfusion/ej2-base/styles/material.css";
    @import "../node_modules/@syncfusion/ej2-layouts/styles/material.css";
    #element {
        width: auto;
        margin: auto;
        border-radius: 3px;
        justify-content: center;
    }
    /* Media Queries for various devices */
    @media only screen and (max-width: 965px) {
        .sample_container.avatar-types {
            max-width: 265px;
            margin: auto;
            margin-top: 0;
        }
        .e-avatar-showcase.e-card {
            width: 120px;
        }
    }
    @media only screen and (min-width: 965px) {
        .sample_container.avatar-types {
            max-width: 488px;
            margin: auto;
            margin-top: 35px;
        }
        .e-avatar-showcase.e-card {
            width: 150px;
        }
    }
    .sample_container.avatar-types .avatar-block {
        display: inline-block;
        vertical-align: top;
    }
    /* Avatar image source in 'background-image' property */
    .e-avatar img.image {

```

```

        background-repeat: no-repeat;
        background-size: cover;
        background-position: center;
    }
    /* SVG Icons */
    .chrome {
        background: transparent url("data:image/svg+xml,%3Csvg
xmlns='http://www.w3.org/2000/svg' viewBox='0 0 32 32'%3E%3Cpath
fill='%23ffffff' d='M16.033 11.049a5.155 5.155 0 1 1 0 10.312 5.156 5.156 0
0 1 0-10.312zM16.124 0c1.281-.003 9.659.318 14.268 9.043h-
.016l.01.018c.33.578 3.745 6.94-.485 14.969 0 0-4.215 8.107-14.565 7.968l-
.452-.012-.004.007-.004.007.02-.037c.564-.98 5.112-8.884 6.357-11.067l.016-
.028.007-.008.04-.069.11-.127a7.085 7.085 0 0 0 1.457-2.967l.01-.046.035-
.151c.088-.424.148-.944.128-1.549l-.006-.117v-.004l-.007-.143-.006-.07-.005-
.079-.012-.116v-.011-.001-.008-.016-.158a7.2 7.2 0 0 0-.096-.572l-.018-.081-
.013-.064a9.801 9.801 0 0 0-.692-2.016c-.165-.243-.332-.489-.512-.733l-.142-
.187 8.728-2.554s-10.538-.01-13.018-.001l.021.005H16.642l-.14-.013a7.034
7.034 0 0 0-1.132-.003l-.167.016h-.047l-.034-.001c-.193.002-1.213.045-
2.492.764l-.005.003-.033.016a7.158 7.158 0 0 0-3.25 3.533l-.059.148-6.485-
6.404s4.74 8.311 6.165 10.779l.065.113.023.088a7.14 7.14 0 0 0 7.777
5.118l.144-.02L14.854 32h-.027c-.667-.005-7.894-.234-12.744-7.906 0 0-4.925-
7.698.37-16.573l.252-.411.001-.002C2.822 6.904 6.58.374 15.958.003c0 0 .057-
.003.166-.003z'/%3E%3C/svg%3E") no-repeat 100% 100%;
    }
    .svg_icons {
        width: 32px;
        height: 32px;
        display: inline-block;
    }
    /* Card Customization */
    .e-avatar-showcase.e-card {
        height: 113px;
        padding: 5px;
        margin: 5px;
        box-shadow: 0 1px 3px rgba(0, 0, 0, 0.12), 0 1px 2px rgba(0, 0, 0,
0.24);
        border-radius: 8px;
    }
    .e-avatar-showcase.e-card .e-card-header .e-card-header-title {
        align-self: center;
        font-size: 18px;
        letter-spacing: 1px;
        text-shadow: #eaeaea 1px 1px 2px;
    }
    .e-avatar-showcase.e-card .e-card-header .e-card-sub-title {
        color: rgba(0, 0, 0, 0.75);
        white-space: pre-line;
        font-size: 14px;
        text-shadow: #eaeaea 1px 1px 2px;
    }
    .e-avatar-showcase.e-card .e-card-header .e-card-sub-title p {
        margin: 0;
    }
    .e-avatar-showcase.e-card .e-card-content {
        align-self: center;
        padding: 10px 0 10px 0;
        overflow: visible;
    }

```

```

.e-avatar-showcase.e-card .e-card-content .e-avatar {
    font-size: 18px;
}
/* Font Icons */
@font-face {
    font-family: 'Contacts';
    src: url(data:application/x-font-ttf;charset=utf-
8;base64,AAEAAAAKAIAAAwAgTlMvMj0gSRgAAAEoAAAAVmNtYXN0eOdaAAABjAAAADhnbHlmiAn
WagAAAcwAAADwaGVhZBD04psAADQAAAAANmhoZWEHiwNuAAAArAAAACRobXR4C9AAAAAAAYAAAAA
MbG9jYQAwAHgAAAEHAAAAACGlheHABDwA1AAABCAAAACBuYW1lby+ImAAAArWAAAIxcG9zdGUbI4A
AAATwAAAAOwABAAADUv9qAFoEAAAAAAD3QABAAAAAAAAAAAAAAAAAAAAAwABAAAAQAQAAW9ja18
PPPUACwPoAAAAANb8zuYAAAAA1vzO5gAAAAAD3QPqAAAAACAACAAAAAAAAAAAAEAAAADACkAAgAAAA
AAgAAAAAoACgAAAP8AAAAAAAAAAAPwAZAABQAAAAAnoCvAAAAIwCegK8AAAB4AAxAQIAAAIABQMAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAUGZFZABA5wDnAQNS/2oAWgPqAJYAAAABAAAAAAAAABAAAAAPoAAA
D6AAAAAAAAAgAAAAAMAAAAUAAMAAQAAABQABAakAAAAABAAEAAEAAOcb//8AA0cA//8AAAAABAAQAAAA
BAAIAAAAAAADAAeAACAAAAAAAO+A+oADQAZAAA3FBYXIT4BNS4BJyEOARMeARc+ATcuAScOAS4YFwM
zFxdGd4H+zYGr4QOCY2KCAwOCYmGCnCtOISFOK3qiAwOiAeligwICg2JjggICggAAAAACAAAAAAP
dA+oAFAAoAAABDgEHIicjDgEHLgEnLgEnPgE3HgEFFBYfARYfATcXFhc2JDcmJCcGBAOkBfK3Kio
XEFcpBBEMRUSBBfK3tvL8cVRLDggBBsQLDDPARMFb7tz87+7QI+ndEEBwI/Izh2DS+RVZ3RBAT
RnWCmN3BIETecAgcBBPK1tFIEBPIAAAAABIA3gABAAAAAAAAAAAAEAAAABAAAAAABAAgAAQABAAA
AAAAACAACACQABAAAAAADAAgAEABAAAAAAAAAAEAAgAGAABAAAAAAAAFAAAsAIAABAAAAAAGAAGAKwA
BAAAAAAAKACwAMwABAAAAAALABIAxwADAAEEECQAAAAIacQADAAEEECQABABAAcwADAAEEECQACAA4
AgwADAAEEECQADABAAkQADAAEEECQAEABAAoQADAAEEECQAFABYAsQADAAEEECQAGABAAxwADAAEEECQA
KfGAlwADAAEEECQALACQBLyBDb250YWN0c1JlZ3VsYXJDb250YWN0c0Nvb3R5Y3RzVmVyc2lvd3d
xLjBDb250YWN0c0ZvbnQgZ2VzZXJhdGVkIHVzaW5nIFN5bmNmdXNpb24gTWV0cm8gU3RlZGlvd3d
3LnN5bmNmdXNpb24uY29tACAAQwBvAG4AdABhAGMAdABzAFIAZQBnAHUAbABhAHIAQwBvAG4AdAB
hAGMAdABzAEMAbwBuAHQAYQBjAHQAcwBwAGUAcgBzAGkAbwBuACAAMQAUADAAQwBvAG4AdABhAGM
AdABzAEYAbwBuAHQAIAbnAGUAbgBlAHIAAYQB0AGUAZAAGAHUAcwBpAG4AZwAgAFMAeQBuAGMAZgB
1AHMAaQBvAG4AIAbnAGUAdABYAG8AIAbTAHQAdQBkAGkAbwB3AHcAdwAuAHMAeQBuAGMAZgB1AHM
AaQBvAG4ALgBjAG8AbQAAAAACAAAAAaAAAAAaAAAAAaAAAAAaAAAAAaAAAAAaAAAAAaAAAAAaAAAA
ABHVzZXIKY2hhdC0wMS13ZgAAAA==) format('truetype');
    font-weight: normal;
    font-style: normal;
}
.people,
.e-people {
    font-family: 'Contacts';
}
.e-people:before {
    content: '\e700';
}
.e-avatar .e-people.icons {
    font-size: 24px;
}
/* Custom Avatar Background Color */
.e-avatar.custom {
    background-color: blueviolet;
}

```

```
{% previewsample "page.domainurl/code-snippet/avatar/media-formats-cs1" %}
```

## Integrate avatar into listview in Vue Avatar component

Avatar can be integrated into various components to make a wide variety of applications. Some of the integrations are shown in the following section.

*ListView*

Avatar is integrated into the listview to create contacts applications. The `xsmall` size avatar is used to match the size of the list item. Letters and images are also used as avatar content.

**APP.VUE**

```
<template>
  <div id='element'>
    <!-- ListView element -->
    <ejs-listview id='letterAvatarList' :dataSource='dataSource'
:sortOrder='sortOrder' showHeader='true'
    headerTitle='Contacts' :template='listTemplate' >
    </ejs-listview>
  </div>
</template>
<script>
import Vue from "vue";
import { ListViewPlugin } from "@syncfusion/ej2-vue-lists";
Vue.use(ListViewPlugin);
var listtemplateVue = Vue.component("demo", {
  template: ` <div class="listWrapper">
    <span :class="['e-avatar e-avatar-small e-avatar-circle']" v-
if="data.avatar !== ''">{{data.avatar}}</span>
    <span :class="[data.pic + ' e-avatar e-avatar-small e-avatar-
circle']" v-if="data.pic !== ' ' "> </span>
    <span class="list-text">{{data.text}}</span>
  </div>`,
  data() {
    return {
      data: {}
    };
  }
});
export default {
  data() {
    return {
      dataSource: [
        { id: 's_01', text: 'Robert', avatar: '', pic: 'pic04' },
        { id: 's_02', text: 'Nancy', avatar: 'N', pic: '' },
        { id: 's_05', text: 'John', pic: 'pic01', avatar: '' },
        { id: 's_03', text: 'Andrew', avatar: 'A', pic: '' },
        { id: 's_06', text: 'Michael', pic: 'pic02', avatar: '' },
        { id: 's_07', text: 'Steven', pic: 'pic03', avatar: '' },
        { id: 's_08', text: 'Margaret', avatar: 'M', pic: '' }
      ],
      listTemplate: function () {
        return { template : listtemplateVue}
      },
      sortOrder: 'Ascending'
    };
  }
}
</script>
<style>
  @import "../node_modules/@syncfusion/ej2-base/styles/material.css";
  @import "../node_modules/@syncfusion/ej2-layouts/styles/material.css";
  #element {
```

```

        width: 400px;
        margin: auto;
        border-radius: 3px;
        justify-content: center;
    }
    /* ListView Customization */
    #letterAvatarList {
        max-width: 350px;
        margin: auto;
        border: 1px solid #dddddd;
        border-radius: 3px;
    }
    #letterAvatarList .listWrapper {
        width: inherit;
        height: inherit;
        position: relative;
    }
    #letterAvatarList .e-list-header {
        height: 54px;
    }
    .material #letterAvatarList .e-list-header .e-text {
        line-height: 22px;
    }
    .fabric #letterAvatarList .e-list-header .e-text {
        line-height: 22px;
    }
    .bootstrap #letterAvatarList .e-list-header .e-text {
        line-height: 13px;
    }
    .highcontrast #letterAvatarList .e-list-header .e-text {
        line-height: 20px;
    }
    #letterAvatarList .e-list-item {
        cursor: pointer;
        height: 50px;
        line-height: 44px;
        border: 0;
    }
    /* Badge Positioning */
    #letterAvatarList .e-badge {
        margin-top: 12px;
    }
    #letterAvatarList .listWrapper .list-text {
        width: 60%;
        display: inline-block;
        vertical-align: middle;
        margin: 0 50px;
    }
    /* Avatar Positioning */
    #letterAvatarList .listWrapper .e-avatar {
        position: absolute;
        top: calc(100% - 40px);
        font-size: 10px;
        left: 5px;
    }
    /* Avatar Background Customization */
    #letterAvatarList .e-list-item:nth-child(1) .e-avatar {

```

```

        background-color: #039BE5;
    }
    #letterAvatarList .e-list-item:nth-child(3) .e-avatar {
        background-color: #E91E63;
    }
    #letterAvatarList .e-list-item:nth-child(5) .e-avatar {
        background-color: #009688;
    }
    /* Avatar images using 'background-image' property */
    .pic01 {
        background-image:
url('https://ej2.syncfusion.com/demos/src/grid/images/2.png');
    }
    .pic02 {
        background-image:
url('https://ej2.syncfusion.com/demos/src/grid/images/5.png');
    }
    .pic03 {
        background-image:
url('https://ej2.syncfusion.com/demos/src/grid/images/6.png');
    }
    .pic04 {
        background-image:
url('https://ej2.syncfusion.com/demos/src/grid/images/7.png');
    }
</style>

```

{% previewsample "page.domainurl/code-snippet/avatar/listview-cs1" %}

### Integrate avatar into badge in Vue Avatar component

The badge is dependent and supportive component, and it can be used with avatar to create a notification avatar.

The default avatar (.e-avatar) and circle avatar (.e-avatar-circle) have been used with notification badges (.e-badge-notification) in the following sample.

### APP.VUE

```

<template>
  <div id='element'>
    <div class="sample_container avatar-badge">
      <div class="avatar-block">
        <!-- Card Component -->
        <div class="e-card e-avatar-showcase">
          <div class="e-card-content">
            <div class="avatar-sub-block">
              <!-- xSmall Avatar-->
              <div class="e-avatar e-avatar-xsmall">
                
              </div>
              <!-- Notification Badge -->
              <span class="e-badge e-badge-primary e-badge-
overlap e-badge-notification e-badge-circle">6</span>
            </div>
          </div>
        </div>
      </div>
    </div>
  </div>

```



```


<!-- Small Avatar-->
  <div class="e-avatar e-avatar-small">
    
  </div>
  <!-- Notification Badge -->
  <span class="e-badge e-badge-primary e-badge-
overlap e-badge-notification e-badge-circle">12</span>
</div>
<div class="avatar-sub-block">
  <!-- Avatar-->
  <div class="e-avatar">
    
  </div>
  <!-- Notification Badge -->
  <span class="e-badge e-badge-primary e-badge-
overlap e-badge-notification">46</span>
</div>
<div class="avatar-sub-block">
  <!-- Large Avatar-->
  <div class="e-avatar e-avatar-large">
    
  </div>
  <!-- Notification Badge -->
  <span class="e-badge e-badge-primary e-badge-
overlap e-badge-notification">82</span>
</div>
<div class="avatar-sub-block">
  <!-- xLarge Avatar-->
  <div class="e-avatar e-avatar-xlarge">
    
  </div>
  <!-- Notification Badge -->
  <span class="e-badge e-badge-primary e-badge-
overlap e-badge-notification">99+</span>
</div>
</div>
</div>
<div class="circleAvatar avatar-block">
  <!-- Card Component -->
  <div class="e-card e-avatar-showcase">
    <div class="e-card-content">
      <div class="avatar-sub-block">
        <!-- xSmall Circle Avatar-->
        <div class="e-avatar e-avatar-circle e-avatar-
xsmall">


```

```


    </div>
    <!-- Notification Badge -->
    <span class="e-badge e-badge-primary e-badge-
overlap e-badge-notification e-badge-circle">6</span>
    </div>
    <div class="avatar-sub-block">
    <!-- Small Circle Avatar-->
    <div class="e-avatar e-avatar-circle e-avatar-
small">
        
        </div>
        <!-- Notification Badge -->
        <span class="e-badge e-badge-primary e-badge-
overlap e-badge-notification e-badge-circle">12</span>
        </div>
        <div class="avatar-sub-block">
        <!-- Circle Avatar-->
        <div class="e-avatar e-avatar-circle">
            
            </div>
            <!-- Notification Badge -->
            <span class="e-badge e-badge-primary e-badge-
overlap e-badge-notification">46</span>
            </div>
            <div class="avatar-sub-block">
            <!-- Large Circle Avatar-->
            <div class="e-avatar e-avatar-circle e-avatar-
large">
                
                </div>
                <!-- Notification Badge -->
                <span class="e-badge e-badge-primary e-badge-
overlap e-badge-notification">82</span>
                </div>
                <div class="avatar-sub-block">
                <!-- xLarge Circle Avatar-->
                <div class="e-avatar e-avatar-circle e-avatar-
xlarge">
                    
                    </div>
                    <!-- Notification Badge -->
                    <span class="e-badge e-badge-primary e-badge-
overlap e-badge-notification">99+</span>
                    </div>
                </div>
            </div>
        </div>
    </div>

```

```

        </div>
    </div>
</div>
</template>
<script>
import Vue from "vue";
export default {
    data() {
        return {};
    }
}
</script>
<style>
    @import "../node_modules/@syncfusion/ej2-base/styles/material.css";
    @import "../node_modules/@syncfusion/ej2-layouts/styles/material.css";
    #element {
        display: block;
        margin: auto;
        border-radius: 3px;
        justify-content: center;
    }
    .sample_container.avatar-badge .avatar-sub-block {
        display: inline-block;
        position: relative;
        margin: 7px
    }
    .sample_container.avatar-badge .avatar-block {
        display: inline-block;
        vertical-align: top;
    }
    /* Media Queries for various devices */
    @media only screen and (max-width: 965px) {
        .sample_container.avatar-badge {
            max-width: 332px;
            margin: auto;
            margin-top: 0;
        }
        .circleAvatar {
            margin-top: 15px;
        }
        .e-avatar-showcase.e-card {
            width: 320px;
        }
    }
    @media only screen and (min-width: 965px) {
        .sample_container.avatar-badge {
            max-width: 825px;
            margin: auto;
            margin-top: 70px;
        }
        .e-avatar-showcase.e-card {
            width: 400px;
        }
    }
    .e-avatar.image {
        background-repeat: no-repeat;
        background-size: cover;
    }

```

```

        background-position: center;
    }
    /* Card Customization */
    .e-avatar-showcase.e-card {
        height: 140px;
        padding: 4px;
        margin: 5px;
        box-shadow: 0 1px 3px rgba(0, 0, 0, 0.12), 0 1px 2px rgba(0, 0, 0,
0.24);
        border-radius: 8px;
    }
    .e-avatar-showcase.e-card .e-card-header .e-card-header-title {
        align-self: center;
        font-size: 18px;
        letter-spacing: 1px;
        text-shadow: #eaeaea 1px 1px 2px;
    }
    .e-avatar-showcase.e-card .e-card-header .e-card-sub-title {
        color: rgba(0, 0, 0, 0.75);
        white-space: pre-line;
        font-size: 14px;
        text-shadow: #eaeaea 1px 1px 2px;
    }
    .e-avatar-showcase.e-card .e-card-header .e-card-sub-title p {
        margin: 0;
    }
    .e-avatar-showcase.e-card .e-card-content {
        align-self: center;
        padding: 10px 11px 10px 0px;
        color: rgba(0, 0, 0, 0.75);
        overflow: visible;
    }
}
</style>

```

{% previewsample "page.domainurl/code-snippet/avatar/badge-cs1" %}

## Badge

### Getting Started with the Vue Badge Component in Vue 2

This article provides a step-by-step guide for setting up a Vue 2 project using [Vue-CLI](#) and integrating the Syncfusion Vue Badge component using the [Composition API](#) / [Options API](#).

#### Prerequisites

##### [System requirements for Syncfusion Vue UI components](#)

#### Setting up the Vue 2 project

To generate a Vue 2 project using Vue-CLI, use the [vue create](#) command. Follow these steps to install Vue CLI and create a new project:

```
`bash
```

```
npm install -g @vue/cli
```

```
vue create quickstart
```

```
cd quickstart
```

```
npm run serve
```

```
,
```

```
or
```

```
`bash
```

```
yarn global add @vue/cli
```

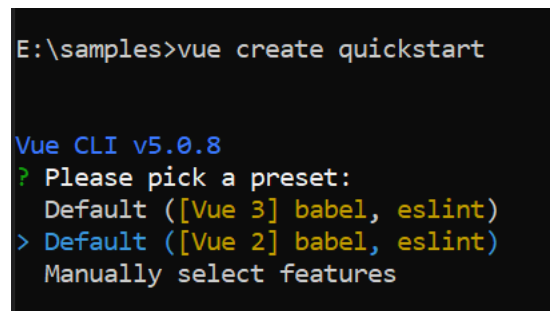
```
vue create quickstart
```

```
cd quickstart
```

```
yarn run serve
```

```
,
```

When creating a new project, choose the option `Default ([Vue 2] babel, eslint)` from the menu.



```
E:\samples>vue create quickstart

Vue CLI v5.0.8
? Please pick a preset:
  Default ([Vue 3] babel, eslint)
> Default ([Vue 2] babel, eslint)
  Manually select features
```

Once the `quickstart` project is set up with default settings, proceed to add Syncfusion components to the project.

#### Add Syncfusion Vue packages

Syncfusion packages are available at [npmjs.com](https://www.npmjs.com). To use Vue components, install the required npm package.

This article uses the [Vue Badge component](#) as an example. Install the `@syncfusion/ej2-vue-notifications` package by running the following command:

```
`bash
```

```
npm install @syncfusion/ej2-vue-notifications --save
```

```
,
```

```
or
```

```
`bash
```

```
yarn add @syncfusion/ej2-vue-notifications
```

```
,
```

#### Import Syncfusion CSS styles

You can import themes for the Syncfusion Vue component in various ways, such as using CSS or SASS styles from npm packages, CDN, [CRG](#) and [Theme Studio](#). Refer to [themes topic](#) to know more about built-in themes and different ways to refer to themes in a Vue project.

In this article, the **Material** theme is applied using CSS styles, which are available in installed packages. The necessary **Material** CSS styles for the Badge component and its dependents were imported into the `<style>` section of **src/App.vue** file.

#### ~/SRC/APP.VUE

```
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-notifications/styles/material.css";
</style>
```

Add Syncfusion Vue component

Add an HTML span element with **e-badge** class inside any wrapper element (**h1**) into the `<template>` section of the **App.vue** file in **src** directory.

#### ~/SRC/APP.VUE

```
<template>
<div id="app">
<h1>Badge Component <span class="e-badge">New</span></h1>
</div>
</template>
```

We can also use [CRG](#) to generate combined component styles.

Here is the summarized code for the above steps in the **src/App.vue** file:

#### APP.VUE

```
<template>
  <div id='element'>
    <h1>Badge Component <span class="e-badge e-badge-
primary">New</span></h1>
  </div>
</template>
<script>
export default {
  data() {
    return {};
  }
}
</script>
<style>
  @import "../node_modules/@syncfusion/ej2-vue-
notifications/styles/material.css";
  #element {
    display: flex;
    width: 400px;
    margin: auto;
    border: 1px solid #dddddd;
    border-radius: 3px;
    justify-content: center;
  }
</style>
```

### Run the project

To run the project, use the following command:

```
`bash
```

```
npm run serve
```

```
,
```

or

```
`bash
```

```
yarn run serve
```

```
,
```

```
{% previewsample "page.domainurl/code-snippet/badge/getting-started-cs1" %}
```

See Also

[Types of Badge](#)

### Getting Started with the Vue Badge Component in Vue 3

This article provides a step-by-step guide for setting up a [Vite](#) project with a JavaScript environment and integrating the Syncfusion Vue Badge component using the [Composition API](#) / [Options API](#).

The **Composition API** is a new feature introduced in Vue.js 3 that provides an alternative way to organize and reuse component logic. It allows developers to write components as functions that use smaller, reusable functions called composition functions to manage their properties and behavior.

The **Options API** is the traditional way of writing Vue.js components, where the component logic is organized into a series of options that define the component's properties and behavior. These options include data, methods, computed properties, watchers, lifecycle hooks, and more.

### Prerequisites

[System requirements for Syncfusion Vue UI components](#)

### Set up the Vite project

A recommended approach for beginning with Vue is to scaffold a project using [Vite](#). To create a new Vite project, use one of the commands that are specific to either NPM or Yarn.

```
`bash
```

```
npm create vite@latest
```

```
,
```

or

```
`bash
```

```
yarn create vite
```

```
,
```

Using one of the above commands will lead you to set up additional configurations for the project as below:

1. Define the project name: We can specify the name of the project directly. Let's specify the name of the project as **my-project** for this article.

```
`bash
```

```
? Project name: » my-project
```

```
,
```

2. Select **Vue** as the framework. It will create a Vue 3 project.

```
`bash
```

```
? Select a framework: » - Use arrow-keys. Return to submit.
```

```
Vanilla
```

```
Vue
```

```
React
```

```
Preact
```

```
Lit
```

```
Svelte
```

```
Others
```

```
,
```

3. Choose **JavaScript** as the framework variant to build this Vite project using JavaScript and Vue.

```
`bash
```

```
? Select a variant: » - Use arrow-keys. Return to submit.
```

```
JavaScript
```

```
TypeScript
```

```
Customize with create-vue ↗
```

```
Nuxt ↗
```

```
,
```

4. Upon completing the aforementioned steps to create the **my-project**, run the following command to install its dependencies:

```
`bash
```

```
cd my-project
```

```
npm install
```

```
,
```

```
or
```

```
`bash
```

```
cd my-project
```

```
yarn install
```

```
,
```



Now that `my-project` is ready to run with default settings, let's add Syncfusion components to the project.

### Add Syncfusion Vue packages

Syncfusion Vue component packages are available at [npmjs.com](https://www.npmjs.com). To use Syncfusion Vue components in the project, install the corresponding npm package.

This article uses the [Vue Badge component](#) as an example. To use the Vue Badge component in the project, the `@syncfusion/ej2-vue-notifications` package needs to be installed using the following command:

```
`bash
npm install @syncfusion/ej2-vue-notifications --save
`
```

or

```
`bash
yarn add @syncfusion/ej2-vue-notifications
`
```

### Import Syncfusion CSS styles

You can import themes for the Syncfusion Vue component in various ways, such as using CSS or SASS styles from npm packages, CDN, [CRG](#) and [Theme Studio](#). Refer to [themes topic](#) to know more about built-in themes and different ways to refer to themes in a Vue project.

In this article, `Material` theme is applied using CSS styles, which are available in installed packages. The necessary `Material` CSS styles for the Badge component and its dependents were imported into the `<style>` section of `src/App.vue` file.

#### ~/SRC/APP.VUE

```
<style>
@import "../node_modules/@syncfusion/ej2-vue-
notifications/styles/material.css";
</style>
```

### Add Syncfusion Vue component

Follow the below steps to add the Vue Badge component using `Composition API` or `Options API`:

1. Add an HTML span element with `e-badge` class inside any wrapper element (`h1`) into the `section` of the `App.vue` file in `src` directory.

#### ~/SRC/APP.VUE

```
<template>
<div id='element'>
<h1>Badge Component <span class="e-badge e-badge-primary">New</span></h1>
</div>
</template>
```

Here is the summarized code for the above steps in the `src/App.vue` file:

~/SRC/APP.VUE

```
<template>
<div id='element'>
<h1>Badge Component <span class="e-badge e-badge-primary">New</span></h1>
</div>
</template>
<style>
@import "../node_modules/@syncfusion/ej2-vue-
notifications/styles/material.css";
#element {
display: flex;
width: 400px;
margin: auto;
border: 1px solid #dddddd;
border-radius: 3px;
justify-content: center;
}
</style>
```

## Run the project

To run the project, use the following command:

```
`bash
```

```
npm run dev
```

```
`
```

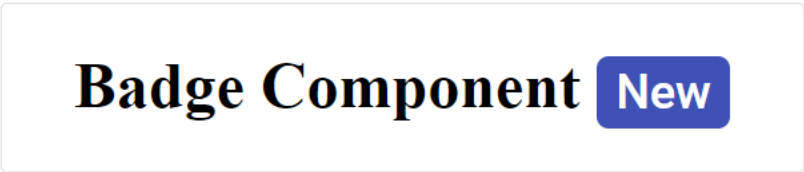
or

```
`bash
```

```
yarn run dev
```

```
`
```

The output will appear as follows:



For migrating from Vue 2 to Vue 3, refer to the [migration](#) documentation.

See also

- [Getting Started with Vue UI Components using Composition API and TypeScript](#)
- [Getting Started with Vue UI Components using Options API and TypeScript](#)

## Types in Vue Badge component

This section explains different styles and types of the badges.

### Badge styles

The Essential JS 2 Badge has the following predefined styles that can be used with `.e-badge` class to change the appearance of a badge.

Class Name	Description
:-----	:-----
e-badge-primary	Represents a primary notification.
e-badge-secondary	Represents a secondary notification.
e-badge-success	Represents a positive notification.
e-badge-danger	Represents a negative notification.
e-badge-warning	Represents notification with caution.
e-badge-info	Represents an informative notification.
e-badge-light	Represents notification in light variant.
e-badge-dark	Represents notification in dark variant.

### APP.VUE

```
<template>
  <div id='element'>
    <div class="sample_container">
      <div class="block" style="display:inline-block">
        <div class="e-card e-badge-showcase">
          <div class="e-card-content">
            <div>
              <span class="e-badge e-badge-
primary">Primary</span>
            </div>
          </div>
          <div class="e-card-content">
            <div>
              <code>.e-badge-primary</code>
            </div>
          </div>
        </div>
      </div>
      <div class="block" style="display:inline-block">
        <div class="e-card e-badge-showcase">
          <div class="e-card-content">
            <span class="e-badge e-badge-
secondary">Secondary</span>
          </div>
          <div class="e-card-content">
            <code>.e-badge-secondary</code>
          </div>
        </div>
      </div>
      <div class="block" style="display:inline-block">
        <div class="e-card e-badge-showcase">
          <div class="e-card-content">
            <span class="e-badge e-badge-success">Success</span>
          </div>
        </div>
      </div>
    </div>
  </div>
```

```

        </div>
        <div class="e-card-content">
            <code>.e-badge-success</code>
        </div>
    </div>
</div>
<div class="block" style="display:inline-block">
    <div class="e-card e-badge-showcase">
        <div class="e-card-content">
            <span class="e-badge e-badge-danger">Danger</span>
        </div>
        <div class="e-card-content">
            <code>.e-badge-danger</code>
        </div>
    </div>
</div>
<div class="block" style="display:inline-block">
    <div class="e-card e-badge-showcase">
        <div class="e-card-content">
            <span class="e-badge e-badge-warning">Warning</span>
        </div>
        <div class="e-card-content">
            <code>.e-badge-warning</code>
        </div>
    </div>
</div>
<div class="block" style="display:inline-block">
    <div class="e-card e-badge-showcase">
        <div class="e-card-content">
            <span class="e-badge e-badge-info">Info</span>
        </div>
        <div class="e-card-content">
            <code>.e-badge-info</code>
        </div>
    </div>
</div>
<div class="block" style="display:inline-block">
    <div class="e-card e-badge-showcase">
        <div class="e-card-content">
            <span class="e-badge e-badge-light">Light</span>
        </div>
        <div class="e-card-content">
            <code>.e-badge-light</code>
        </div>
    </div>
</div>
<div class="block" style="display:inline-block">
    <div class="e-card e-badge-showcase">
        <div class="e-card-content">
            <span class="e-badge e-badge-dark">Dark</span>
        </div>
        <div class="e-card-content">
            <code>.e-badge-dark</code>
        </div>
    </div>
</div>
</div>
</div>

```

```

    </div>
  </template>
  <script>
import Vue from "vue";
export default {
  data() {
    return {};
  }
}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-
notifications/styles/material.css";
#element {
  width: 100%;
  margin: auto;
}
.sample_container {
  width: 800px;
  margin: auto;
}
.block {
  margin: 10px 10px 10px 10px;
  vertical-align: top;
}
.e-badge-showcase.e-card {
  width: 150px;
  height: 90px;
  padding: 5px;
  margin: 5px;
  box-shadow: 0 1px 3px rgba(0, 0, 0, 0.12), 0 1px 2px rgba(0, 0, 0,
0.24);
  border-radius: 8px;
}
.e-badge-showcase.e-card .e-card-header .e-card-header-title {
  align-self: center;
  font-size: 18px;
  letter-spacing: 1px;
  text-shadow: #eaeaea 1px 1px 2px;
}
.e-badge-showcase.e-card .e-card-header .e-card-sub-title {
  color: rgba(0, 0, 0, 0.75);
  white-space: pre-line;
  font-size: 14px;
  text-shadow: #eaeaea 1px 1px 2px
}
.e-badge-showcase.e-card .e-card-header .e-card-sub-title p {
  margin: 0;
}
.e-badge-showcase.e-card .e-card-content {
  align-self: center;
  padding: 0 0 10px 0;
}
.e-badge-showcase.e-card .e-card-content .e-badge {
  font-size: 12px;
}

```

```
</style>
```

```
{% previewsample "page.domainurl/code-snippet/badge/types-cs1" %}
```

### Badge types

The types of Essential JS 2 badges are as follows:

- Circle
- Pill
- Link
- Notification
- Overlap
- Dot
- Position

### Circle

The circle badge style can be applied by adding the modifier class `e-badge-circle` to the target element.

### APP.VUE

```
<template>
  <div id='element'>
    <div style="position:relative;display:inline-block;margin:20px 20px
10px 20px;">
      <div class="skype svg_icons"></div>
      <span class="e-badge e-badge-success e-badge-overlap e-badge-
notification e-badge-circle">18</span>
    </div>
    <div style="position:relative;display:inline-block;margin:20px 20px
10px 20px;">
      <div class="twitter svg_icons"></div>
      <span class="e-badge e-badge-info e-badge-overlap e-badge-
notification e-badge-circle">9</span>
    </div>
    <div style="position:relative;display:inline-block;margin:20px 20px
10px 20px;">
      <div class="facebook svg_icons"></div>
      <span class="e-badge e-badge-info e-badge-overlap e-badge-
notification e-badge-circle">2</span>
    </div>
    <div style="position:relative;display:inline-block;margin:20px 20px
10px 20px;">
      <div class="firefox svg_icons"></div>
      <span class="e-badge e-badge-danger e-badge-overlap e-badge-
notification e-badge-circle">35</span>
    </div>
  </div>
</template>
<script>
import Vue from "vue";
export default {
  data() {
    return {};
  }
}
```

```

}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-vue-base/styles/material.css";
  @import "../node_modules/@syncfusion/ej2-vue-
notifications/styles/material.css";
  #element {
    display: flex;
    width: 400px;
    margin: auto;
    border: 1px solid #dddddd;
    border-radius: 3px;
    justify-content: center;
    position: relative;
    top: 130px;
  }
  .facebook {
    background-image: url("data:image/svg+xml;charset=UTF-8,%3csvg
id='Layer_1' data-name='Layer 1' xmlns='http://www.w3.org/2000/svg'
viewBox='0 0 32 32'%3e%3cdefs%3e%3cstyle%3e.cls-1%7bfill:%233c5a99;%7d.cls-
2%7bfill:%23fff;%7d%3c/style%3e%3c/defs%3e%3ctitle%3eIcon%3c/title%3e%3crect
class='cls-1' x='1' y='1' width='28' height='28' rx='5.75'
ry='5.75'/%3e%3cpath class='cls-2'
d='M22.53,8.7h2.58V4.64H21.24a5.25,5.25,0,0,0-4,2s-1.06,1-
1.08,3.92h0v3H12.36v4.31h3.82V29h4.41V17.86h3.81.53-4.31H20.59v-
3h0A1.78,1.78,0,0,1,22.53,8.7Z'/%3e%3c/svg%3e");
  }
  .skype {
    background-image: url("data:image/svg+xml;charset=UTF-8,%3csvg
id='Layer_1' data-name='Layer 1' xmlns='http://www.w3.org/2000/svg'
viewBox='0 0 32 32'%3e%3cdefs%3e%3cstyle%3e.cls-1%7bfill:%2331c4ed;%7d.cls-
2%7bfill:%23fff;fill-
rule:evenodd;%7d%3c/style%3e%3c/defs%3e%3ctitle%3eIcon%3c/title%3e%3crect
class='cls-1' x='1' y='1' width='28' height='28' rx='5.75'
ry='5.75'/%3e%3cpath class='cls-2' d='M19.5,19.14a4,4,0,0,1-
1.75,1.31,7.22,7.22,0,0,1-
2.68,4.6A6.6,6.6,0,0,1,12,20.27,4,4,0,0,1,10.58,19,2.71,2.71,0,0,1,10,17.51a1
.05,1.05,0,0,1,.36-.8,1.28,1.28,0,0,1,.9-
.33,1.14,1.14,0,0,1,.75,2.6,1.89,1.89,0,0,1,.51,7.3,4.54,4.54,0,0,0,.49,8.6,2,2
,0,0,0,.72,5.5A3.08,3.08,0,0,0,15,19a3,3,0,0,0,1.73-.45,1.23,1.23,0,0,0,.63-
1.06,1,1,0,0,0-.33-.8,2.3,2.3,0,0,0-.92-.49c-.39-.12-.92-.26-1.57-
.39a12.2,12.2,0,0,1-2.25-.67,3.64,3.64,0,0,1-1.48-1.06,2.47,2.47,0,0,1-.29-
.47,2.84,2.84,0,0,1-.26-1.22,2.71,2.71,0,0,1,.58-1.72,3.71,3.71,0,0,1,1.67-
1.14A7.37,7.37,0,0,1,15,9.14a7,7,0,0,1,2,.26,4.38,4.38,0,0,1,1.42,7,3.1,3.1,
0,0,1,.84,9.2,2.11,2.11,0,0,1,.26,1,1.13,1.13,0,0,1-.35,8.2,1.18,1.18,0,0,1-
.88,3.6,1.09,1.09,0,0,1-.74-.23,2.68,2.68,0,0,1-.51-.67,3,3,0,0,0-.77-
.94A2.42,2.42,0,0,0,14.87,11a2.76,2.76,0,0,0-1.49,3.6,1,1,0,0,0-
.53,8.1,7.8,7.8,0,0,0,.17,5,1.63,1.63,0,0,0,.52,3.8,4.48,4.48,0,0,0,.69,2.71,1.5,0
,1,.24c.69,1.5,1.33,3.2,1.89,4.9a6.29,6.29,0,0,1,1.47,6.7,2.81,2.81,0,0,1,1,1,3,
3,0,0,1,.35,1.49,3.15,3.15,0,0,1-.6,1.89Zm4-2.23a7.73,7.73,0,0,0,.2-1.81c0-
.22,0-.43,0-.64a8.53,8.53,0,0,0-8.56-7.83,8.72,8.72,0,0,0-
1.46,12A5.08,5.08,0,0,0,11,6a5,5,0,0,0-
5,4.91,4.83,4.83,0,0,0,.68,2.48,7.33,7.33,0,0,0-
.13,9.4,6.51,6.51,0,0,0,0,.77,8.52,8.52,0,0,0,8.58,8.46,8.9,8.9,0,0,0,1.57-
.14A5.09,5.09,0,0,0,19,24a4.94,4.94,0,0,0,5-4.91,4.86,4.86,0,0,0-.52-
2.18Z'/%3e%3c/svg%3e");
  }

```

```

        .twitter {
            background-image: url("data:image/svg+xml;charset=UTF-8,%3csvg
            id='Layer_1' data-name='Layer 1' xmlns='http://www.w3.org/2000/svg'
            viewBox='0 0 32 32'%3e%3cdefs%3e%3cstyle%3e.cls-1%7bfill:%231da1f2;%7d.cls-
            2%7bfill:%23fff;%7d%3c/style%3e%3c/defs%3e%3ctitle%3eIcon%3c/title%3e%3crect
            class='cls-1' x='1' y='1' width='28' height='28' rx='5.75'
            ry='5.75'/%3e%3cpath class='cls-2'
            d='M11.67,23.11A11.3,11.3,0,0,0,23.05,11.73c0-.17,0-.35,0-.52a8,8,0,0,0,2-
            2.07,8,8,0,0,1-2.29.63,4,4,0,0,0,1.75-2.21,8,8,0,0,1-2.54,1,4,4,0,0,0-
            6.81,3.65A11.36,11.36,0,0,1,6.89,8a4,4,0,0,0-
            .54,2,4,4,0,0,0,1.78,3.33,4,4,0,0,1-1.81-
            .5v.05a4,4,0,0,0,3.2,3.92,4,4,0,0,1-1,.14,3.67,3.67,0,0,1-.75-
            .07,4,4,0,0,0,3.74,2.78,8.06,8.06,0,0,1-5,1.71,7.61,7.61,0,0,1-1-
            .06,11.31,11.31,0,0,0,6.14,1.8'/%3e%3c/svg%3e");
        }
        .firefox {
            background-image: url("data:image/svg+xml;charset=UTF-8,%3csvg
            id='Layer_1' data-name='Layer 1' xmlns='http://www.w3.org/2000/svg'
            viewBox='0 0 32 32'%3e%3cdefs%3e%3cstyle%3e.cls-1%7bfill:%2346bf56;%7d.cls-
            2%7bfill:%23fff;%7d%3c/style%3e%3c/defs%3e%3ctitle%3eIcon%3c/title%3e%3crect
            class='cls-1' x='1' y='1' width='28' height='28' rx='5.75'
            ry='5.75'/%3e%3cpath class='cls-2'
            d='M15.19,4.68A10.21,10.21,0,0,0,6.4,20.12l-1.29,5.2,5.12-
            1.47a10.23,10.23,0,1,0,5-19.17Zm.11,18.75a8.41,8.41,0,0,1-4.81-1.51-
            2.91.85.74-3a8.49,8.49,0,1,1,7,3.66Z'/%3e%3cpath class='cls-2'
            d='M12.6,14.09l.36-.41c.2-.23.5-.45.51-.78a2.3,2.3,0,0,0-.21-.93c-.07-.19-
            .15-.37-.22-.56a4,4,0,0,0-.48-1,.94.94,0,0,0-1-.16,2.52,2.52,0,0,0-
            1.39,2.9,9.09,9.09,0,0,0,7.22,6.56s2.16.31,2.74-1a2.39,2.39,0,0,0,.19-
            1,.81.81,0,0,0-.48-.63,10.88,10.88,0,0,0-1-.53,1.51,1.51,0,0,0-1-
            .29,1.64,1.64,0,0,0-.63.51c-.23.25-.47.49-
            .71.74C16.46,17.49,13.41,16.37,12.6,14.09Z'/%3e%3c/svg%3e");
        }
        .svg_icons {
            width: 32px;
            height: 32px;
            display: inline-block;
        }
    </style>

```

{% previewsample "page.domainurl/code-snippet/badge/circle-cs1" %}

### Pill

The pill badge style can be applied by adding the modifier class `e-badge-pill` to the target element.

### APP.VUE

```

<template>
    <div id='element'>
        <h1>Badge Component <span class="e-badge e-badge-primary e-badge-
        pill">New</span></h1>
    </div>
</template>
<script>
import Vue from "vue";
export default {
    data() {

```



```

    return {};
  }
}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-vue-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-
notifications/styles/material.css";
#element {
  display: flex;
  width: 400px;
  margin: auto;
  border: 1px solid #dddddd;
  border-radius: 3px;
  justify-content: center;
  position: relative;
  top: 130px;
}
</style>

```

{% previewsample "page.domainurl/code-snippet/badge/pill-cs1" %}

#### [Link](#)

When badge modifier classes are applied to the anchor tag, the badge's appearance will change from normal state to hover state on mouseover.

#### **APP.VUE**

```

<template>
  <div id='element'>
    <div style="display: inline-block; margin-top: 15px;">
      <a href="#" class="e-badge e-badge-primary">Link Badge</a>
    </div>
  </div>
</template>
<script>
import Vue from "vue";
export default {
  data() {
    return {};
  }
}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-vue-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-
notifications/styles/material.css";
#element {
  display: flex;
  width: 400px;
  height: 50px;
  margin: auto;
  border: 1px solid #dddddd;
  border-radius: 3px;
  justify-content: center;
  position: relative;
}

```

```

        top: 130px;
    }
</style>

```

{% previewsample "page.domainurl/code-snippet/badge/link-cs1" %}

### Notification

The notification badge style can be applied by adding the modifier class `.e-badge-notification` to the target element. Notification badges are used when a content or a context needs special attention. While using the notification badge, set the parent element to `position: relative`.

### APP.VUE

```

<template>
  <div id='element'>
    <div style="position:relative;display:inline-block;margin:20px 20px 10px 20px;">
      <div class="skype svg_icons"></div>
      <span class="e-badge e-badge-success e-badge-overlap e-badge-notification">99</span>
    </div>
    <div style="position:relative;display:inline-block;margin:20px 20px 10px 20px;">
      <div class="twitter svg_icons"></div>
      <span class="e-badge e-badge-info e-badge-overlap e-badge-notification">27</span>
    </div>
    <div style="position:relative;display:inline-block;margin:20px 20px 10px 20px;">
      <div class="facebook svg_icons"></div>
      <span class="e-badge e-badge-info e-badge-overlap e-badge-notification">2</span>
    </div>
    <div style="position:relative;display:inline-block;margin:20px 20px 10px 20px;">
      <div class="firefox svg_icons"></div>
      <span class="e-badge e-badge-danger e-badge-overlap e-badge-notification">35</span>
    </div>
  </div>
</template>
<script>
import Vue from "vue";
export default {
  data() {
    return {};
  }
}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-vue-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-notifications/styles/material.css";
#element {
  display: flex;
  width: 400px;

```

```

margin: auto;
border: 1px solid #dddddd;
border-radius: 3px;
justify-content: center;
position: relative;
top: 130px;
}
.facebook {
background-image: url("data:image/svg+xml;charset=UTF-8,%3csvg
id='Layer_1' data-name='Layer 1' xmlns='http://www.w3.org/2000/svg'
viewBox='0 0 32 32'%3e%3cdefs%3e%3cstyle%3e.cls-1%7bfill:%233c5a99;%7d.cls-
2%7bfill:%23fff;%7d%3c/style%3e%3c/defs%3e%3ctitle%3eIcon%3c/title%3e%3crect
class='cls-1' x='1' y='1' width='28' height='28' rx='5.75'
ry='5.75'/%3e%3cpath class='cls-2'
d='M22.53,8.7h2.58V4.64H21.24a5.25,5.25,0,0,0-4,2s-1.06,1-
1.08,3.92h0v3H12.36v4.31h3.82V29h4.41V17.86h3.81.53-4.31H20.59v-
3h0A1.78,1.78,0,0,1,22.53,8.7Z'/%3e%3c/svg%3e");
}
.skype {
background-image: url("data:image/svg+xml;charset=UTF-8,%3csvg
id='Layer_1' data-name='Layer 1' xmlns='http://www.w3.org/2000/svg'
viewBox='0 0 32 32'%3e%3cdefs%3e%3cstyle%3e.cls-1%7bfill:%2331c4ed;%7d.cls-
2%7bfill:%23fff;fill-
rule:evenodd;%7d%3c/style%3e%3c/defs%3e%3ctitle%3eIcon%3c/title%3e%3crect
class='cls-1' x='1' y='1' width='28' height='28' rx='5.75'
ry='5.75'/%3e%3cpath class='cls-2' d='M19.5,19.14a4,4,0,0,1-
1.75,1.31,7.22,7.22,0,0,1-
2.68.46A6.6,6.6,0,0,1,12,20.27,4,4,0,0,1,10.58,19,2.71,2.71,0,0,1,10,17.51a1
.05,1.05,0,0,1,.36-.8,1.28,1.28,0,0,1,.9-
.33,1.14,1.14,0,0,1,.75.26,1.89,1.89,0,0,1,.51.73,4.54,4.54,0,0,0,.49.86,2,2
,0,0,0,.72.55A3.08,3.08,0,0,0,15,19a3,3,0,0,0,1.73-.45,1.23,1.23,0,0,0,.63-
1.06,1,1,0,0,0-.33-.8,2.3,2.3,0,0,0-.92-.49c-.39-.12-.92-.26-1.57-
.39a12.2,12.2,0,0,1-2.25-.67,3.64,3.64,0,0,1-1.48-1.06,2.47,2.47,0,0,1-.29-
.47,2.84,2.84,0,0,1-.26-1.22,2.71,2.71,0,0,1,.58-1.72,3.71,3.71,0,0,1,1.67-
1.14A7.37,7.37,0,0,1,15,9.14a7,7,0,0,1,2,.26,4.38,4.38,0,0,1,1.42.7,3.1,3.1,
0,0,1,.84.92,2.11,2.11,0,0,1,.26,1,1.13,1.13,0,0,1-.35.82,1.18,1.18,0,0,1-
.88.36,1.09,1.09,0,0,1-.74-.23,2.68,2.68,0,0,1-.51-.67,3,3,0,0,0-.77-
.94A2.42,2.42,0,0,0,14.87,11a2.76,2.76,0,0,0-1.49.36,1,1,0,0,0-
.53.81.78.78,0,0,0,.17.5,1.63,1.63,0,0,0,.52.38,4.48,4.48,0,0,0,.69.27l.15,0
,1,.24c.69.15,1.33.32,1.89.49a6.29,6.29,0,0,1,1.47.67,2.81,2.81,0,0,1,1,1,3,
3,0,0,1,.35,1.49,3.15,3.15,0,0,1-.6,1.89Zm4-2.23a7.73,7.73,0,0,0,.2-1.81c0-
.22,0-.43,0-.64a8.53,8.53,0,0,0-8.56-7.83,8.72,8.72,0,0,0-
1.46.12A5.08,5.08,0,0,0,11,6a5,5,0,0,0-
5,4.91,4.83,4.83,0,0,0,.68,2.48,7.33,7.33,0,0,0-
.13.94,6.51,6.51,0,0,0,0,.77,8.52,8.52,0,0,0,8.58,8.46,8.9,8.9,0,0,0,1.57-
.14A5.09,5.09,0,0,0,19,24a4.94,4.94,0,0,0,5-4.91,4.86,4.86,0,0,0-.52-
2.18Z'/%3e%3c/svg%3e");
}
.twitter {
background-image: url("data:image/svg+xml;charset=UTF-8,%3csvg
id='Layer_1' data-name='Layer 1' xmlns='http://www.w3.org/2000/svg'
viewBox='0 0 32 32'%3e%3cdefs%3e%3cstyle%3e.cls-1%7bfill:%231da1f2;%7d.cls-
2%7bfill:%23fff;%7d%3c/style%3e%3c/defs%3e%3ctitle%3eIcon%3c/title%3e%3crect
class='cls-1' x='1' y='1' width='28' height='28' rx='5.75'
ry='5.75'/%3e%3cpath class='cls-2'
d='M11.67,23.11A11.3,11.3,0,0,0,23.05,11.73c0-.17,0-.35,0-.52a8,8,0,0,0,2-
2.07,8,8,0,0,1-2.29.63,4,4,0,0,0,1.75-2.21,8,8,0,0,1-2.54,1,4,4,0,0,0-

```

```

6.81,3.65A11.36,11.36,0,0,1,6.89,8a4,4,0,0,0-
.54,2,4,4,0,0,0,1.78,3.33,4,4,0,0,1-1.81-
.5v.05a4,4,0,0,0,3.2,3.92,4,4,0,0,1-1,.14,3.67,3.67,0,0,1-.75-
.07,4,4,0,0,0,3.74,2.78,8.06,8.06,0,0,1-5,1.71,7.61,7.61,0,0,1-1-
.06,11.31,11.31,0,0,0,6.14,1.8'/%3e%3c/svg%3e");
}
.firefox {
background-image: url("data:image/svg+xml;charset=UTF-8,%3csvg
id='Layer_1' data-name='Layer 1' xmlns='http://www.w3.org/2000/svg'
viewBox='0 0 32 32'%3e%3cdefs%3e%3cstyle%3e.cls-1%7bfill:%2346bf56;%7d.cls-
2%7bfill:%23fff;%7d%3c/style%3e%3c/defs%3e%3ctitle%3eIcon%3c/title%3e%3crect
class='cls-1' x='1' y='1' width='28' height='28' rx='5.75'
ry='5.75'/%3e%3cpath class='cls-2'
d='M15.19,4.68A10.21,10.21,0,0,0,6.4,20.12l-1.29,5.2,5.12-
1.47a10.23,10.23,0,1,0,5-19.17Zm.11,18.75a8.41,8.41,0,0,1-4.81-1.51-
2.91.85.74-3a8.49,8.49,0,1,1,7,3.66Z'/%3e%3cpath class='cls-2'
d='M12.6,14.09l.36-.41c.2-.23.5-.45.51-.78a2.3,2.3,0,0,0-.21-.93c-.07-.19-
.15-.37-.22-.56a4,4,0,0,0-.48-1,.94.94,0,0,0-1-.16,2.52,2.52,0,0,0-
1.39,2.9,9.09,9.09,0,0,0,7.22,6.56s2.16.31,2.74-1a2.39,2.39,0,0,0,.19-
1,.81.81,0,0,0-.48-.63,10.88,10.88,0,0,0-1-.53,1.51,1.51,0,0,0-1-
.29,1.64,1.64,0,0,0-.63.51c-.23.25-.47.49-
.71.74C16.46,17.49,13.41,16.37,12.6,14.09Z'/%3e%3c/svg%3e");
}
.svg_icons {
width: 32px;
height: 32px;
display: inline-block;
}
</style>

```

{% previewsample "page.domainurl/code-snippet/badge/notification-cs1" %}

### Dot

Dot can be applied by adding the modifier class `e-badge-dot` to the target element. Dot badges are similar to notification badges, but in a minimalistic way. While using the dot badge, set the parent element to `position: relative`.

### APP.VUE

```

<template>
  <div id='element'>
    <div style="position:relative;display:inline-block;margin:20px 20px
10px 20px;">
      <div class="skype svg_icons"></div>
      <span class="e-badge e-badge-success e-badge-overlap e-badge-
dot"></span>
    </div>
    <div style="position:relative;display:inline-block;margin:20px 20px
10px 20px;">
      <div class="twitter svg_icons"></div>
      <span class="e-badge e-badge-info e-badge-overlap e-badge-
dot"></span>
    </div>
    <div style="position:relative;display:inline-block;margin:20px 20px
10px 20px;">
      <div class="facebook svg_icons"></div>

```

```

        <span class="e-badge e-badge-info e-badge-overlap e-badge-
dot"></span>
      </div>
      <div style="position:relative;display:inline-block;margin:20px 20px
10px 20px;">
        <div class="firefox svg_icons"></div>
        <span class="e-badge e-badge-danger e-badge-overlap e-badge-
dot"></span>
      </div>
    </div>
  </template>
</script>
<script>
import Vue from "vue";
export default {
  data() {
    return {};
  }
}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-vue-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-
notifications/styles/material.css";
#element {
  display: flex;
  width: 400px;
  margin: auto;
  border: 1px solid #dddddd;
  border-radius: 3px;
  justify-content: center;
  position: relative;
  top: 130px;
}
.facebook {
  background-image: url("data:image/svg+xml;charset=UTF-8,%3csvg
id='Layer_1' data-name='Layer 1' xmlns='http://www.w3.org/2000/svg'
viewBox='0 0 32 32'%3e%3cdefs%3e%3cstyle%3e.cls-1%7bfill:%233c5a99;%7d.cls-
2%7bfill:%23fff;%7d%3c/style%3e%3c/defs%3e%3ctitle%3eIcon%3c/title%3e%3crect
class='cls-1' x='1' y='1' width='28' height='28' rx='5.75'
ry='5.75'/%3e%3cpath class='cls-2'
d='M22.53,8.7h2.58V4.64H21.24a5.25,5.25,0,0,0-4,2s-1.06,1-
1.08,3.92h0v3H12.36v4.31h3.82V29h4.41V17.86h3.81.53-4.31H20.59v-
3h0A1.78,1.78,0,0,1,22.53,8.7Z'/%3e%3c/svg%3e");
}
.skype {
  background-image: url("data:image/svg+xml;charset=UTF-8,%3csvg
id='Layer_1' data-name='Layer 1' xmlns='http://www.w3.org/2000/svg'
viewBox='0 0 32 32'%3e%3cdefs%3e%3cstyle%3e.cls-1%7bfill:%2331c4ed;%7d.cls-
2%7bfill:%23fff;fill-
rule:evenodd;%7d%3c/style%3e%3c/defs%3e%3ctitle%3eIcon%3c/title%3e%3crect
class='cls-1' x='1' y='1' width='28' height='28' rx='5.75'
ry='5.75'/%3e%3cpath class='cls-2' d='M19.5,19.14a4,4,0,0,1-
1.75,1.31,7.22,7.22,0,0,1-
2.68.46A6.6,6.6,0,0,1,12,20.27,4,4,0,0,1,10.58,19,2.71,2.71,0,0,1,10,17.51a1
.05,1.05,0,0,1,.36-.8,1.28,1.28,0,0,1,.9-
.33,1.14,1.14,0,0,1,.75.26,1.89,1.89,0,0,1,.51.73,4.54,4.54,0,0,0,.49.86,2,2
,0,0,0,.72.55A3.08,3.08,0,0,0,15,19a3,3,0,0,0,1.73-.45,1.23,1.23,0,0,0,.63-

```

```

1.06,1,1,0,0,0,-.33-.8,2.3,2.3,0,0,0,-.92-.49c-.39-.12-.92-.26-1.57-
.39a12.2,12.2,0,0,1-2.25-.67,3.64,3.64,0,0,1-1.48-1.06,2.47,2.47,0,0,1-.29-
.47,2.84,2.84,0,0,1-.26-1.22,2.71,2.71,0,0,1,.58-1.72,3.71,3.71,0,0,1,1.67-
1.14A7.37,7.37,0,0,1,15,9.14a7,7,0,0,1,2,.26,4.38,4.38,0,0,1,1.42.7,3.1,3.1,
0,0,1,.84.92,2.11,2.11,0,0,1,.26,1,1.13,1.13,0,0,1-.35.82,1.18,1.18,0,0,1-
.88.36,1.09,1.09,0,0,1-.74-.23,2.68,2.68,0,0,1-.51-.67,3,3,0,0,0-.77-
.94A2.42,2.42,0,0,0,14.87,11a2.76,2.76,0,0,0-1.49.36,1,1,0,0,0-
.53.81.78.78,0,0,0,.17.5,1.63,1.63,0,0,0,.52.38,4.48,4.48,0,0,0,.69.271.15,0
,1,.24c.69.15,1.33.32,1.89.49a6.29,6.29,0,0,1,1.47.67,2.81,2.81,0,0,1,1,1,3,
3,0,0,1,.35,1.49,3.15,3.15,0,0,1-.6,1.89Zm4-2.23a7.73,7.73,0,0,0,.2-1.81c0-
.22,0-.43,0-.64a8.53,8.53,0,0,0-8.56-7.83,8.72,8.72,0,0,0-
1.46.12A5.08,5.08,0,0,0,11,6a5,5,0,0,0-
5,4.91,4.83,4.83,0,0,0,.68,2.48,7.33,7.33,0,0,0-
.13.94,6.51,6.51,0,0,0,0,.77,8.52,8.52,0,0,0,8.58,8.46,8.9,8.9,0,0,0,1.57-
.14A5.09,5.09,0,0,0,19,24a4.94,4.94,0,0,0,5-4.91,4.86,4.86,0,0,0-.52-
2.18Z'/%3e%3c/svg%3e");
    }
    .twitter {
      background-image: url("data:image/svg+xml;charset=UTF-8,%3csvg
id='Layer_1' data-name='Layer 1' xmlns='http://www.w3.org/2000/svg'
viewBox='0 0 32 32'%3e%3cdefs%3e%3cstyle%3e.cls-1%7bfill:%231da1f2;%7d.cls-
2%7bfill:%23fff;%7d%3c/style%3e%3c/defs%3e%3ctitle%3eIcon%3c/title%3e%3crect
class='cls-1' x='1' y='1' width='28' height='28' rx='5.75'
ry='5.75'/%3e%3cpath class='cls-2'
d='M11.67,23.11A11.3,11.3,0,0,0,23.05,11.73c0-.17,0-.35,0-.52a8,8,0,0,0,2-
2.07,8,8,0,0,1-2.29.63,4,4,0,0,0,1.75-2.21,8,8,0,0,1-2.54,1,4,4,0,0,0-
6.81,3.65A11.36,11.36,0,0,1,6.89,8a4,4,0,0,0-
.54,2,4,4,0,0,0,1.78,3.33,4,4,0,0,1-1.81-
.5v.05a4,4,0,0,0,3.2,3.92,4,4,0,0,1-1,.14,3.67,3.67,0,0,1-.75-
.07,4,4,0,0,0,3.74,2.78,8.06,8.06,0,0,1-5,1.71,7.61,7.61,0,0,1-1-
.06,11.31,11.31,0,0,0,6.14,1.8'/%3e%3c/svg%3e");
    }
    .firefox {
      background-image: url("data:image/svg+xml;charset=UTF-8,%3csvg
id='Layer_1' data-name='Layer 1' xmlns='http://www.w3.org/2000/svg'
viewBox='0 0 32 32'%3e%3cdefs%3e%3cstyle%3e.cls-1%7bfill:%2346bf56;%7d.cls-
2%7bfill:%23fff;%7d%3c/style%3e%3c/defs%3e%3ctitle%3eIcon%3c/title%3e%3crect
class='cls-1' x='1' y='1' width='28' height='28' rx='5.75'
ry='5.75'/%3e%3cpath class='cls-2'
d='M15.19,4.68A10.21,10.21,0,0,0,6.4,20.12l-1.29,5.2,5.12-
1.47a10.23,10.23,0,1,0,5-19.17Zm.11,18.75a8.41,8.41,0,0,1-4.81-1.51-
2.91.85.74-3a8.49,8.49,0,1,1,7,3.66Z'/%3e%3cpath class='cls-2'
d='M12.6,14.09l.36-.41c.2-.23.5-.45.51-.78a2.3,2.3,0,0,0-.21-.93c-.07-.19-
.15-.37-.22-.56a4,4,0,0,0-.48-1,.94.94,0,0,0-1-.16,2.52,2.52,0,0,0-
1.39,2.9,9.09,9.09,0,0,0,7.22,6.56s2.16.31,2.74-1a2.39,2.39,0,0,0,.19-
1,.81.81,0,0,0-.48-.63,10.88,10.88,0,0,0-1-.53,1.51,1.51,0,0,0-1-
.29,1.64,1.64,0,0,0-.63.51c-.23.25-.47.49-
.71.74C16.46,17.49,13.41,16.37,12.6,14.09Z'/%3e%3c/svg%3e");
    }
    .svg_icons {
      width: 32px;
      height: 32px;
      display: inline-block;
    }
    .svg_icons + .e-badge.e-badge-overlap {
      transform: translateX(-100%);
    }
  }

```

```
</style>
```

```
{% previewsample "page.domainurl/code-snippet/badge/dot-cs1" %}
```

### Overlap

The overlap badge can be used with notification or dot badge, which overlaps with the target element by adding the modifier class `e-badge-overlap`. While using the overlap badge, set the parent element to `position: relative`.

### APP.VUE

```
<template>
  <div id='element'>
    <div style="position:relative;display:inline-block;margin:20px 20px
10px 20px;">
      <div class="skype svg_icons"></div>
      <span class="e-badge e-badge-success e-badge-overlap e-badge-
notification">99+</span>
    </div>
    <div style="position:relative;display:inline-block;margin:20px 20px
10px 20px;">
      <div class="twitter svg_icons"></div>
      <span class="e-badge e-badge-info e-badge-overlap e-badge-
notification">27</span>
    </div>
    <div style="position:relative;display:inline-block;margin:20px 20px
10px 20px;">
      <div class="facebook svg_icons"></div>
      <span class="e-badge e-badge-info e-badge-overlap e-badge-
notification">2</span>
    </div>
    <div style="position:relative;display:inline-block;margin:20px 20px
10px 20px;">
      <div class="firefox svg_icons"></div>
      <span class="e-badge e-badge-danger e-badge-overlap e-badge-
notification">35</span>
    </div>
  </div>
</template>
<script>
import Vue from "vue";
export default {
  data() {
    return {};
  }
}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-vue-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-
notifications/styles/material.css";
#element {
  display: flex;
  width: 400px;
  margin: auto;
  border: 1px solid #dddddd;
```

```

border-radius: 3px;
justify-content: center;
position: relative;
top: 130px;
}
.facebook {
background-image: url("data:image/svg+xml;charset=UTF-8,%3csvg
id='Layer_1' data-name='Layer 1' xmlns='http://www.w3.org/2000/svg'
viewBox='0 0 32 32'%3e%3cdefs%3e%3cstyle%3e.cls-1%7bfill:%233c5a99;%7d.cls-
2%7bfill:%23fff;%7d%3c/style%3e%3c/defs%3e%3ctitle%3eIcon%3c/title%3e%3crect
class='cls-1' x='1' y='1' width='28' height='28' rx='5.75'
ry='5.75'/%3e%3cpath class='cls-2'
d='M22.53,8.7h2.58V4.64H21.24a5.25,5.25,0,0,0-4,2s-1.06,1-
1.08,3.92h0v3H12.36v4.31h3.82V29h4.41V17.86h3.81.53-4.31H20.59v-
3h0A1.78,1.78,0,0,1,22.53,8.7Z'/%3e%3c/svg%3e");
}
.skype {
background-image: url("data:image/svg+xml;charset=UTF-8,%3csvg
id='Layer_1' data-name='Layer 1' xmlns='http://www.w3.org/2000/svg'
viewBox='0 0 32 32'%3e%3cdefs%3e%3cstyle%3e.cls-1%7bfill:%2331c4ed;%7d.cls-
2%7bfill:%23fff;fill-
rule:evenodd;%7d%3c/style%3e%3c/defs%3e%3ctitle%3eIcon%3c/title%3e%3crect
class='cls-1' x='1' y='1' width='28' height='28' rx='5.75'
ry='5.75'/%3e%3cpath class='cls-2' d='M19.5,19.14a4,4,0,0,1-
1.75,1.31,7.22,7.22,0,0,1-
2.68,4.6A6.6,6.6,0,0,1,12,20.27,4,4,0,0,1,10.58,19,2.71,2.71,0,0,1,10,17.51a1
.05,1.05,0,0,1,.36-.8,1.28,1.28,0,0,1,.9-
.33,1.14,1.14,0,0,1,.75,2.6,1.89,1.89,0,0,1,.51,7.3,4.54,4.54,0,0,0,.49,8.6,2,2
,0,0,0,.72,5.5A3.08,3.08,0,0,0,15,19a3,3,0,0,0,1.73-.45,1.23,1.23,0,0,0,.63-
1.06,1,1,0,0,0-.33-.8,2.3,2.3,0,0,0-.92-.49c-.39-.12-.92-.26-1.57-
.39a12.2,12.2,0,0,1-2.25-.67,3.64,3.64,0,0,1-1.48-1.06,2.47,2.47,0,0,1-.29-
.47,2.84,2.84,0,0,1-.26-1.22,2.71,2.71,0,0,1,.58-1.72,3.71,3.71,0,0,1,1.67-
1.14A7.37,7.37,0,0,1,15,9.14a7,7,0,0,1,2,.26,4.38,4.38,0,0,1,1.42,7,3.1,3.1,
0,0,1,.84,9.2,2.11,2.11,0,0,1,.26,1,1.13,1.13,0,0,1-.35,8.2,1.18,1.18,0,0,1-
.88,3.6,1.09,1.09,0,0,1-.74-.23,2.68,2.68,0,0,1-.51-.67,3,3,0,0,0-.77-
.94A2.42,2.42,0,0,0,14.87,11a2.76,2.76,0,0,0-1.49,3.6,1,1,0,0,0-
.53,8.1,7.8,7.8,0,0,0,.17,5,1.63,1.63,0,0,0,.52,3.8,4.48,4.48,0,0,0,.69,2.7,1.15,0
,1,.24c,6.9,1.33,3.2,1.89,4.9a6.29,6.29,0,0,1,1.47,6.7,2.81,2.81,0,0,1,1,1,3,
3,0,0,1,.35,1.49,3.15,3.15,0,0,1-.6,1.89Zm4-2.23a7.73,7.73,0,0,0,.2-1.8,1c0-
.22,0-.43,0-.64a8.53,8.53,0,0,0-8.56-7.83,8.72,8.72,0,0,0-
1.46,1.2A5.08,5.08,0,0,0,11,6a5,5,0,0,0-
5,4.9,4.83,4.83,0,0,0,.68,2.48,7.33,7.33,0,0,0-
.13,9.4,6.5,6.5,0,0,0,.77,8.52,8.52,0,0,0,8.58,8.46,8.9,8.9,0,0,0,1.57-
.14A5.09,5.09,0,0,0,19,24a4.94,4.94,0,0,0,5-4.9,4.86,4.86,0,0,0-.52-
2.18Z'/%3e%3c/svg%3e");
}
.twitter {
background-image: url("data:image/svg+xml;charset=UTF-8,%3csvg
id='Layer_1' data-name='Layer 1' xmlns='http://www.w3.org/2000/svg'
viewBox='0 0 32 32'%3e%3cdefs%3e%3cstyle%3e.cls-1%7bfill:%231dalf2;%7d.cls-
2%7bfill:%23fff;%7d%3c/style%3e%3c/defs%3e%3ctitle%3eIcon%3c/title%3e%3crect
class='cls-1' x='1' y='1' width='28' height='28' rx='5.75'
ry='5.75'/%3e%3cpath class='cls-2'
d='M11.67,23.11A11.3,11.3,0,0,0,23.05,11.73c0-.17,0-.35,0-.52a8,8,0,0,0,2-
2.07,8,8,0,0,1-2.29,6.3,4,4,0,0,0,1.75-2.21,8,8,0,0,1-2.54,1,4,4,0,0,0-
6.81,3.65A11.36,11.36,0,0,1,6.89,8a4,4,0,0,0-
.54,2,4,4,0,0,0,1.78,3.33,4,4,0,0,1-1.81-

```



```
.5v.05a4,4,0,0,0,3.2,3.92,4,4,0,0,1-1,.14,3.67,3.67,0,0,1-.75-
.07,4,4,0,0,0,3.74,2.78,8.06,8.06,0,0,1-5,1.71,7.61,7.61,0,0,1-1-
.06,11.31,11.31,0,0,0,6.14,1.8'/%3e%3c/svg%3e");
}
.firefox {
background-image: url("data:image/svg+xml;charset=UTF-8,%3csvg
id='Layer_1' data-name='Layer 1' xmlns='http://www.w3.org/2000/svg'
viewBox='0 0 32 32'%3e%3cdefs%3e%3cstyle%3e.cls-1%7bfill:%2346bf56;%7d.cls-
2%7bfill:%23fff;%7d%3c/style%3e%3c/defs%3e%3ctitle%3eIcon%3c/title%3e%3crect
class='cls-1' x='1' y='1' width='28' height='28' rx='5.75'
ry='5.75'/%3e%3cpath class='cls-2'
d='M15.19,4.68A10.21,10.21,0,0,0,6.4,20.12l-1.29,5.2,5.12-
1.47a10.23,10.23,0,1,0,5-19.17Zm.11,18.75a8.41,8.41,0,0,1-4.81-1.51-
2.91.85.74-3a8.49,8.49,0,1,1,7,3.66Z'/%3e%3cpath class='cls-2'
d='M12.6,14.09l.36-.41c.2-.23.5-.45.51-.78a2.3,2.3,0,0,0-.21-.93c-.07-.19-
.15-.37-.22-.56a4,4,0,0,0-.48-1,.94.94,0,0,0-1-.16,2.52,2.52,0,0,0-
1.39,2.9,9.09,9.09,0,0,0,7.22,6.56s2.16.31,2.74-1a2.39,2.39,0,0,0,.19-
1,.81.81,0,0,0-.48-.63,10.88,10.88,0,0,0-1-.53,1.51,1.51,0,0,0-1-
.29,1.64,1.64,0,0,0-.63.51c-.23.25-.47.49-
.71.74C16.46,17.49,13.41,16.37,12.6,14.09Z'/%3e%3c/svg%3e");
}
.svg_icons {
width: 32px;
height: 32px;
display: inline-block;
}
</style>
```

```
{% previewsample "page.domainurl/code-snippet/badge/overlap-cs1" %}
```

### Position

The default position of the notification or dot badge is top. But, the position can be changed to **bottom** using the modifier class **.e-badge-bottom**. For example, the bottom class modifier is used with dot badge to display the status in the avatar as shown in the following sample.

### APP.VUE

```
<template>
  <div id='element'>
    <div class="badge-block">
      <div class="contact svg_icons"></div>
      <!-- Success Colored Bottom Dot Badge -->
      <span class="e-badge e-badge-success e-badge-overlap e-badge-dot
e-badge-bottom"></span>
    </div>
    <div class="badge-block">
      <div class="skype svg_icons"></div>
      <!-- Info Colored Bottom Dot Badge -->
      <span class="e-badge e-badge-info e-badge-overlap e-badge-dot e-
badge-bottom"></span>
    </div>
    <div class="badge-block">
      <div class="facebook svg_icons"></div>
      <!-- Info Colored Dot Badge -->
      <span class="e-badge e-badge-info e-badge-overlap e-badge-
dot"></span>
    </div>
  </div>
</template>
```

```

        </div>
        <div class="badge-block">
            <div class="pinterest svg_icons"></div>
            <!-- Danger Colored Dot Badge -->
            <span class="e-badge e-badge-danger e-badge-overlap e-badge-dot
e-badge-bottom"></span>
        </div>
    </div>
</template>
<script>
import Vue from "vue";
export default {
    data() {
        return {};
    }
}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-vue-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-
notifications/styles/material.css";
#element {
    display: flex;
    width: 400px;
    margin: auto;
    border: 1px solid #dddddd;
    border-radius: 3px;
    justify-content: center;
    position: relative;
    top: 130px;
}
.badge-block {
    position: relative;
    display: inline-block;
    margin: 10px 13px 10px 10px;
}
/* SVG Icons */
.facebook {
    background-image: url("data:image/svg+xml;charset=UTF-8,%3csvg
id='Layer_1' data-name='Layer 1' xmlns='http://www.w3.org/2000/svg'
viewBox='0 0 32 32'%3e%3cdefs%3e%3cstyle%3e.cls-1%7bfill:%233c5a99;%7d.cls-
2%7bfill:%23fff;%7d%3c/style%3e%3c/defs%3e%3ctitle%3eIcon%3c/title%3e%3crect
class='cls-1' x='1' y='1' width='28' height='28' rx='5.75'
ry='5.75'%3e%3cpath class='cls-2'
d='M22.53,8.7h2.58V4.64H21.24a5.25,5.25,0,0,0-4,2s-1.06,1-
1.08,3.92h0v3H12.36v4.31h3.82V29h4.41V17.86h3.81.53-4.31H20.59v-
3h0A1.78,1.78,0,0,1,22.53,8.7Z'%3e%3c/svg%3e");
}
.skype {
    background-image: url("data:image/svg+xml;charset=UTF-8,%3csvg
id='Layer_1' data-name='Layer 1' xmlns='http://www.w3.org/2000/svg'
viewBox='0 0 32 32'%3e%3cdefs%3e%3cstyle%3e.cls-1%7bfill:%2331c4ed;%7d.cls-
2%7bfill:%23fff;fill-
rule:evenodd;%7d%3c/style%3e%3c/defs%3e%3ctitle%3eIcon%3c/title%3e%3crect
class='cls-1' x='1' y='1' width='28' height='28' rx='5.75'
ry='5.75'%3e%3cpath class='cls-2' d='M19.5,19.14a4,4,0,0,1-
1.75,1.31,7.22,7.22,0,0,1-

```

```
2.68.46A6.6,6.6,0,0,1,12,20.27,4,4,0,0,1,10.58,19,2.71,2.71,0,0,1,10,17.51a1
.05,1.05,0,0,1,.36-.8,1.28,1.28,0,0,1,.9-
.33,1.14,1.14,0,0,1,.75.26,1.89,1.89,0,0,1,.51.73,4.54,4.54,0,0,0,.49.86,2,2
,0,0,0,.72.55A3.08,3.08,0,0,0,15,19a3,3,0,0,0,1.73-.45,1.23,1.23,0,0,0,.63-
1.06,1,1,0,0,0-.33-.8,2.3,2.3,0,0,0-.92-.49c-.39-.12-.92-.26-1.57-
.39a12.2,12.2,0,0,1-2.25-.67,3.64,3.64,0,0,1-1.48-1.06,2.47,2.47,0,0,1-.29-
.47,2.84,2.84,0,0,1-.26-1.22,2.71,2.71,0,0,1,.58-1.72,3.71,3.71,0,0,1,1.67-
1.14A7.37,7.37,0,0,1,15,9.14a7,7,0,0,1,2,.26,4.38,4.38,0,0,1,1.42.7,3.1,3.1,
0,0,1,.84.92,2.11,2.11,0,0,1,.26,1,1.13,1.13,0,0,1-.35.82,1.18,1.18,0,0,1-
.88.36,1.09,1.09,0,0,1-.74-.23,2.68,2.68,0,0,1-.51-.67,3,3,0,0,0-.77-
.94A2.42,2.42,0,0,0,14.87,11a2.76,2.76,0,0,0-1.49.36,1,1,0,0,0-
.53.81.78,0,0,0,.17.5,1.63,1.63,0,0,0,.52.38,4.48,4.48,0,0,0,.69.271.15,0
,1,.24c.69.15,1.33.32,1.89.49a6.29,6.29,0,0,1,1.47.67,2.81,2.81,0,0,1,1,1,3,
3,0,0,1,.35,1.49,3.15,3.15,0,0,1-.6,1.89Zm4-2.23a7.73,7.73,0,0,0,.2-1.81c0-
.22,0-.43,0-.64a8.53,8.53,0,0,0-8.56-7.83,8.72,8.72,0,0,0-
1.46.12A5.08,5.08,0,0,0,11,6a5,5,0,0,0-
5,4.91,4.83,4.83,0,0,0,.68,2.48,7.33,7.33,0,0,0-
.13.94,6.51,6.51,0,0,0,0,.77,8.52,8.52,0,0,0,8.58,8.46,8.9,8.9,0,0,0,1.57-
.14A5.09,5.09,0,0,0,19,24a4.94,4.94,0,0,0,5-4.91,4.86,4.86,0,0,0-.52-
2.18Z'/%3e%3c/svg%3e");
}
.twitter {
  background-image: url("data:image/svg+xml;charset=UTF-8,%3csvg
id='Layer_1' data-name='Layer 1' xmlns='http://www.w3.org/2000/svg'
viewBox='0 0 32 32'%3e%3cdefs%3e%3cstyle%3e.cls-1%7bfill:%231da1f2;%7d.cls-
2%7bfill:%23fff;%7d%3c/style%3e%3c/defs%3e%3ctitle%3eIcon%3c/title%3e%3crect
class='cls-1' x='1' y='1' width='28' height='28' rx='5.75'
ry='5.75'%3e%3cpath class='cls-2'
d='M11.67,23.11A11.3,11.3,0,0,0,23.05,11.73c0-.17,0-.35,0-.52a8,8,0,0,0,2-
2.07,8,8,0,0,1-2.29.63,4,4,0,0,0,1.75-2.21,8,8,0,0,1-2.54,1,4,4,0,0,0-
6.81,3.65A11.36,11.36,0,0,1,6.89,8a4,4,0,0,0-
.54,2,4,4,0,0,1.78,3.33,4,4,0,0,1-1.81-
.5v.05a4,4,0,0,0,3.2,3.92,4,4,0,0,1-1,.14,3.67,3.67,0,0,1-.75-
.07,4,4,0,0,0,3.74,2.78,8.06,8.06,0,0,1-5,1.71,7.61,7.61,0,0,1-1-
.06,11.31,11.31,0,0,0,6.14,1.8'/%3e%3c/svg%3e");
}
.whatsapp {
  background: transparent url("data:image/svg+xml,%3Csvg
xmlns='http://www.w3.org/2000/svg' viewBox='0 0 32 32'%3E%3Cpath
fill='#25D366' d='M11.863 8.305c-.444-.011-.813.335-1.18.677-.12.11-
.235.228-.344.35-.489.552-.798 1.186-.703 1.935.103.807.187 1.63.423
2.402.65 2.13 1.962 3.843 3.605 5.309 1.51 1.346 3.162 2.47 5.151
2.977a9.847 9.847 0 0 0 1.966.268c.984.044 2.378-.942 2.728-1.86a.37.37 0 0
0 .022-.158c-.028-.435-.052-.87-.097-1.303-.009-.088-.083-.212-.158-
.242a276.61 276.61 0 0 0-3.316-1.281c-.277-.105-.526-.057-.749.159-.337.325-
.707.617-1.029.956-.207.217-.384.257-.617.078-.619-.48-1.26-.933-1.853-
1.443-.93-.8-1.7-1.744-2.38-2.763-.19-.284-.199-.483.077-.724.294-.256.517-
.592.804-.856.366-.336.37-.69.192-1.12a85.3 85.3 0 0 1-.964-2.46c-.3-.792-
.294-.795-1.258-.833a.894.894 0 0 0-.32-.068zm4.016-5.566c5.806-.062 10.95
3.973 12.154 9.89 1.123 5.523-1.825 11.269-6.975 13.536-3.56 1.568-7.1
1.477-10.606-.216-.13-.063-.337-.05-.479.004-1.39.532-2.777 1.08-4.164
1.625-.069.027-.14.049-.266.093.412-1.455.8-2.847 1.205-4.235.064-.216.03-
.357-.112-.526-2.056-2.454-3.018-5.292-2.889-8.483.226-5.636 4.415-10.442
9.968-11.475.727-.136 1.45-.205 2.164-.213zm.203-2.086a14.416 14.416 0 0 0-
4.595.74c-5.636 1.87-9.435 6.872-9.764 12.803-.189 3.43.773 6.54 2.82
9.307.133.182.178.333.12.563-.576 2.266-1.137 4.535-1.718 6.866.178-.065.31-
.111.438-.162 2.147-.838 4.297-1.67 6.44-2.521.324-.128.598-.141.925-.005
```

```

1.091.452 2.228.737 3.4.883 3.548.44 6.831-.306 9.797-2.291 5.426-3.631
7.685-10.342 5.562-16.54A14.243 14.243 0 0 0 16.082.654zM16.032
0c.494.004.992.03 1.492.078 6.504.613 12.026 5.686 13.158 12.108.805 4.565-
.231 8.687-3.16 12.275-2.333 2.857-5.366 4.599-9.012 5.227a14.618 14.618 0 0
1-7.895-.793.818.818 0 0 0-.649 0c-2.55 1.005-5.105 1.999-7.659 2.996-
.08.031-.162.058-.305.1081.783-3.13c.393-1.573.791-3.144 1.172-4.72a.617.617
0 0 0-.089-.442c-1.92-2.71-2.94-5.725-2.8-9.04.26-6.171 3.233-10.635 8.781-
13.33C11.81.386 13.89-.017 16.032 0z'/%3E%3C/svg%3E") no-repeat 100% 100%;
    }
    .firefox {
        background-image: url("data:image/svg+xml;charset=UTF-8,%3Csvg
id='Layer_1' data-name='Layer 1' xmlns='http://www.w3.org/2000/svg'
viewBox='0 0 32 32'%3E%3Cdefs%3E%3Cstyle%3E.cls-1%7Bfill:%2346bf56;%7d.cls-
2%7Bfill:%23fff;%7d%3C/style%3E%3Cdefs%3E%3Ctitle%3EIcon%3C/title%3E%3Crect
class='cls-1' x='1' y='1' width='28' height='28' rx='5.75'
ry='5.75'/%3E%3Cpath class='cls-2'
d='M15.19,4.68A10.21,10.21,0,0,0,6.4,20.12l-1.29,5.2,5.12-
1.47a10.23,10.23,0,1,0,5-19.17Zm.11,18.75a8.41,8.41,0,0,1-4.81-1.51-
2.91.85.74-3a8.49,8.49,0,1,1,7,3.66Z'/%3E%3Cpath class='cls-2'
d='M12.6,14.09l.36-.41c.2-.23.5-.45.51-.78a2.3,2.3,0,0,0-.21-.93c-.07-.19-
.15-.37-.22-.56a4,4,0,0,0-.48-1,.94.94,0,0,0-1-.16,2.52,2.52,0,0,0-
1.39,2.9,9.09,9.09,0,0,0,7.22,6.56s2.16.31,2.74-1a2.39,2.39,0,0,0,.19-
1,.81.81,0,0,0-.48-.63,10.88,10.88,0,0,0-1-.53,1.51,1.51,0,0,0-1-
.29,1.64,1.64,0,0,0-.63.51c-.23.25-.47.49-
.71.74C16.46,17.49,13.41,16.37,12.6,14.09Z'/%3E%3C/svg%3E");
    }
    .contact {
        background: transparent url("data:image/svg+xml,%3Csvg
xmlns='http://www.w3.org/2000/svg' viewBox='0 0 32 32'%3E%3Cellipse cx='16'
cy='16' fill='%234285f4' rx='16' ry='16'/%3E%3Cpath fill='%23FFF' d='M13.55
16.95h4.9c2.7 0 4.85 2.05 4.85 4.6 0 .9-.25 1.75-.75 2.45H9.45c-.5-.7-.75-
1.55-.75-2.45 0-2.55 2.15-4.6 4.85-4.6zM16.05 8c2.05 0 3.7 1.65 3.7 3.7 0
2.05-1.65 3.7-3.7 3.7-2.05 0-3.7-1.65-3.7-3.7.05-2.05 1.7-3.7 3.7-
3.7z'/%3E%3C/svg%3E") no-repeat 100% 100%;
    }
    .chrome {
        background: transparent url("data:image/svg+xml,%3Csvg
xmlns='http://www.w3.org/2000/svg' viewBox='0 0 32 32'%3E%3Cpath
fill='#ffffff' d='M16.033 11.049a5.155 5.155 0 1 1 0 10.312 5.156 5.156 0 0
1 0-10.312zM16.124 0c1.281-.003 9.659.318 14.268 9.043h-.016l.01.018c.33.578
3.745 6.94-.485 14.969 0 0-4.215 8.107-14.565 7.968l-.452-.012-.004.007-
.004.007.02-.037c.564-.98 5.112-8.884 6.357-11.067l.016-.028.007-.008.04-
.069.11-.127a7.085 7.085 0 0 0 1.457-2.967l.01-.046.035-.151c.088-.424.148-
.944.128-1.549l-.006-.117v-.004l-.007-.143-.006-.07-.005-.079-.012-.116v-
.011-.001-.008-.016-.158a7.2 7.2 0 0 0-.096-.572l-.018-.081-.013-.064a9.801
9.801 0 0 0-.692-2.016c-.165-.243-.332-.489-.512-.733l-.142-.187 8.728-
2.554s-10.538-.01-13.018-.001l.021.005H16.642l-.14-.013a7.034 7.034 0 0 0-
1.132-.003l-.167.016h-.047l-.034-.001c-.193.002-1.213.045-2.492.764l-
.005.003-.033.016a7.158 7.158 0 0 0-3.25 3.533l-.059.148-6.485-6.404s4.74
8.311 6.165 10.779l.065.113.023.088a7.14 7.14 0 0 0 7.777 5.118l.144-
.02L14.854 32h-.027c-.667-.005-7.894-.234-12.744-7.906 0 0-4.925-7.698.37-
16.573l.252-.411.001-.002C2.822 6.904 6.58.374 15.958.003c0 0 .057-.003.166-
.003z'/%3E%3C/svg%3E") no-repeat 100% 100%;
    }
    .pinterest {
        background: transparent url("data:image/svg+xml,%3Csvg
xmlns='http://www.w3.org/2000/svg' viewBox='0 0 32 32'%3E%3Cellipse cx='16'

```

```
cy='16' fill='%23bd081c' rx='16' ry='16'/%3E%3Cpath fill='%23FFF' d='M16.22
6.458c4.807-.009 9.028 1.888 9.663 5.307.787 4.256-2.438 8.866-8.213 8.535-
1.565-.09-2.222-.666-3.448-1.22-.675 2.628-1.5 5.147-3.942 6.463-.754-3.972
1.107-6.954 1.971-10.12-1.474-1.842.177-5.547 3.284-4.634 3.824 1.123-3.31
6.845 1.48 7.56 5 .746 7.04-6.441 3.94-8.779-4.48-3.376-13.042-.077-11.989
4.755.256 1.181 1.9 1.54.657 3.17-2.868-.471-3.724-2.15-3.614-4.39.178-3.664
4.435-6.229 8.705-6.583.506-.043 1.01-.064 1.507-.064z'/%3E%3C/svg%3E") no-
repeat 100% 100%;
}
.svg_icons {
width: 32px;
height: 32px;
display: inline-block;
}
.svg_icons + .e-badge.e-badge-overlap {
transform: translateX(-100%);
}
}
</style>
```

```
{% previewsample "page.domainurl/code-snippet/badge/position-cs1" %}
```

## How To

### Badge customization in Vue Badge component

#### *Colour customization*

Even though badges come with **8 predefined colors**, you can also customize the colour of the badge as desired.

#### **APP.VUE**

```
<template>
  <div id='element'>
    <h1>Color Customization <span class="e-badge e-badge-primary e-
badge-pill green">New</span></h1>
    <h1>Color Customization <span class="e-badge e-badge-primary e-
badge-pill bue">New</span></h1>
    <h1>Color Customization <span class="e-badge e-badge-primary e-
badge-pill purple">New</span></h1>
    <h1>Color Customization <span class="e-badge e-badge-primary e-
badge-pill gradient">New</span></h1>
  </div>
</template>
<script>
import Vue from "vue";
export default {
  data() {
    return {};
  }
}
</script>
<style>
  @import "../node_modules/@syncfusion/ej2-base/styles/material.css";
  @import "../node_modules/@syncfusion/ej2-vue-
notifications/styles/material.css";
  #element {
    display: table;
  }
</style>
```

```

width: 560px;
margin: auto;
height: 200px;
border: 1px solid #dddddd;
border-radius: 3px;
justify-content: center;
position: relative;
top: 55px;
}
h1 {
text-align: center;
}
#element .e-badge.green {
background: #329378;
color: #fff;
}
#element .e-badge.blue {
background: #5f65b8;
color: #fff;
}
#element .e-badge.gradient {
background: #4776E6;
/* fallback for old browsers */
background: -webkit-linear-gradient(to top, #8E54E9, #4776E6);
/* Chrome 10-25, Safari 5.1-6 */
background: linear-gradient(to top, #8E54E9, #4776E6);
/* W3C, IE 10+/ Edge, Firefox 16+, Chrome 26+, Opera 12+, Safari 7+ */
color: #fff;
}
#element .e-badge.purple {
background: #9a428f;
color: #fff;
}
}
</style>

```

```
{% previewsample "page.domainurl/code-snippet/badge/color-cs1" %}
```

### Customize badge size

Badges are designed to change its size based on the content. To change the size of a badge, adjust the font size of the badge.

### APP.VUE

```

<template>
  <div id='element'>
    <h1>Badge Component <span class="e-badge e-badge-primary
size_1">New</span></h1>
    <h1>Badge Component <span class="e-badge e-badge-primary
size_2">New</span></h1>
    <h1>Badge Component <span class="e-badge e-badge-primary
size_3">New</span></h1>
  </div>
</template>
<script>
import Vue from "vue";
export default {

```

```

data() {
  return {};
}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-
notifications/styles/material.css";
#element {
  display: block;
  width: 400px;
  margin: auto;
  border: 1px solid #dddddd;
  border-radius: 3px;
  justify-content: center;
}
h1 {
  text-align: center;
}
.e-badge.size_1 {
  font-size: 12px;
}
.e-badge.size_2 {
  font-size: 16px;
}
.e-badge.size_3 {
  font-size: 18px;
}
</style>

```

{% previewsample "page.domainurl/code-snippet/badge/size-cs1" %}

### Custom position

Even though the badges support the conventional **top** and **bottom** positions, the position of the badges can be changed as desired. This can be done by adding a custom class to the badge element to override the default position applied from the source.

### APP.VUE

```

<template>
  <div id='element'>
    <div style="width: 200px;margin: 10px auto;">
      <div class="badge-block">
        <div class="whatsapp svg_icons"></div>
        <!-- Warning Colored Notification Badge -->
        <span class="e-badge e-badge-warning e-badge-notification e-
badge-overlap leftTop">99+</span>
      </div>
      <div class="badge-block">
        <div class="facebook svg_icons"></div>
        <!-- Danger Colored Notification Badge -->
        <span class="e-badge e-badge-danger e-badge-notification e-
badge-overlap leftTop">99+</span>
      </div>
      <div class="badge-block">

```

```

        <div class="skype svg_icons"></div>
        <!-- Secondary Colored Notification Badge -->
        <span class="e-badge e-badge-secondary e-badge-notification
e-badge-overlap leftTop">18</span>
    </div>
</div>
<div style="width: 200px;margin: 10px auto;">
    <div class="badge-block">
        <div class="whatsapp svg_icons"></div>
        <!-- Warning Colored Notification Badge -->
        <span class="e-badge e-badge-warning e-badge-notification e-
badge-overlap leftBottom">99+</span>
    </div>
    <div class="badge-block">
        <div class="facebook svg_icons"></div>
        <!-- Danger Colored Notification Badge -->
        <span class="e-badge e-badge-danger e-badge-notification e-
badge-overlap leftBottom">99+</span>
    </div>
    <div class="badge-block">
        <div class="skype svg_icons"></div>
        <!-- Secondary Colored Notification Badge -->
        <span class="e-badge e-badge-secondary e-badge-notification
e-badge-overlap leftBottom">18</span>
    </div>
</div>
</div>
</template>
<script>
import Vue from "vue";
export default {
    data() {
        return {};
    }
}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-
notifications/styles/material.css";
#element {
    display: flex;
    width: 500px;
    margin: auto;
    border: 1px solid #dddddd;
    border-radius: 3px;
    justify-content: center;
    position: relative;
    top: 130px;
}
.badge-block {
    position: relative;
    display: inline-block;
    margin: 10px 13px 10px 10px;
}
.badge-block .e-badge.leftBottom {
    transform: translateX(-150%) translateY(200%);

```



```

    }
    .badge-block .e-badge.leftTop {
    transform: translateX(-150%);
    }
    /* SVG Icons */
    .facebook {
    background-image: url("data:image/svg+xml;charset=UTF-8,%3csvg
id='Layer_1' data-name='Layer 1' xmlns='http://www.w3.org/2000/svg'
viewBox='0 0 32 32'%3e%3cdefs%3e%3cstyle%3e.cls-1%7bfill:%233c5a99;%7d.cls-
2%7bfill:%23fff;%7d%3c/style%3e%3c/defs%3e%3ctitle%3eIcon%3c/title%3e%3crect
class='cls-1' x='1' y='1' width='28' height='28' rx='5.75'
ry='5.75'/%3e%3cpath class='cls-2'
d='M22.53,8.7h2.58V4.64H21.24a5.25,5.25,0,0,0-4,2s-1.06,1-
1.08,3.92h0v3H12.36v4.31h3.82V29h4.41V17.86h3.81.53-4.31H20.59v-
3h0A1.78,1.78,0,0,1,22.53,8.7Z'/%3e%3c/svg%3e");
    }
    .skype {
    background-image: url("data:image/svg+xml;charset=UTF-8,%3csvg
id='Layer_1' data-name='Layer 1' xmlns='http://www.w3.org/2000/svg'
viewBox='0 0 32 32'%3e%3cdefs%3e%3cstyle%3e.cls-1%7bfill:%2331c4ed;%7d.cls-
2%7bfill:%23fff;fill-
rule:evenodd;%7d%3c/style%3e%3c/defs%3e%3ctitle%3eIcon%3c/title%3e%3crect
class='cls-1' x='1' y='1' width='28' height='28' rx='5.75'
ry='5.75'/%3e%3cpath class='cls-2' d='M19.5,19.14a4,4,0,0,1-
1.75,1.31,7.22,7.22,0,0,1-
2.68,4.6A6.6,6.6,0,0,1,12,20.27,4,4,0,0,1,10.58,19,2.71,2.71,0,0,1,10,17.51a1
.05,1.05,0,0,1,.36-.8,1.28,1.28,0,0,1,.9-
.33,1.14,1.14,0,0,1,.75,2.6,1.89,1.89,0,0,1,.51,7.3,4.54,4.54,0,0,0,.49,8.6,2,2
,0,0,0,.72,5.5A3.08,3.08,0,0,0,15,19a3,3,0,0,0,1.73-.45,1.23,1.23,0,0,0,.63-
1.06,1,1,0,0,0-.33-.8,2.3,2.3,0,0,0-.92-.49c-.39-.12-.92-.26-1.57-
.39a12.2,12.2,0,0,1-2.25-.67,3.64,3.64,0,0,1-1.48-1.06,2.47,2.47,0,0,1-.29-
.47,2.84,2.84,0,0,1-.26-1.22,2.71,2.71,0,0,1,.58-1.72,3.71,3.71,0,0,1,1.67-
1.14A7.37,7.37,0,0,1,15,9.14a7,7,0,0,1,2,.26,4.38,4.38,0,0,1,1.42,7,3.1,3.1,
0,0,1,.84,9.2,2.11,2.11,0,0,1,.26,1,1.13,1.13,0,0,1-.35,8.2,1.18,1.18,0,0,1-
.88,3.6,1.09,1.09,0,0,1-.74-.23,2.68,2.68,0,0,1-.51-.67,3,3,0,0,0-.77-
.94A2.42,2.42,0,0,0,14.87,11a2.76,2.76,0,0,0-1.49,3.6,1,1,0,0,0-
.53,8.1,7.8,7.8,0,0,0,.17,5,1.63,1.63,0,0,0,.52,3.8,4.48,4.48,0,0,0,.69,2.7,1.15,0
,1,.24c.69,1.5,1.33,3.2,1.89,4.9a6.29,6.29,0,0,1,1.47,6.7,2.81,2.81,0,0,1,1,1,3,
3,0,0,1,.35,1.49,3.15,3.15,0,0,1-.6,1.89Zm4-2.23a7.73,7.73,0,0,0,.2-1.81c0-
.22,0-.43,0-.64a8.53,8.53,0,0,0-8.56-7.83,8.72,8.72,0,0,0-
1.46,12A5.08,5.08,0,0,0,11,6a5,5,0,0,0-
5,4.91,4.83,4.83,0,0,0,.68,2.48,7.33,7.33,0,0,0-
.13,9.4,6.51,6.51,0,0,0,0,.77,8.52,8.52,0,0,0,8.58,8.46,8.9,8.9,0,0,0,1.57-
.14A5.09,5.09,0,0,0,19,24a4.94,4.94,0,0,0,5-4.91,4.86,4.86,0,0,0-.52-
2.18Z'/%3e%3c/svg%3e");
    }
    .twitter {
    background: transparent url("data:image/svg+xml,%3Csvg
xmlns='http://www.w3.org/2000/svg' viewBox='0 0 32 32'%3E%3Cpath
fill='%231da1f2' d='M22.155 5.26c1.888 0 3.594.658 4.792 1.712a14.9 14.9 0 0
0 4.169-1.316c-.49 1.267-1.531 2.33-2.887 3A15.3 15.3 0 0 0 32 7.802a12.298
12.298 0 0 1-3.276 2.807c.013.233.019.467.019.703 0 7.164-6.604 15.427-
18.679 15.427-3.708 0-7.158-.897-10.064-2.436.514.05 1.037.076 1.566.076
3.076 0 5.907-.867 8.153-2.322-2.872-.043-5.297-1.612-6.132-3.766a7.864
7.864 0 0 0 2.964-.093c-3.003-.498-5.266-2.688-5.266-5.316l.001-.067a7.637
7.637 0 0 0 2.974.678c-1.762-.974-2.921-2.633-2.921-4.514 0-.994.324-

```

```

1.925.889-2.726 3.238 3.28 8.075 5.438 13.532 5.666a4.542 4.542 0 0 1-.17-
1.237c0-2.994 2.939-5.421 6.565-5.421z'/%3E%3C/svg%3E") no-repeat 100% 100%;
}
.whatsapp {
background: transparent url("data:image/svg+xml,%3Csvg
xmlns='http://www.w3.org/2000/svg' viewBox='0 0 32 32'%3E%3Cpath
fill='%2325D366' d='M11.863 8.305c-.444-.011-.813.335-1.18.677-.12.11-
.235.228-.344.35-.489.552-.798 1.186-.703 1.935.103.807.187 1.63.423
2.402.65 2.13 1.962 3.843 3.605 5.309 1.51 1.346 3.162 2.47 5.151
2.977a9.847 9.847 0 0 0 1.966.268c.984.044 2.378-.942 2.728-1.86a.37.37 0 0
0 .022-.158c-.028-.435-.052-.87-.097-1.303-.009-.088-.083-.212-.158-
.242a276.61 276.61 0 0 0-3.316-1.281c-.277-.105-.526-.057-.749.159-.337.325-
.707.617-1.029.956-.207.217-.384.257-.617.078-.619-.48-1.26-.933-1.853-
1.443-.93-.8-1.7-1.744-2.38-2.763-.19-.284-.199-.483.077-.724.294-.256.517-
.592.804-.856.366-.336.37-.69.192-1.12a85.3 85.3 0 0 1-.964-2.46c-.3-.792-
.294-.795-1.258-.833a.894.894 0 0 0-.32-.068zm4.016-5.566c5.806-.062 10.95
3.973 12.154 9.89 1.123 5.523-1.825 11.269-6.975 13.536-3.56 1.568-7.1
1.477-10.606-.216-.13-.063-.337-.05-.479.004-1.39.532-2.777 1.08-4.164
1.625-.069.027-.14.049-.266.093.412-1.455.8-2.847 1.205-4.235.064-.216.03-
.357-.112-.526-2.056-2.454-3.018-5.292-2.889-8.483.226-5.636 4.415-10.442
9.968-11.475.727-.136 1.45-.205 2.164-.213zm.203-2.086a14.416 14.416 0 0 0-
4.595.74c-5.636 1.87-9.435 6.872-9.764 12.803-.189 3.43.773 6.54 2.82
9.307.133.182.178.333.12.563-.576 2.266-1.137 4.535-1.718 6.866.178-.065.31-
.111.438-.162 2.147-.838 4.297-1.67 6.44-2.521.324-.128.598-.141.925-.005
1.091.452 2.228.737 3.4.883 3.548.44 6.831-.306 9.797-2.291 5.426-3.631
7.685-10.342 5.562-16.54A14.243 14.243 0 0 0 16.082.654zM16.032
0c.494.004.992.03 1.492.078 6.504.613 12.026 5.686 13.158 12.108.805 4.565-
.231 8.687-3.16 12.275-2.333 2.857-5.366 4.599-9.012 5.227a14.618 14.618 0 0
1-7.895-.793.818.818 0 0 0-.649 0c-2.55 1.005-5.105 1.999-7.659 2.996-
.08.031-.162.058-.305.1081.783-3.13c.393-1.573.791-3.144 1.172-4.72a.617.617
0 0 0-.089-.442c-1.92-2.71-2.94-5.725-2.8-9.04.26-6.171 3.233-10.635 8.781-
13.33C11.81.386 13.89-.017 16.032 0z'/%3E%3C/svg%3E") no-repeat 100% 100%;
}
.firefox {
background: transparent url("data:image/svg+xml,%3Csvg
xmlns='http://www.w3.org/2000/svg' viewBox='0 0 32 32'%3E%3Cpath
fill='%23e66000' d='M27.579 8.9021.048.09a.54.54 0 0 1-.045-.082zm-7.06-
4.054c-.351.003-.712.024-1.081.067 1.2.192 2.284.501 2.729 1.457-.625-.005-
1.3-.054-1.58.291 2.275 1.189 5.035 1.886 5.315 5.096-.36-.17-.48-.58-1.005-
.582.23 1.595.78 3.799.145 5.388-.35-.422-.365-1.185-.865-1.457.105 2.68-
.025 5.119-1.29 6.407-.035-.572.405-1.424 0-1.892-1.15 3.86-8.29 5.12-10.2
1.746 3.016.292 3.73-1.751 6.176-2.038-.125-.937-.915-1.676-2.01-1.748-1.19-
.076-2.215 1.096-3.305 1.02-.56-.04-1.18-.554-1.725-1.02-1.77-1.514-.894-
2.931 1.58-1.892.325-.797-.145-1.698-.43-2.33.64-.954 2.245-.928 2.3-2.475-
1.76-.012-2.745-.81-3.45-1.894.33-1.317 1.165-2.124 2.015-2.912-1.65.172-
2.605 1.05-3.734 1.748-1.075-.346-2.275.005-3.16.29-1-.539-1.15-1.941-1.58-
3.057-1.1 1.167-1.445 3.102-1.44 5.388-.765 1.018-1.57 1.998-1.575
3.786.42.168.53-.698.715-.292-2.335 8.76 5.49 16.323 13.644 16.018 8.354-
.313 13.649-8.231 13.219-17.182-.65-.12.03 1.098-.575 1.019.025-2.797-.825-
5.474-2.155-6.407.31.28.204.983.35 1.4281.032.083-.083-.157c-1.265-2.286-
3.702-3.923-6.977-3.897zM16.103 0c2.763-.007 5.09.615 6.784 1.421 4.995
2.381 9.719 7.97 9.049 16.162-.73 8.88-8.514 14.467-16.088 14.417C6.794
31.937-.645 24.6.045 14.816.32 10.992 1.724 8.529 2.634 7.1 4.599 3.998
8.959.67 14.123.111a18.887 18.887 0 0 1 1.98-.11z'/%3E%3C/svg%3E") no-repeat
100% 100%;
}
.contact {

```

```

background: transparent url("data:image/svg+xml,%3Csvg
xmlns='http://www.w3.org/2000/svg' viewBox='0 0 32 32'%3E%3Cellipse cx='16'
cy='16' fill='%234285f4' rx='16' ry='16'/%3E%3Cpath fill='%23FFF' d='M13.55
16.95h4.9c2.7 0 4.85 2.05 4.85 4.6 0 .9-.25 1.75-.75 2.45H9.45c-.5-.7-.75-
1.55-.75-2.45 0-2.55 2.15-4.6 4.85-4.6zM16.05 8c2.05 0 3.7 1.65 3.7 3.7 0
2.05-1.65 3.7-3.7 3.7-2.05 0-3.7-1.65-3.7-3.7.05-2.05 1.7-3.7 3.7-
3.7z'/%3E%3C/svg%3E") no-repeat 100% 100%;
}
.chrome {
background: transparent url("data:image/svg+xml,%3Csvg
xmlns='http://www.w3.org/2000/svg' viewBox='0 0 32 32'%3E%3Cpath
fill='%23ffffff' d='M16.033 11.049a5.155 5.155 0 1 1 0 10.312 5.156 5.156 0
0 1 0-10.312zM16.124 0c1.281-.003 9.659.318 14.268 9.043h-
.016l.01.018c.33.578 3.745 6.94-.485 14.969 0 0-4.215 8.107-14.565 7.968l-
.452-.012-.004.007-.004.007.02-.037c.564-.98 5.112-8.884 6.357-11.067l.016-
.028.007-.008.04-.069.11-.127a7.085 7.085 0 0 0 1.457-2.967l.01-.046.035-
.151c.088-.424.148-.944.128-1.549l-.006-.117v-.004l-.007-.143-.006-.07-.005-
.079-.012-.116v-.011-.001-.008-.016-.158a7.2 7.2 0 0 0-.096-.572l-.018-.081-
.013-.064a9.801 9.801 0 0 0-.692-2.016c-.165-.243-.332-.489-.512-.733l-.142-
.187 8.728-2.554s-10.538-.01-13.018-.001l.021.005H16.642l-.14-.013a7.034
7.034 0 0 0-1.132-.003l-.167.016h-.047l-.034-.001c-.193.002-1.213.045-
2.492.764l-.005.003-.033.016a7.158 7.158 0 0 0-3.25 3.533l-.059.148-6.485-
6.404s4.74 8.311 6.165 10.779l.065.113.023.088a7.14 7.14 0 0 0 7.777
5.118l.144-.02L14.854 32h-.027c-.667-.005-7.894-.234-12.744-7.906 0 0-4.925-
7.698.37-16.573l.252-.411.001-.002C2.822 6.904 6.58.374 15.958.003c0 0 .057-
.003.166-.003z'/%3E%3C/svg%3E") no-repeat 100% 100%;
}
.pinterest {
background: transparent url("data:image/svg+xml,%3Csvg
xmlns='http://www.w3.org/2000/svg' viewBox='0 0 32 32'%3E%3Cellipse cx='16'
cy='16' fill='%23bd081c' rx='16' ry='16'/%3E%3Cpath fill='%23FFF' d='M16.22
6.458c4.807-.009 9.028 1.888 9.663 5.307.787 4.256-2.438 8.866-8.213 8.535-
1.565-.09-2.222-.666-3.448-1.22-.675 2.628-1.5 5.147-3.942 6.463-.754-3.972
1.107-6.954 1.971-10.12-1.474-1.842.177-5.547 3.284-4.634 3.824 1.123-3.31
6.845 1.48 7.56 5 .746 7.04-6.441 3.94-8.779-4.48-3.376-13.042-.077-11.989
4.755.256 1.181 1.9 1.54.657 3.17-2.868-.471-3.724-2.15-3.614-4.39.178-3.664
4.435-6.229 8.705-6.583.506-.043 1.01-.064 1.507-.064z'/%3E%3C/svg%3E") no-
repeat 100% 100%;
}
.svg_icons {
width: 32px;
height: 32px;
display: inline-block;
}
}
</style>

```

{% previewsample "page.domainurl/code-snippet/badge/custom-position-cs1" %}

### Integrate badge into listview in Vue Badge component

The badges can be integrated with the `listview` component to indicate new notification with colour based on priority.

In the following sample, `default` badges are used and there is no need to customize the badge size. The component will automatically adjust the size based on the container element.

### APP.VUE

```

<template>
  <div id='element'>
    <div class="sample_container badge-list">
      <!-- Listview element -->
      <ejs-listview id='lists' :dataSource='data' :fields='fields'
showHeader='true' headerTitle='Inbox' :template='template' >
      </ejs-listview>
    </div>
  </div>
</template>
<style>
@import "../node_modules/@syncfusion/ej2-vue-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-
notifications/styles/material.css";
#element {
  display: block;
  width: 400px;
  margin: auto;
  border-radius: 3px;
  justify-content: center;
}
.sample_container.badge-list {
  max-width: 350px;
  margin: auto;
}
#lists {
  margin: auto;
  border: 1px solid rgba(0, 0, 0, 0.12)
}
#lists .e-list-item {
  cursor: pointer;
  height: 50px;
  line-height: 48px;
  border: 0;
}
#lists ul li:first-child {
  display: none;
}
/* SVG Icons and Customization */
.list_svg {
  width: 24px;
  height: 24px;
  display: inline-block;
  margin-top: 11px;
  margin-right: 16px;
}
.list_text {
  width: 60%;
  display: inline-block;
  vertical-align: top;
}
.updates {
  background: transparent url("data:image/svg+xml,%3Csvg
xmlns='http://www.w3.org/2000/svg' viewBox='0 0 32 32'%3E%3Cpath
fill='#9E9E9E' d='M17.024 13.812l.022.162v.893c0 .487-.007.942-.022 1.366-
.012.423-.027.833-.042 1.23v2.435c0 .415.022.799.064
1.15.042.35.11.559.202.622.091.063.232.095.422.095h.328l.105.324c-.041.073-

```

```
.125.136-.253.189-.127.055-.226.063-.296.028H14.126a.556.556 0 0 1-.34-
.109v-.27c.086-.108.22-.162.403-.162h.38c.185-.109.294-.219.328-.328.036-
.108.054-.3.054-.573a7.32 7.32 0 0 0 .042-.819c0-.29.007-.527.021-
.709V17.233c0-.636-.018-1.041-.053-1.213-.034-.173-.082-.386-.138-.64a20.09
20.09 0 0 1-.455.19c-.176.072-.291.108-.348.1081-.105-.162c0-.162.09-
.307.274-.4341.38-.27a6.472 6.472 0 0 1 .847-.406c.227-.144.469-.265.73-
.365.26-.1.554-.176.878-.229zm-.952-5.736a.9.9 0 0 1
.613.243c.184.162.343.361.477.595.135.235.2.451.2.65 0 .451-.129.802-.39
1.054-.261.254-.453.397-.572.434a1.424 1.424 0 0 1-.412.054.934.934 0 0 1-
.455-.122c-.15-.082-.293-.23-.433-.447a1.834 1.834 0 0 1-.277-.676c-.027-
.307.044-.63.213-.974.17-.342.373-.577.613-.703.17-.072.31-.108.423-.108zM16
3.465C9.088 3.465 3.464 9.088 3.464 16c0 6.913 5.624 12.536 12.536
12.536S28.536 22.913 28.536 16c0-6.912-5.624-12.535-12.536-12.535zM16
1.6c7.94 0 14.4 6.46 14.4 14.4S23.94 30.4 16 30.4 1.6 23.94 1.6 16 8.06 1.6
16 1.6z'/%3E%3C/svg%3E") no-repeat 100% 100%;
}
.promotion {
background: transparent url("data:image/svg+xml,%3Csvg
xmlns='http://www.w3.org/2000/svg' viewBox='0 0 32 32'%3E%3Cpath
fill='#9E9E9E' d='M6 4c.4 0 .7.2 1 .4.5.6.5 1.5-.1 2-.5.5-1.4.5-1.9-.1s-.5-
1.4.1-1.9c.2-.3.6-.4.9-.4zm0-1c-.6 0-1.2.2-1.6.6-1 .9-1.1 2.4-.2 3.4.9 1 2.4
1 3.4.2 1-.9 1-2.4.1-3.4C7.3 3.2 6.6 3 6 3zm.3-3l7.1 1 18 19.6L18.9 32 .9
12.4.6 5.3z'/%3E%3C/svg%3E") no-repeat 100% 100%;
}
.social {
background: transparent url("data:image/svg+xml,%3Csvg
xmlns='http://www.w3.org/2000/svg' viewBox='0 0 32 32'%3E%3Cpath
fill='#9E9E9E' d='M15.254 19.089c.996 0 3.587 1.992 3.786 8.271-19.03.199s-
.498-7.97 5.579-8.37c2.392-.398 4.783 1.993 6.178 1.793 1.295-.198 2.59-
1.892 3.487-1.892zm3.886-2.69c.797 0 1.793 1.495 2.989 1.694 1.096.198
3.188-1.993 5.18-1.595 5.082.3 4.684 7.573 4.684 7.573l-11.558-.1c-.697-
3.687-2.391-5.181-3.288-5.38.598-1.594 1.495-2.192 1.993-2.192zm-8.17-
9.963c2.79 0 4.98 2.49 4.98 5.679 0 3.089-2.19 5.679-4.98 5.679-2.79 0-
4.982-2.491-4.982-5.68 0-3.088 2.192-5.678 4.982-5.678zm11.657-1.994c2.49 0
4.583 2.293 4.583 5.082 0 2.79-2.092 5.082-4.583 5.082s-4.583-2.293-4.583-
5.082c0-2.79 2.092-5.082 4.583-5.082z'/%3E%3C/svg%3E") no-repeat 100% 100%;
}
.primary {
background: transparent url("data:image/svg+xml,%3Csvg
xmlns='http://www.w3.org/2000/svg' viewBox='0 0 32 32'%3E%3Cpath
fill='#9E9E9E' d='M1 7.5h10v2H2v18h28v-18h-9v-2h10a1 1 0 0 1 1 1v20a1 1 0 0
1-1 1H1a1 1 0 0 1-1-1v-20a1 1 0 0 1 1-1zm14-5h2v13.172l2.536-2.536 1.414
1.414L17 18.5l-1 1-1-1-3.95-3.95 1.414-1.414L15 15.672z'/%3E%3C/svg%3E") no-
repeat 100% 100%;
}
.draft {
background: transparent url("data:image/svg+xml,%3Csvg
xmlns='http://www.w3.org/2000/svg' viewBox='0 0 32 32'%3E%3Cpath
fill='#9E9E9E' d='M10.262 23.448l2.295 2.298-3.443 1.149zm6.886-6.895l2.296
2.298-5.739 5.746-2.295-2.298zm3.443-3.448l2.295 2.299-2.295 2.298-2.295-
2.298zM21 3.414V7h3.785zM6 2v28h20V9h-4.833C20.062 9 19 8.137 19
7.032V2zm.167-2h14.414L28 7.586V30c0 1.103-.73 2-1.833 2h-20C5.064 32 4
31.103 4 30V2C4 .897 5.064 0 6.167 0z'/%3E%3C/svg%3E") no-repeat 100% 100%;
}
.outbox {
background: transparent url("data:image/svg+xml,%3Csvg
xmlns='http://www.w3.org/2000/svg' viewBox='0 0 32 32'%3E%3Cpath
```

```

fill='#9E9E9E' d='M2 9.63V26h28.001V9.67L15.998 20.08zM2 6v1.134l14.001
10.452 14-10.408L30 6zm0-2h28c1.103 0 2 .897 2 2v20c0 1.103-.897 2-2 2H2c-
1.103 0-2-.897-2-2V6c0-1.103.897-2 2-2z'/%3E%3C/svg%3E") no-repeat 100%
100%;
    }
    .sent {
        background: transparent url("data:image/svg+xml,%3Csvg
xmlns='http://www.w3.org/2000/svg' viewBox='0 0 32 32'%3E%3Cpath
fill='#9E9E9E' d='M6.756 7.52716.997 8.487-7.02 8.516 20.126-8.457zM0
2.48132 13.603L0.24 29.52111.135-13.506z'/%3E%3C/svg%3E") no-repeat 100%
100%;
    }
    .important {
        background: transparent url("data:image/svg+xml,%3Csvg
xmlns='http://www.w3.org/2000/svg' viewBox='0 0 32 32'%3E%3Cpath
fill='#9E9E9E' d='M1.3 2.7c.7 0 1.3.6 1.3 1.3v25.2c0 .7-.6 1.3-1.3 1.3-.7 0-
1.3-.6-1.3-1.3V4c0-.7.6-1.3 1.3-1.3zm10.3-1.2C18.2 1.5 23.7 5 32 2v17.5c-
11.1 4-17.1-3.7-27.7 1V3.1c2.7-1.2 5-1.6 7.3-1.6z'/%3E%3C/svg%3E") no-repeat
100% 100%;
    }
    .starred {
        background: transparent url("data:image/svg+xml,%3Csvg
xmlns='http://www.w3.org/2000/svg' viewBox='0 0 32 32'%3E%3Cpath
fill='#9E9E9E' d='M16 1.6l4.45 9.48 9.95 1.52-7.2 7.38 1.7 10.42-8.9-4.92-
8.9 4.92 1.7-10.42-7.2-7.38 9.951-1.52z'/%3E%3C/svg%3E") no-repeat 100%
100%;
    }
</style>
<script>
import Vue from "vue";
import { ListViewPlugin } from "@syncfusion/ej2-vue-lists";
Vue.use(ListViewPlugin);
var templateVue = Vue.component("template", {
    template: `
        <div class="listWrapper" style="width: inherit;height: inherit;">
            <span :class="[data.icons + 'list_svg']">&#160;</span>
            <span class="list_text">{{data.text}}</span>
            <span :class="[data.badge]" v-if="data.messages !== ''"
style="float: right;margin-top: 16px;font-size:
12px;">{{data.messages}}</span>
        </div>
    `,
    data() {
        return {
            data: {}
        };
    }
});
export default {
    data: function() {
        return {
            data: [
                { id: 'p_01', text: 'Primary', messages: '3 New', badge: 'e-
badge e-badge-primary', icons: 'primary', type: 'Primary' },
                { id: 'p_02', text: 'Social', messages: '27 New', badge: 'e-
badge e-badge-secondary', icons: 'social', type: 'Primary' },
            ]
        };
    }
};

```

```

        { id: 'p_03', text: 'Promotions', messages: '7 New', badge: 'e-
        badge e-badge-success', icons: 'promotion', type: 'Primary' },
        { id: 'p_04', text: 'Updates', messages: '13 New', badge: 'e-
        badge e-badge-info', icons: 'updates', type: 'Primary' },
        { id: 'p_05', text: 'Starred', messages: '', badge: '', icons:
        'starred', type: 'All Labels' },
        { id: 'p_06', text: 'Important', messages: '2 New', badge: 'e-
        badge e-badge-danger', icons: 'important', type: 'All Labels' },
        { id: 'p_07', text: 'Sent', messages: '', badge: '', icons:
        'sent', type: 'All Labels' },
        { id: 'p_08', text: 'Outbox', messages: '', badge: '', icons:
        'outbox', type: 'All Labels' },
        { id: 'p_09', text: 'Drafts', messages: '7 New', badge: 'e-badge
        e-badge-warning', icons: 'draft', type: 'All Labels' },
    ],
    fields: { groupBy: 'type' },
    template: function () {
        return { template : templateVue}
    }
    };
}
}
</script>

```

```
{% previewsample "page.domainurl/code-snippet/badge/listview-cs1" %}
```

### Dynamic badge content in Vue Badge component

Badges in real-time needs to be updated dynamically based on the requirements. In this sample, using Vue data binding, the badges content will be updated dynamically. Click the increment button to change the badge value.

### APP.VUE

```

<template>
  <div id='element'>
    <div class="sample_container badge-list">
      <!-- Listview element -->
      <ejs-listview id="lists" :dataSource="data" :headerTitle="title"
      :showHeader="header" :template="listTemplate" :fields="fieldData"></ejs-
      listview>
      <p class='crossline'></p>
      <span class='incr_button'>
        <button class='e-btn e-primary' v-
        on:click="buttonClick()">Increment Badge Count</button>
      </span>
    </div>
  </div>
</template>
<script>
import Vue from "vue";
import { ListViewPlugin } from "@syncfusion/ej2-vue-lists";
Vue.use(ListViewPlugin);
var templateVue = Vue.component("template", {
  template: `
    <div class="listWrapper" style="width: inherit; height: inherit;">
      <span :class="[data.icons + 'list_svg']">&#160;</span>
    `

```

```

        <span class="list_text">{{data.text}} </span>
        <span :class="[data.badge]" v-if="data.messages !== ''"
style="float: right;margin-top: 16px;font-size: 12px;">{{data.messages}}
New</span>
    </div>
    ,
    data() {
        return {
            data: {}
        };
    }
});
export default {
    data: function() {
        return {
            data: [
                {
                    id: "p_01",
                    text: "Primary",
                    messages: 3,
                    badge: "e-badge e-badge-primary",
                    icons: "primary",
                    type: "Primary"
                },
                {
                    id: "p_02",
                    text: "Social",
                    messages: 27,
                    badge: "e-badge e-badge-secondary",
                    icons: "social",
                    type: "Primary"
                },
                {
                    id: "p_03",
                    text: "Promotions",
                    messages: 7,
                    badge: "e-badge e-badge-success",
                    icons: "promotion",
                    type: "Primary"
                },
                {
                    id: "p_04",
                    text: "Updates",
                    messages: 13,
                    badge: "e-badge e-badge-info",
                    icons: "updates",
                    type: "Primary"
                },
                {
                    id: "p_05",
                    text: "Starred",
                    messages: "",
                    badge: "",
                    icons: "starred",
                    type: "All Labels"
                },
            ]
        };
    }
};

```



```

        id: "p_06",
        text: "Important",
        messages: 2,
        badge: "e-badge e-badge-danger",
        icons: "important",
        type: "All Labels"
      },
      {
        id: "p_07",
        text: "Sent",
        messages: "",
        badge: "",
        icons: "sent",
        type: "All Labels"
      },
      {
        id: "p_08",
        text: "Outbox",
        messages: "",
        badge: "",
        icons: "outbox",
        type: "All Labels"
      },
      {
        id: "p_09",
        text: "Drafts",
        messages: 7,
        badge: "e-badge e-badge-warning",
        icons: "draft",
        type: "All Labels"
      }
    ],
    title: "Inbox",
    header: true,
    listTemplate: function() {
      return { template: templateVue };
    },
    fieldData: { groupBy: "type" }
  };
},
methods: {
  butonClick: function() {
    let badgeElements = Array.prototype.slice.call(
      document.getElementById("lists").getElementsByClassName("e-
badge")
    );
    badgeElements.forEach((element) => {
      element.textContent =
        Number(element.textContent.split(" ")[0]) + 1 + " New";
    });
  }
}
}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-layouts/styles/material.css";

```

```

@import "../node_modules/@syncfusion/ej2-vue-buttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-lists/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-notifications/styles/material.css";
#element {
    width: 400px;
    margin: auto;
    border-radius: 3px;
    justify-content: center;
}
/* ListView Customization */
#letterAvatarList {
    max-width: 350px;
    margin: auto;
    border: 1px solid #dddddd;
    border-radius: 3px;
}
#letterAvatarList .listWrapper {
    width: inherit;
    height: inherit;
    position: relative;
}
#letterAvatarList .e-list-header {
    height: 54px;
}
.material #letterAvatarList .e-list-header .e-text {
    line-height: 22px;
}
.fabric #letterAvatarList .e-list-header .e-text {
    line-height: 22px;
}
.bootstrap #letterAvatarList .e-list-header .e-text {
    line-height: 13px;
}
.highcontrast #letterAvatarList .e-list-header .e-text {
    line-height: 20px;
}
#letterAvatarList .e-list-item {
    cursor: pointer;
    height: 50px;
    line-height: 44px;
    border: 0;
}
/* Badge Positioning */
#letterAvatarList .e-badge {
    margin-top: 12px;
}
#letterAvatarList .listWrapper .list-text {
    width: 60%;
    display: inline-block;
    vertical-align: middle;
    margin: 0 50px;
}
/* Avatar Positioning */
#letterAvatarList .listWrapper .e-avatar {
    position: absolute;
    top: calc(100% - 40px);

```

```

        font-size: 10px;
        left: 5px;
    }
    /* Avatar Background Customization */
    #letterAvatarList .e-list-item:nth-child(1) .e-avatar {
        background-color: #039BE5;
    }
    #letterAvatarList .e-list-item:nth-child(3) .e-avatar {
        background-color: #E91E63;
    }
    #letterAvatarList .e-list-item:nth-child(5) .e-avatar {
        background-color: #009688;
    }
}
</style>

```

{% previewsample "page.domainurl/code-snippet/badge/dynamic-badge-cs1" %}

## Barcode

### Getting Started with the Vue Barcode Component in Vue 2

This article provides a step-by-step guide for setting up a Vue 2 project using [Vue-CLI](#) and integrating the Syncfusion Vue Barcode component using the [Composition API](#) / [Options API](#).

#### Prerequisites

##### [System requirements for Syncfusion Vue UI components](#)

#### Dependencies

The following list of dependencies are required to use the **Barcode Generator** component in your application.

```

`javascript
|-- @syncfusion/ej2-vue-barcode-generator
|-- @syncfusion/ej2-base
|-- @syncfusion/ej2-data
|-- @syncfusion/ej2-navigations
|-- @syncfusion/ej2-inputs
|-- @syncfusion/ej2-popups
|-- @syncfusion/ej2-buttons
|-- @syncfusion/ej2-lists
|-- @syncfusion/ej2-splitbuttons
|-- @syncfusion/ej2-barcode-generator
|-- @syncfusion/ej2-vue-base
,

```

### Setting up the Vue 2 project

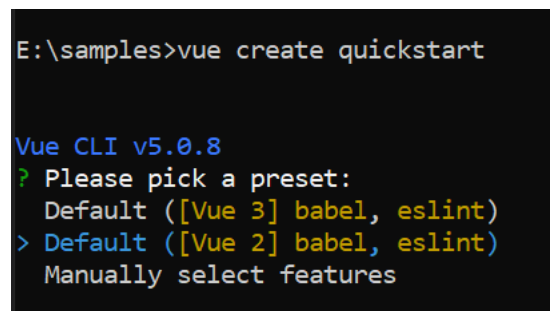
To generate a Vue 2 project using Vue-CLI, use the [vue create](#) command. Follow these steps to install Vue CLI and create a new project:

```
`bash
npm install -g @vue/cli
vue create quickstart
cd quickstart
npm run serve
`
```

or

```
`bash
yarn global add @vue/cli
vue create quickstart
cd quickstart
yarn run serve
`
```

When creating a new project, choose the option `Default ([Vue 2] babel, eslint)` from the menu.



```
E:\samples>vue create quickstart

Vue CLI v5.0.8
? Please pick a preset:
  Default ([Vue 3] babel, eslint)
> Default ([Vue 2] babel, eslint)
  Manually select features
```

Once the `quickstart` project is set up with default settings, proceed to add Syncfusion components to the project.

### Adding Syncfusion Vue packages

Syncfusion packages are available at [npmjs.com](https://www.npmjs.com). To use Vue components, install the required npm package.

This article uses the [Vue Barcode component](#) as an example. Install the `@syncfusion/ej2-vue-barcode-generator` package by running the following command:

```
`bash
npm install @syncfusion/ej2-vue-barcode-generator --save
`

or
```

```
`bash
```

```
yarn add @syncfusion/ej2-vue-barcode-generator
```

```
,
```

### Add Syncfusion Vue component

Follow the below steps to add the Vue Barcode component using **Composition API** or **Options API**:

1\ First, import and register the Barcode component in the **script** section of the **src/App.vue** file. If you are using the **Composition API**, you should add the **setup** attribute to the **script** tag to indicate that Vue will be using the **Composition API**.

#### **COMPOSITION API (~SRC/APP.VUE)**

```
<script setup>
import { QRCodeGeneratorComponent as EjsQrcodegenerator } from
 '@syncfusion/ej2-vue-barcode-generator';
</script>
```

#### **OPTIONS API (~SRC/APP.VUE)**

```
<script>
import { QRCodeGeneratorComponent } from '@syncfusion/ej2-vue-barcode-
generator';
export default {
  components: {
    'ejs-qrcodegenerator': QRCodeGeneratorComponent
  }
}
</script>
```

2\ In the **template** section, define the Barcode component with [width](#), [height](#), [value](#), [mode](#) property.

#### **~/SRC/APP.VUE**

```
<template>
<div id="app" class="barcodeStyle">
<ejs-qrcodegenerator
id="barcode"
ref="barcodeControl"
:width="width"
:height="height"
:value="value"
:mode="mode"
></ejs-qrcodegenerator>
</div>
</template>
```

3\ Declare the value for **width**, **height**, **value**, **mode** property in the **script** section

#### **COMPOSITION API (~SRC/APP.VUE)**

```
<script setup>
const width = "200px";
const height = "150px";
```

```
const type = "Codabar";
const value = "123456789";
const mode = "SVG";
</script>
```

### **OPTIONS API (~SRC/APP.VUE)**

```
<script>
data () {
  return {
    width: "200px",
    height: "150px",
    type: "Codabar",
    value: "123456789",
    mode: "SVG",
  }
}
</script>
```

Here is the summarized code for the above steps in the **src/App.vue** file:

### **COMPOSITION API (~SRC/APP.VUE)**

```
<template>
  <div id="app" class="barcodeStyle">
    <ejs-barcodegenerator
      id="barcode"
      ref="barcodeControl"
      :width="width"
      :height="height"
      :type="type"
      :value="value"
      :mode="mode"
    ></ejs-barcodegenerator>
  </div>
</template>
<style>
  .barcodeStyle {
    height: 150px;
    width: 200px;
    padding-left: 40%;
    padding-top: 9%;
  }
</style>
<script setup>
import { BarcodeGeneratorComponent as EjsBarcodegenerator } from
'@syncfusion/ej2-vue-barcode-generator';
const width = "200px";
const height = "150px";
const type = "Codabar";
const value = "123456789";
const mode = "SVG";

</script>
```

**OPTIONS API (~SRC/APP.VUE)**

```

<template>
  <div id="app" class="barcodeStyle">
    <ejs-barcodegenerator
      id="barcode"
      ref="barcodeControl"
      :width="width"
      :height="height"
      :type="type"
      :value="value"
      :mode="mode"
    ></ejs-barcodegenerator>
  </div>
</template>
<style>
  .barcodeStyle {
    height: 150px;
    width: 200px;
    padding-left: 40%;
    padding-top: 9%;
  }
</style>
<script>
import { BarcodeGeneratorComponent } from '@syncfusion/ej2-vue-barcode-generator';
export default {
  components: {
    'ejs-barcodegenerator': BarcodeGeneratorComponent
  },
  name: 'app',
  data () {
    return {
      width: "200px",
      height: "150px",
      type: "Codabar",
      value: "123456789",
      mode: "SVG",
    }
  }
}
</script>

```

```
{% previewsample "page.domainurl/code-snippet/barcode/getting-started/initialize-cs1" %}
```

**Adding QR Generator control**

You can add the QR code in our barcode generator component.

**COMPOSITION API (~SRC/APP.VUE)**

```

<template>
  <div id="app" class="barcodeStyle">
    <ejs-qrcodegenerator
      id="barcode"
      ref="barcodeControl"
      :width="width"
      :height="height"
    >

```

```

        :value="value"
        :mode="mode"
      ></ejs-qrcodegenerator>
    </div>
  </template>
<style>
  .barcodeStyle {
    height: 150px;
    width: 200px;
    padding-left: 40%;
    padding-top: 9%;
  }
</style>
<script setup>
import { QRCodeGeneratorComponent as EjsQrcodegenerator } from
 '@syncfusion/ej2-vue-barcode-generator';
const width = '200px';
const height = '150px';
const mode = 'SVG';
const value = 'Syncfusion';
</script>

```

### OPTIONS API (~SRC/APP.VUE)

```

<template>
  <div id="app" class="barcodeStyle">
    <ejs-qrcodegenerator
      id="barcode"
      ref="barcodeControl"
      :width="width"
      :height="height"
      :value="value"
      :mode="mode"
    ></ejs-qrcodegenerator>
  </div>
</template>
<style>
  .barcodeStyle {
    height: 150px;
    width: 200px;
    padding-left: 40%;
    padding-top: 9%;
  }
</style>
<script>
import { QRCodeGeneratorComponent } from '@syncfusion/ej2-vue-barcode-
generator';
export default {
  components: {
    'ejs-qrcodegenerator': QRCodeGeneratorComponent
  },
  name: 'app',
  data () {
    return {
      width: "200px",
      height: "150px",
    }
  }
}

```



```

        mode: "SVG",
        value: "Syncfusion",
      }
    }
  }
}
</script>

```

{% previewsample "page.domainurl/code-snippet/barcode/getting-started/qrcode-cs1" %}

### Adding Datamatrix Generator control

You can add the datamatrix code in our barcode generator component.

#### **COMPOSITION API (~SRC/APP.VUE)**

```

<template>
  <div id="app" class="barcodeStyle">
    <ejs-datamatrixgenerator
      id="barcode"
      ref="barcodeControl"
      :width="width"
      :height="height"
      :value="value"
      :mode="mode"
    ></ejs-datamatrixgenerator>
  </div>
</template>
<style>
  .barcodeStyle {
    height: 150px;
    width: 200px;
    padding-left: 40%;
    padding-top: 9%;
  }
</style>
<script setup>
import { DataMatrixGeneratorComponent as EjsDatamatrixgenerator } from
'@syncfusion/ej2-vue-barcode-generator';
const width = "200px";
const height = "150px";
const mode = "SVG";
const value = "Syncfusion";

</script>

```

#### **OPTIONS API (~SRC/APP.VUE)**

```

<template>
  <div id="app" class="barcodeStyle">
    <ejs-datamatrixgenerator
      id="barcode"
      ref="barcodeControl"
      :width="width"
      :height="height"
      :value="value"
      :mode="mode"
    ></ejs-datamatrixgenerator>
  </div>
</template>

```

```

    </div>
  </template>
  <style>
    .barcodeStyle {
      height: 150px;
      width: 200px;
      padding-left: 40%;
      padding-top: 9%;
    }
  </style>
  <script>
import { DataMatrixGeneratorComponent } from '@syncfusion/ej2-vue-barcode-generator';
export default {
  components: {
    'ejs-datamatrixgenerator': DataMatrixGeneratorComponent
  },
  name: 'app',
  data () {
    return {
      width: "200px",
      height: "150px",
      mode: "SVG",
      value: "Syncfusion",
    }
  }
}
  </script>

```

{% previewsample "page.domainurl/code-snippet/barcode/getting-started/datamatrix-cs1" %}

## Getting Started with Syncfusion Barcode Component in Vue 3

This section explains how to use Barcode component in Vue 3 application.

### Prerequisites

#### [System requirements for Syncfusion Vue UI components](#)

#### Creating Vue application using Vue CLI

The easiest way to create a Vue application is to use the [Vue CLI](#). Vue CLI versions higher than [4.5.0](#) are mandatory for creating applications using Vue 3. Use the following command to uninstall older versions of the Vue CLI.

```
`bash
```

```
npm uninstall vue-cli -g
```

```
,
```

Use the following commands to install the latest version of Vue CLI.

```
`bash
```

```
npm install -g @vue/cli
```

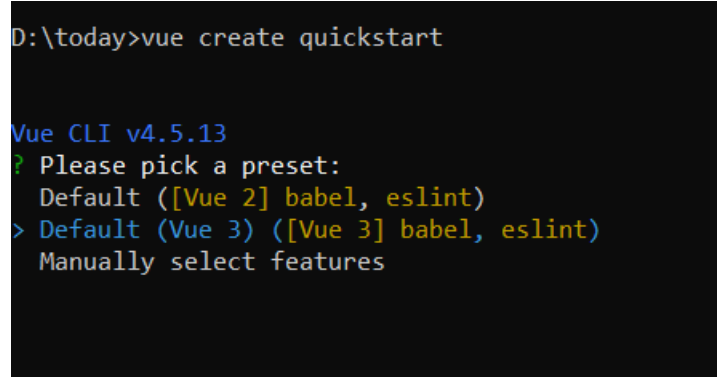
```
npm install -g @vue/cli-init
```

```
,
```

Create a new project using the following command.

```
`bash
vue create quickstart
`
```

Initiating a new project prompts us to select the type of project to be used for the current application. Select the option **Default (Vue 3)** from the menu.



```
D:\today>vue create quickstart

Vue CLI v4.5.13
? Please pick a preset:
  Default ([Vue 2] babel, eslint)
> Default (Vue 3) ([Vue 3] babel, eslint)
  Manually select features
```

#### Adding Syncfusion Barcode package in the application

Syncfusion Vue packages are maintained in the [npmjs.com](https://www.npmjs.com) registry.

In this example, the Barcode component will be used. Use the following command to install it.

```
`bash
npm install @syncfusion/ej2-vue-barcode-generator
`
```

#### Adding CSS reference for Syncfusion Vue Barcode component

Import the necessary css styles for the Barcode component along with dependency styles in the **<style>** section of the **src/App.vue** file as follows.

```
`
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-icons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-barcode-generator/styles/material.css";
</style>
`
```

#### Adding Syncfusion Vue Barcode component in the application

You have completed all of the necessary configurations required to render the Syncfusion Vue component. Now, you are going to add the Barcode component using the following steps:

1. Import the Barcode component in the `

```
,  
<script>  
import { BarcodeGeneratorComponent, QRCodeGeneratorComponent,  
DataMatrixGeneratorComponent } from '@syncfusion/ej2-vue-barcode-generator';  
</script>  
,
```

2. Register the Barcode component, which is used in this example. The following table contains a list of directives and the tag names that can be used in the Barcode component.

Directive Name	Tag Name
-----	-----
BarcodeGeneratorComponent	ejs-barcodegenerator
QRCodeGeneratorComponent	ejs-qrcodegenerator
DataMatrixGeneratorComponent	ejs-datamatrixgenerator

3. By using the following code sample, you can add the component definition in template section.

```
,  
<template>  
<div id="app" class="barcodeStyle">  
<ejs-barcodegenerator  
id="barcode"  
ref="barcodeControl"  
:width="width"  
:height="height"  
:type="type"  
:value="value"  
:mode="mode"  
:displayText="displaytext"  
  
</ejs-barcodegenerator>  
</div>  
</template>  
,
```

4. Declare the barcode generator properties in the `script` section. Here, declare the width, Height, type, value, and mode values for the `ejs-barcodegenerator` property.

```
`js
data() {
return {
width: "200px",
height: "150px",
type: "Codabar",
value: "123456789",
mode: "SVG",
displaytext: { text: 'BarcodeGenerator'}
};
}
`
```

5. Summarizing the above steps, update the `src/App.vue` file with following code.

```
`
<template>
<div id="app" class="barcodeStyle">
<ejs-barcodegenerator
id="barcode"
ref="barcodeControl"
:width="width"
:height="height"
:type="type"
:displayText="displaytext"
:value="value"
:mode="mode"

</ejs-barcodegenerator>
</div>
</template>
<script>
```

```
import {
  BarcodeGeneratorComponent,
} from "@syncfusion/ej2-vue-barcode-generator";
import "../node_modules/@syncfusion/ej2-base/styles/material.css";
import "../node_modules/@syncfusion/ej2-icons/styles/material.css";
import "../node_modules/@syncfusion/ej2-vue-barcode-generator/styles/material.css";
export default {
  name: "App",
  components: {
    "ejs-barcodegenerator": BarcodeGeneratorComponent,
  },
  data() {
    return {
      width: "200px",
      height: "150px",
      type: "Codabar",
      value: "123456789",
      mode: "SVG",
      displaytext: { text: 'BarcodeGenerator'},
    };
  },
};
</script>
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-icons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-barcode-generator/styles/material.css";
</style>
`
```

### Running the application

Run the application using the following command.

```
`bash
npm run serve
```

,

Web server will be initiated, open the quick start app in the browser at port `localhost:8080`.



#### [Adding Syncfusion Vue QR Generator control in the application](#)

You can add the QR code generator component by using the following code sample.

,

```
<template>
<div id="app" class="barcodeStyle">
<ejs-qrcodegenerator
id="qrcode"
ref="qrcodeControl"
:width="width"
:height="height"
:type="type"
:displayText="displaytext"
:value="value"
:mode="mode"

</ejs-qrcodegenerator>
</div>
</template>
<script>
import {
QRCodeGeneratorComponent,
} from "@syncfusion/ej2-vue-barcode-generator";
import "../node_modules/@syncfusion/ej2-base/styles/material.css";
import "../node_modules/@syncfusion/ej2-icons/styles/material.css";
import "../node_modules/@syncfusion/ej2-vue-barcode-generator/styles/material.css";
```

```
export default {
  name: "App",
  components: {
    "ejs-qrcodegenerator": QRCodeGeneratorComponent,
  },
  data() {
    return {
      width: "200px",
      height: "150px",
      type: "Codabar",
      value: "123456789",
      mode: "SVG",
      displaytext: { text: 'QRcodeGenerator' },
    };
  },
};
</script>
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-icons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-barcode-generator/styles/material.css";
</style>
`
```



QRcodeGenerator

[Adding Syncfusion Vue Datamatrix Generator control in the application](#)

You can add the datamatrix code generator component by using the following code sample.

`



```
<template>
<div id="app" class="barcodeStyle">
<ejs-datamatrixgenerator
id="qrcode"
ref="qrcodeControl"
:width="width"
:height="height"
:type="type"
:displayText="displaytext"
:value="value"
:mode="mode"

</ejs-datamatrixgenerator>
</div>
</template>
<script>
import {
DataMatrixGeneratorComponent,
} from "@syncfusion/ej2-vue-barcode-generator";
import "../node_modules/@syncfusion/ej2-base/styles/material.css";
import "../node_modules/@syncfusion/ej2-icons/styles/material.css";
import "../node_modules/@syncfusion/ej2-vue-barcode-generator/styles/material.css";
export default {
name: "App",
components: {
"ejs-datamatrixgenerator": DataMatrixGeneratorComponent,
},
data() {
return {
width: "200px",
height: "150px",
type: "Codabar",
value: "123456789",
```

```

mode: "SVG",
displaytext: { text: 'DataMatrix' },
};
},
};
</script>
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-icons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-barcode-generator/styles/material.css";
</style>
`

```



DataMatrix

Refer the following sample, [vue3-barcode-generator-getting-started](#).

### BarcodeGenerator in Vue Barcode component

#### Code39

The Code 39 character set includes the digits 0-9, the letters A-Z (upper case only), and the symbols: space, minus (-), plus (+), period (.), dollar sign (\$), slash (/), and percent (%). A special start / stop character is placed at the beginning and ending of each barcode. The barcode can be of any length; even more than 25 characters begin to push the bounds. Code 39 is the only type of barcode that does not require a checksum for common use.

#### APP.VUE

```

<template>
  <div id="app" class="barcodeStyle">
    <ejs-barcodegenerator
      id="barcode"
      ref="barcodeControl"
      :width="width"
      :height="height"
      :type="type"
      :value="value"
      :mode="mode"
    ></ejs-barcodegenerator>
  </div>

```

```

</template>
<style>
  .barcodeStyle {
    height: 150px;
    width: 200px;
    padding-left: 40%;
    padding-top: 9%;
  }
</style>
<script>
import Vue from 'vue';
import { BarcodeGeneratorPlugin } from '@syncfusion/ej2-vue-barcode-generator';
Vue.use(BarcodeGeneratorPlugin);
export default {
  name: 'app'
  data () {
    return {
      width: "200px",
      height: "150px",
      mode: "SVG",
      type: "Code39",
      value: "SYNCFUSION",
    }
  }
}
</script>

```

{% previewsample "page.domainurl/code-snippet/barcode/barcodegenerator/code39-cs1" %}

### Code39 Extended

Code 39 Extended is an extended version of Code 39 that supports ASCII character set. In Code 39 Extended, you can also code 26 lower letters (a-z) and the special characters in the keyboard.

### APP.VUE

```

<template>
  <div id="app" class="barcodeStyle">
    <ejs-barcodegenerator
      id="barcode"
      ref="barcodeControl"
      :width="width"
      :height="height"
      :type="type"
      :value="value"
      :mode="mode"
    ></ejs-barcodegenerator>
  </div>
</template>
<style>
  .barcodeStyle {
    height: 150px;
    width: 200px;
    padding-left: 40%;
    padding-top: 9%;
  }

```

```

</style>
<script>
import Vue from 'vue';
import { BarcodeGeneratorPlugin } from '@syncfusion/ej2-vue-barcode-generator';
Vue.use(BarcodeGeneratorPlugin);
export default {
  name: 'app'
  data () {
    return {
      width: "200px",
      height: "150px",
      mode: "SVG",
      type: "Code39Extension",
      value: "SYNCFUSION",
    }
  }
}
</script>

```

{% previewsample "page.domainurl/code-snippet/barcode/barcodegenerator/code39extd-cs1" %}

#### Code 11

Code 11 is used primarily for labeling the telecommunication equipment's. The character set includes the digits 0 to 9, a dash (-), and a start / stop code.

#### APP.VUE

```

<template>
  <div id="app" class="barcodeStyle">
    <ejs-barcodegenerator
      id="barcode"
      ref="barcodeControl"
      :width="width"
      :height="height"
      :type="type"
      :value="value"
      :mode="mode"
    ></ejs-barcodegenerator>
  </div>
</template>
<style>
  .barcodeStyle {
    height: 150px;
    width: 200px;
    padding-left: 40%;
    padding-top: 9%;
  }
</style>
<script>
import Vue from 'vue';
import { BarcodeGeneratorPlugin } from '@syncfusion/ej2-vue-barcode-generator';
Vue.use(BarcodeGeneratorPlugin);
export default {
  name: 'app'
}

```

```

    data () {
      return {
        width: "200px",
        height: "150px",
        mode: "SVG",
        type: "Code11",
        value: "112",
      }
    }
  }
</script>

```

{% previewsample "page.domainurl/code-snippet/barcode/barcodegenerator/code11-cs1" %}

### Codabar

Codabar is a variable length symbol that encodes the following 20 characters:

0123456789-\$/+.ABCD

The characters, A, B, C and D are used as start and stop characters. Codabar is used in libraries, blood banks, the package delivery industry and a variety of other information processing applications.

### APP.VUE

```

<template>
  <div id="app" class="barcodeStyle">
    <ejs-barcodegenerator
      id="barcode"
      ref="barcodeControl"
      :width="width"
      :height="height"
      :type="type"
      :value="value"
      :mode="mode"
    ></ejs-barcodegenerator>
  </div>
</template>
<style>
  .barcodeStyle {
    height: 150px;
    width: 200px;
    padding-left: 40%;
    padding-top: 9%;
  }
</style>
<script>
import Vue from 'vue';
import { BarcodeGeneratorPlugin } from '@syncfusion/ej2-vue-barcode-generator';
Vue.use(BarcodeGeneratorPlugin);
export default {
  name: 'app'
  data () {
    return {
      width: "200px",
      height: "150px",
      mode: "SVG",

```

```

        type: "Codabar",
        value: "123456789",
      }
    }
  }
</script>

```

{% previewsample "page.domainurl/code-snippet/barcode/barcodegenerator/codabar-cs1" %}

### Code 32

Code 32 is mainly used for coding pharmaceuticals, cosmetics and dietetics. It is often used to encode Italian Pharmacode that has the following structure:

- 'A' character (ASCII 65), that is not really encoded.
- 8 digits for Pharmacode (It generally begins with / and prefixed with 0).
- 1 digit for checksum module 10, that is automatically calculated by barcode.

The value to be encoded must be 8 digits Pharmacode (prefix it with '0' if necessary) and the 9th digit (the checksum) is automatically calculated by barcode.

### APP.VUE

```

<template>
  <div id="app" class="barcodeStyle">
    <ejs-barcodegenerator
      id="barcode"
      ref="barcodeControl"
      :width="width"
      :height="height"
      :type="type"
      :value="value"
      :mode="mode"
    ></ejs-barcodegenerator>
  </div>
</template>
<style>
  .barcodeStyle {
    height: 150px;
    width: 200px;
    padding-left: 40%;
    padding-top: 9%;
  }
</style>
<script>
import Vue from 'vue';
import { BarcodeGeneratorPlugin } from '@syncfusion/ej2-vue-barcode-generator';
Vue.use(BarcodeGeneratorPlugin);
export default {
  name: 'app'
  data () {
    return {
      width: "200px",
      height: "150px",
      mode: "SVG",
    }
  }
}

```

```

        type: "Code32",
        value: "01234567",
    }
}
}
</script>

```

{% previewsample "page.domainurl/code-snippet/barcode/barcodegenerator/code32-cs1" %}

### Code 93

Code 93 is designed to complement and improve upon Code 39. It can represent the entire ASCII character set by using combinations of 2 characters. Code 93 is a continuous, variable-length symbology and produces denser code. The Standard Mode (default implementation) can encode uppercase letters (A-Z), digits (0-9), and special characters like \*, -, \$, %, (Space), ., /, and +.

### APP.VUE

```

<template>
  <div id="app" class="barcodeStyle">
    <ejs-barcodegenerator
      id="barcode"
      ref="barcodeControl"
      :width="width"
      :height="height"
      :type="type"
      :value="value"
      :mode="mode"
    ></ejs-barcodegenerator>
  </div>
</template>
<style>
  .barcodeStyle {
    height: 150px;
    width: 200px;
    padding-left: 40%;
    padding-top: 9%;
  }
</style>
<script>
import Vue from 'vue';
import { BarcodeGeneratorPlugin } from '@syncfusion/ej2-vue-barcode-generator';
Vue.use(BarcodeGeneratorPlugin);
export default {
  name: 'app'
  data () {
    return {
      width: "200px",
      height: "150px",
      mode: "SVG",
      type: "Code93",
      value: "01234567",
    }
  }
}
</script>

```

```
{% previewsample "page.domainurl/code-snippet/barcode/barcodegenerator/code93-cs1" %}
```

### Code 93 Extended

The Code 93 Extended Barcode symbology is continuous, variable length, and self-checking. It is based on Code 93 Barcode. The Extended Version can encode all 128 ASCII characters.

### Code 128

Code 128 is a variable length, high density, alphanumeric, linear bar code symbology, capable of encoding the full 128-character ASCII character set and extended character sets. This symbology includes a checksum digit for verification and the barcode can also be verified character-by-character by verifying the parity of each data byte.

#### Code 128 Code Sets

- Code Set A (or Chars Set A) includes all of the standard upper case U.S. alphanumeric keyboard characters and punctuation characters along with the control characters, (namely, characters with ASCII values from 0 to 95 inclusive), and seven special characters.
- Code Set B (or Chars Set B) includes all of the standard upper case alphanumeric keyboard characters and punctuation characters along with the lower case alphabetic characters (namely, characters with ASCII values from 32 to 127 inclusive), and seven special characters.
- Code Set C (or Chars Set C) includes the set of 100 digit pairs from 00 to 99 inclusive along with three special characters. This allows numeric data to be encoded as two data digits per symbol character, at effectively twice the density of standard data.

#### Code 128 Special characters

The last seven characters of Code Sets A and B (character values 96 - 102) and the last three characters of Code Set C (character values 100 - 102) are special non-data characters with no ASCII character equivalents that have a particular significance to the Barcode reading device.

### APP.VUE

```
<template>
  <div id="app" class="barcodeStyle">
    <ejs-barcodegenerator
      id="barcode"
      ref="barcodeControl"
      :width="width"
      :height="height"
      :type="type"
      :value="value"
      :mode="mode"
    ></ejs-barcodegenerator>
  </div>
</template>
<style>
  .barcodeStyle {
    height: 150px;
    width: 200px;
    padding-left: 40%;
    padding-top: 9%;
  }
</style>
```



```

<script>
import Vue from 'vue';
import { BarcodeGeneratorPlugin } from '@syncfusion/ej2-vue-barcode-generator';
Vue.use(BarcodeGeneratorPlugin);
export default {
  name: 'app'
  data () {
    return {
      width: "200px",
      height: "150px",
      mode: "SVG",
      type: "Code128",
      value: "SYNCFUSION",
    }
  }
}
</script>

```

{% previewsample "page.domainurl/code-snippet/barcode/barcodegenerator/code128-cs1" %}

### Customizing the Barcode color

A page or printed media with barcode often appears colorful in the background and surrounding region with other contents. In such cases the barcode can also be customized to suit the needs. You can achieve this by using for forecolor property .

### APP.VUE

```

<template>
  <div id="app" class="barcodeStyle">
    <ejs-barcodegenerator
      id="barcode"
      ref="barcodeControl"
      :width="width"
      :height="height"
      :type="type"
      :foreColor="foreColor"
      :value="value"
      :mode="mode"
    ></ejs-barcodegenerator>
  </div>
</template>
<style>
  .barcodeStyle {
    height: 150px;
    width: 200px;
    padding-left: 40%;
    padding-top: 9%;
  }
</style>
<script>
import Vue from 'vue';
import { BarcodeGeneratorPlugin } from '@syncfusion/ej2-vue-barcode-generator';
Vue.use(BarcodeGeneratorPlugin);
export default {

```

```

name: 'app'
data () {
  return {
    width: "200px",
    height: "150px",
    mode: "SVG",
    type: "Code128",
    value: "SYNCFUSION",
    foreColor:"red"
  }
}
</script>

```

{% previewsample "page.domainurl/code-snippet/barcode/barcodegenerator/color-cs1" %}

### Customizing the Barcode dimension

The dimension of the barcode can be changed using the height and width property of the barcodegenerator.

### APP.VUE

```

<template>
  <div id="app" class="barcodeStyle">
    <ejs-barcodegenerator
      id="barcode"
      ref="barcodeControl"
      :width="width"
      :height="height"
      :type="type"
      :value="value"
      :mode="mode"
    ></ejs-barcodegenerator>
  </div>
</template>
<style>
  .barcodeStyle {
    height: 150px;
    width: 200px;
    padding-left: 40%;
    padding-top: 9%;
  }
</style>
<script>
import Vue from 'vue';
import { BarcodeGeneratorPlugin } from '@syncfusion/ej2-vue-barcode-generator';
Vue.use(BarcodeGeneratorPlugin);
export default {
  name: 'app'
  data () {
    return {
      width: "300px",
      height: "300px",
      mode: "SVG",
      type: "Code128",

```

```

        value: "SYNCFUSION",
    }
}
}
</script>

```

{% previewsample "page.domainurl/code-snippet/barcode/barcodegenerator/dimension-cs1" %}

### Customizing the text

In barcode generators you can customize the barcode text by using display text property .

### APP.VUE

```

<template>
  <div id="app" class="barcodeStyle" class="barcodeStyle">
    <ejs-barcodegenerator
      id="barcode"
      ref="barcodeControl"
      :width="width"
      :height="height"
      :displayText="displaytext"
      :type="type"
      :value="value"
      :mode="mode"
    ></ejs-barcodegenerator>
  </div>
</template>
<style>
  .barcodeStyle {
    height: 150px;
    width: 200px;
    padding-left: 40%;
    padding-top: 9%;
  }
</style>
<script>
import Vue from 'vue';
import { BarcodeGeneratorPlugin } from '@syncfusion/ej2-vue-barcode-generator';
Vue.use(BarcodeGeneratorPlugin);
export default {
  name: 'app'
  data () {
    return {
      width: "200px",
      height: "150px",
      displaytext: { text: 'text' },
      mode: "SVG",
      type: "Code128",
      value: "SYNCFUSION",
    }
  }
}
</script>

```

```
{% previewsample "page.domainurl/code-snippet/barcode/barcodegenerator/dimension-cs2" %}
```

## Qrcodegenerator in Vue Barcode component

### QR Code

A QR Code is a two-dimensional barcode that consists of a grid of dark and light dots or blocks that form a square. The data encoded in the barcode can be numeric, alphanumeric, or Shift Japanese Industrial Standards (JIS8) characters. The QR Code uses version from 1 to 40. Version 1 measures 21 modules x 21 modules, Version 2 measures 25 modules x 25 modules, and so on. The number of modules increases in steps of 4 modules per side up to Version 40 that measures 177 modules x 177 modules. Each version has its own capacity. By default, the barcode control automatically set the version according to the length of the input text. The QR Barcodes are designed for industrial uses and also commonly used in consumer advertising.

### APP.VUE

```
<template>
  <div id="app" class="barcodeStyle">
    <ejs-qrcodegenerator
      id="barcode"
      ref="barcodeControl"
      :width="width"
      :height="height"
      :value="value"
      :mode="mode"
    ></ejs-qrcodegenerator>
  </div>
</template>
<style>
  .barcodeStyle {
    height: 150px;
    width: 200px;
    padding-left: 40%;
    padding-top: 9%;
  }
</style>
<script>
import Vue from 'vue';
import { QRCodeGeneratorPlugin } from '@syncfusion/ej2-vue-barcode-generator';
Vue.use(QRCodeGeneratorPlugin);
export default {
  name: 'app'
  data () {
    return {
      width: "200px",
      height: "150px",
      mode: "SVG",
      value: "Syncfusion",
    }
  }
}
</script>
```

```
{% previewsample "page.domainurl/code-snippet/barcode/qrcode/qrcode-cs1" %}
```

### Customizing the Barcode color

A page or printed media with barcode often appears colorful in the background and surrounding region with other contents. In such cases the barcode can also be customized to suit the needs. You can achieve this by using for forecolor property .

#### APP.VUE

```
<template>
  <div id="app" class="barcodeStyle">
    <ejs-qrcodegenerator
      id="barcode"
      ref="barcodeControl"
      :width="width"
      :height="height"
      :foreColor="foreColor"
      :value="value"
      :mode="mode"
    ></ejs-qrcodegenerator>
  </div>
</template>
<style>
  .barcodeStyle {
    height: 150px;
    width: 200px;
    padding-left: 40%;
    padding-top: 9%;
  }
</style>
<script>
import Vue from 'vue';
import { QRCodeGeneratorPlugin } from '@syncfusion/ej2-vue-barcode-generator';
Vue.use(QRCodeGeneratorPlugin);
export default {
  name: 'app'
  data () {
    return {
      width: "200px",
      height: "150px",
      mode: "SVG",
      foreColor:"red"
      value: "Syncfusion",
    }
  }
}
</script>
```

{% previewsample "page.domainurl/code-snippet/barcode/qrcode/color-cs1" %}

### Customizing the Barcode dimension

The dimension of the barcode can be changed using the height and width properties of the barcodegenerator.

#### APP.VUE

```
<template>
```

```

<div id="app" class="barcodeStyle">
  <ejs-qrcodegenerator
    id="barcode"
    ref="barcodeControl"
    :width="width"
    :height="height"
    :value="value"
    :mode="mode"
  ></ejs-qrcodegenerator>
</div>
</template>
<style>
  .barcodeStyle {
    height: 150px;
    width: 200px;
    padding-left: 40%;
    padding-top: 9%;
  }
</style>
<script>
import Vue from 'vue';
import { QRCodeGeneratorPlugin } from '@syncfusion/ej2-vue-barcode-generator';
Vue.use(QRCodeGeneratorPlugin);
export default {
  name: 'app'
  data () {
    return {
      width: "300px",
      height: "300px",
      mode: "SVG",
      value: "Syncfusion",
    }
  }
}
</script>

```

{% previewsample "page.domainurl/code-snippet/barcode/qrcode/dimension-cs1" %}

### Customizing the text

In barcode generators You can customize the barcode text by using display text property .

### APP.VUE

```

<template>
  <div id="app" class="barcodeStyle">
    <ejs-qrcodegenerator
      id="barcode"
      ref="barcodeControl"
      :width="width"
      :height="height"
      :displayText="displaytext"
      :value="value"
      :mode="mode"
    ></ejs-qrcodegenerator>
  </div>

```

```

</template>
<style>
  .barcodeStyle {
    height: 150px;
    width: 200px;
    padding-left: 40%;
    padding-top: 9%;
  }
</style>
<script>
import Vue from 'vue';
import { QRCodeGeneratorPlugin } from '@syncfusion/ej2-vue-barcode-generator';
Vue.use(QRCodeGeneratorPlugin);
export default {
  name: 'app'
  data () {
    return {
      width: "200px",
      height: "150px",
      displaytext: { text: 'text' },
      mode: "SVG",
      value: "Syncfusion",
    }
  }
}
</script>

```

{% previewsample "page.domainurl/code-snippet/barcode/qrcode/text-cs1" %}

## Datamatrixgenerator in Vue Barcode component

### Data Matrix

DataMatrix Barcode is a two dimensional barcode that consists of a grid of dark and light dots or blocks forming square or rectangular symbol. The data encoded in the barcode can either be numbers or alphanumeric. They are widely used in printed media such as labels and letters. You can read it easily with the help of a barcode reader and mobile phones.

### APP.VUE

```

<template>
  <div id="app" class="barcodeStyle">
    <ejs-datamatrixgenerator
      id="barcode"
      ref="barcodeControl"
      :width="width"
      :height="height"
      :value="value"
      :mode="mode"
    ></ejs-datamatrixgenerator>
  </div>
</template>
<style>
  .barcodeStyle {
    height: 150px;
    width: 200px;
  }
</style>

```

```

        padding-left: 40%;
        padding-top: 9%;
    }
</style>
<script>
import Vue from 'vue';
import { DataMatrixGeneratorPlugin } from '@syncfusion/ej2-vue-barcode-generator';
Vue.use(DataMatrixGeneratorPlugin);
export default {
    name: 'app'
    data () {
        return {
            width: "200px",
            height: "150px",
            mode: "SVG",
            value: "Syncfusion",
        }
    }
}
</script>

```

{% previewsample "page.domainurl/code-snippet/barcode/datamatrix/datamatrix-cs1" %}

### Customizing the Barcode color

A page or printed media with barcode often appears colorful in the background and surrounding region with other contents. In such cases the barcode can also be customized to suit the needs. You can achieve this by using the forecolor property .

### APP.VUE

```

<template>
    <div id="app" class="barcodeStyle">
        <ejs-datamatrixgenerator
            id="barcode"
            ref="barcodeControl"
            :width="width"
            :height="height"
            :foreColor="foreColor"
            :value="value"
            :mode="mode"
        ></ejs-datamatrixgenerator>
    </div>
</template>
<style>
    .barcodeStyle {
        height: 150px;
        width: 200px;
        padding-left: 40%;
        padding-top: 9%;
    }
</style>
<script>
import Vue from 'vue';
import { DataMatrixGeneratorPlugin } from '@syncfusion/ej2-vue-barcode-generator';

```



```

Vue.use(DataMatrixGeneratorPlugin);
export default {
  name: 'app'
  data () {
    return {
      width: "200px",
      height: "150px",
      mode: "SVG",
      foreColor:"red"
      value: "Syncfusion",
    }
  }
}
</script>

```

{% previewsample "page.domainurl/code-snippet/barcode/datamatrix/color-cs1" %}

### Customizing the Barcode dimension

The dimension of the barcode can be changed using the height and width property of the barcodegenerator.

#### APP.VUE

```

<template>
  <div id="app" class="barcodeStyle">
    <ejs-datamatrixgenerator
      id="barcode"
      ref="barcodeControl"
      :width="width"
      :height="height"
      :value="value"
      :mode="mode"
    ></ejs-datamatrixgenerator>
  </div>
</template>
<style>
  .barcodeStyle {
    height: 150px;
    width: 200px;
    padding-left: 40%;
    padding-top: 9%;
  }
</style>
<script>
import Vue from 'vue';
import { DataMatrixGeneratorPlugin } from '@syncfusion/ej2-vue-barcode-generator';
Vue.use(DataMatrixGeneratorPlugin);
export default {
  name: 'app'
  data () {
    return {
      width: "300px",
      height: "300px",
      mode: "SVG",
      foreColor:"red"
    }
  }
}

```

```

        value: "Syncfusion",
      }
    }
  }
</script>

```

{% previewsample "page.domainurl/code-snippet/barcode/datamatrix/dimension-cs1" %}

### Customizing the text

In barcode generators you can customize the barcode text by using the display text property .

#### APP.VUE

```

<template>
  <div id="app" class="barcodeStyle">
    <ejs-datamatrixgenerator
      id="barcode"
      ref="barcodeControl"
      :width="width"
      :height="height"
      :displayText="displayText"
      :value="value"
      :mode="mode"
    ></ejs-datamatrixgenerator>
  </div>
</template>
<style>
  .barcodeStyle {
    height: 150px;
    width: 200px;
    padding-left: 40%;
    padding-top: 9%;
  }
</style>
<script>
import Vue from 'vue';
import { DataMatrixGeneratorPlugin } from '@syncfusion/ej2-vue-barcode-generator';
Vue.use(DataMatrixGeneratorPlugin);
export default {
  name: 'app'
  data () {
    return {
      width: "200px",
      height: "150px",
      displaytext: { text: 'text' },
      mode: "SVG",
      foreColor:"red"
      value: "Syncfusion",
    }
  }
}
</script>

```

{% previewsample "page.domainurl/code-snippet/barcode/datamatrix/text-cs1" %}

### Export in Vue Barcode component

Export barcode as an image and base64 string is common for barcode,QRcode and datamatrix. The following code samples explain how to export the barcode as an image

and base64 string.

#### Export

Barcode provides the support to export its content as an image in the specified image type and downloads it in the browser.

The following code example shows how to export the barcode as an image

```
`javascript
export default {
  name: 'app',
  data () {
    return {
      width: "200px",
      height: "150px",
      mode: "SVG",
      type: "Code39",
      value: "SYNCFUSION",
    }
  }
  mounted: function() {
    let barcodeInstance;
    let barcodeObj = document.getElementById("barcode");
    barcodeInstance = barcodeObj.ej2_instances[0];
    let filename = 'Export';
    barcodeInstance.exportImage(filename,'JPG');
  }
}
```

The filename specifies the name of the file to be downloaded.

#### Export As Base64String

Barcode provides the support to export its content as an image in the specified image type and returns it as base64 string.

The following code example shows how to export the barcode as a base64 string

```
`javascript
```

```
export default {
  name: 'app',
  data () {
    return {
      width: "200px",
      height: "150px",
      mode: "SVG",
      type: "Code39",
      value: "SYNCFUSION",
      image: function(event) {
        let barcodeInstance;
        let barcodeObj = document.getElementById("barcode");
        barcodeInstance = barcodeObj.ej2_instances[0];
        setTimeout(() => {
          base64string();
        }, 100);
        console.log(barcodeInstance.exportAsBase64Image('JPG'))
      }
    }
  },
  async function base64string(){
    let barcodeInstance;
    let barcodeObj = document.getElementById("barcode");
    barcodeInstance = barcodeObj.ej2_instances[0];
    let base64 = await barcodeInstance.exportAsBase64Image('JPG');
    console.log(base64);
  },
  ,
}
```

**Note:**

Format is to specify the type or format of the exported file. You can export the barcode to the following formats:

\* JPG.

\* PNG.

## Breadcrumb

### Getting Started with the Vue Breadcrumb Component in Vue 2

This article provides a step-by-step guide for setting up a Vue 2 project using [Vue-CLI](#) and integrating the Syncfusion Vue Breadcrumb component

#### Prerequisites

[System requirements for Syncfusion Vue UI components](#)

#### Dependencies

The following list of dependencies are required to use the Breadcrumb component in your application.

```
`js
|-- @syncfusion/ej2-vue-navigations
|-- @syncfusion/ej2-vue-base
|-- @syncfusion/ej2-navigations
|-- @syncfusion/ej2-base
|-- @syncfusion/ej2-data
|-- @syncfusion/ej2-lists
|-- @syncfusion/ej2-inputs
|-- @syncfusion/ej2-popups
|-- @syncfusion/ej2-buttons
`
```

#### Setting up the Vue 2 project

To generate a Vue 2 project using Vue-CLI, use the [vue create](#) command. Follow these steps to install Vue CLI and create a new project:

```
`bash
npm install -g @vue/cli
vue create quickstart
cd quickstart
npm run serve
`
```

or

```
`bash
yarn global add @vue/cli
vue create quickstart
cd quickstart
```

```
yarn run serve
```

When creating a new project, choose the option `Default ([Vue 2] babel, eslint)` from the menu.

```
E:\samples>vue create quickstart

Vue CLI v5.0.8
? Please pick a preset:
  Default ([Vue 3] babel, eslint)
> Default ([Vue 2] babel, eslint)
  Manually select features
```

Once the `quickstart` project is set up with default settings, proceed to add Syncfusion components to the project.

### Add Syncfusion Vue packages

Syncfusion packages are available at [npmjs.com](https://www.npmjs.com). To use Vue components, install the required npm package.

This article uses the [Vue Breadcrumb component](#) as an example. Install the `@syncfusion/ej2-vue-navigations` package by running the following command:

```
`bash
```

```
npm install @syncfusion/ej2-vue-navigations --save
```

or

```
`bash
```

```
yarn add @syncfusion/ej2-vue-navigations
```

### Add Syncfusion Vue component

Follow the below steps to add the Vue Breadcrumb component:

1\. First, import and register the Breadcrumb component in the `script` section of the `src/App.vue` file.

#### ~/SRC/APP.VUE

```
<script>
import { BreadcrumbComponent } from "@syncfusion/ej2-vue-navigations";
export default {
  components:{
    'ejs-breadcrumb': BreadcrumbComponent
  }
}
</script>
```

2\. In the `template` section, define the Breadcrumb component with the [enableNavigation](#) property.

~/SRC/APP.VUE

```
<template>
<div>
<ejs-breadcrumb :enableNavigation='false'></ejs-breadcrumb>
</div>
</template>
```

Here is the summarized code for the above steps in the **src/App.vue** file:

~/SRC/APP.VUE

```
<template>
<div>
<ejs-breadcrumb :enableNavigation='false'></ejs-breadcrumb>
</div>
</template>
<script>
import { BreadcrumbComponent } from "@syncfusion/ej2-vue-navigations";
export default {
  components:{
    'ejs-breadcrumb': BreadcrumbComponent
  },
  data: function() {
    return {};
  }
}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-navigations/styles/material.css";
body {
  margin-top: 100px;
  text-align: center;
}
</style>
```

### Run the project

To run the project, use the following command:

```
`bash
```

```
npm run serve
```

```
,
```

or

```
`bash
```

```
yarn run serve
```

```
,
```

```
{% previewsample "page.domainurl/code-snippet/breadcrumb/getting-started-cs5" %}
```

### Add Items to the Breadcrumb Component

Use `items` property to bind items for Breadcrumb component. The below example demonstrates the basic rendering of Breadcrumb with items support.

#### ~/SRC/APP.VUE

```
<template>
<div>
<ejs-breadcrumb :enableNavigation='false'>
  <e-breadcrumb-items>
    <e-breadcrumb-item iconCss= 'e-icons e-home' url=
'https://ej2.syncfusion.com/vue/demos/'></e-breadcrumb-item>
    <e-breadcrumb-item text= 'Components' url=
'https://ej2.syncfusion.com/vue/demos/datagrid/overview'></e-breadcrumb-
item>
    <e-breadcrumb-item text= 'Navigations' url=
'https://ej2.syncfusion.com/vue/demos/menu/default'></e-breadcrumb-item>
    <e-breadcrumb-item text= 'Breadcrumb' url=
'https://ej2.syncfusion.com/vue/demos/breadcrumb/default'></e-breadcrumb-
item>
  </e-breadcrumb-items>
</ejs-breadcrumb>
</div>
</template>
<script>
import { BreadcrumbComponent, BreadcrumbItemDirective,
BreadcrumbItemsDirective } from "@syncfusion/ej2-vue-navigations";
export default {
  components: {
    'ejs-breadcrumb': BreadcrumbComponent,
    'e-breadcrumb-item': BreadcrumbItemDirective,
    'e-breadcrumb-items': BreadcrumbItemsDirective
  },
  data: function() {
    return {};
  }
}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-navigations/styles/material.css";
body {
  margin-top: 100px;
  text-align: center;
}
</style>
```

{% previewsample "page.domainurl/code-snippet/breadcrumb/getting-started-cs6" %}

### Enable or Disable Navigation

This feature enables or disables the item navigation. By default, the navigation will be enabled when setting `Url` property. To prevent breadcrumb item navigation, set `enableNavigation` property as false in Breadcrumb. The below example shows enabling and disabling the navigation of Breadcrumb items.

#### ~/SRC/APP.VUE



```

<template>
<div>
  <div id="breadcrumb-control">
    <div class="header"><b>Enable Navigation- false</b></div><br />
    <ejs-breadcrumb :enableNavigation='false'>
      <e-breadcrumb-items>
        <e-breadcrumb-item iconCss= 'e-icons e-home' url=
'https://ej2.syncfusion.com/vue/demos/'></e-breadcrumb-item>
        <e-breadcrumb-item text= 'Components' url=
'https://ej2.syncfusion.com/vue/demos/datagrid/overview'></e-breadcrumb-
item>
        <e-breadcrumb-item text= 'Navigations' url=
'https://ej2.syncfusion.com/vue/demos/menu/default'></e-breadcrumb-item>
        <e-breadcrumb-item text= 'Breadcrumb' url=
'https://ej2.syncfusion.com/vue/demos/breadcrumb/default'></e-breadcrumb-
item>
      </e-breadcrumb-items>
    </ejs-breadcrumb>
    <br />
    <br/>
    <div class="header"><b>Enable Navigation- true</b></div><br />
    <ejs-breadcrumb :enableNavigation='true'>
      <e-breadcrumb-items>
        <e-breadcrumb-item iconCss= 'e-icons e-home' url=
'https://ej2.syncfusion.com/vue/demos/'></e-breadcrumb-item>
        <e-breadcrumb-item text= 'Components' url=
'https://ej2.syncfusion.com/vue/demos/datagrid/overview'></e-breadcrumb-
item>
        <e-breadcrumb-item text= 'Navigations' url=
'https://ej2.syncfusion.com/vue/demos/menu/default'></e-breadcrumb-item>
        <e-breadcrumb-item text= 'Breadcrumb' url=
'https://ej2.syncfusion.com/vue/demos/breadcrumb/default'></e-breadcrumb-
item>
      </e-breadcrumb-items>
    </ejs-breadcrumb>
  </div>
</div>
</template>
<script>
import { BreadcrumbComponent, BreadcrumbItemDirective,
BreadcrumbItemsDirective } from "@syncfusion/ej2-vue-navigations";
export default {
  components: {
    'ejs-breadcrumb': BreadcrumbComponent,
    'e-breadcrumb-item': BreadcrumbItemDirective,
    'e-breadcrumb-items': BreadcrumbItemsDirective
  },
  data: function() {
    return {};
  }
}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-navigations/styles/material.css";
body {
  margin-top: 100px;

```

```
    text-align: center;
  }
  #breadcrumb-control {
    margin-left: auto;
    margin-right: auto;
    width: 60%;
  }
  #breadcrumb-control .header {
    text-align: left;
    padding-left: 10px;
  }
  #breadcrumb-control .e-control.e-breadcrumb {
    text-align: left;
  }
</style>
```

{% previewsample "page.domainurl/code-snippet/breadcrumb/getting-started-cs7" %}

## Getting Started with the Vue Breadcrumb Component in Vue 3

This article provides a step-by-step guide for setting up a [Vite](#) project with a JavaScript environment and integrating the Syncfusion Vue Breadcrumb component using the [Composition API](#) / [Options API](#).

The **Composition API** is a new feature introduced in Vue.js 3 that provides an alternative way to organize and reuse component logic. It allows developers to write components as functions that use smaller, reusable functions called composition functions to manage their properties and behavior.

The **Options API** is the traditional way of writing Vue.js components, where the component logic is organized into a series of options that define the component's properties and behavior. These options include data, methods, computed properties, watchers, lifecycle hooks, and more.

### Prerequisites

#### [System requirements for Syncfusion Vue UI components](#)

#### Set up the Vite project

A recommended approach for beginning with Vue is to scaffold a project using [Vite](#). To create a new Vite project, use one of the commands that are specific to either NPM or Yarn.

```
`bash
```

```
npm create vite@latest
```

```
`
```

or

```
`bash
```

```
yarn create vite
```

```
`
```

Using one of the above commands will lead you to set up additional configurations for the project as below:

1. Define the project name: We can specify the name of the project directly. Let's specify the name of the project as **my-project** for this article.

```
`bash
```

```
? Project name: » my-project
```

```
,
```

2. Select **Vue** as the framework. It will create a Vue 3 project.

```
`bash
```

```
? Select a framework: » - Use arrow-keys. Return to submit.
```

```
Vanilla
```

```
Vue
```

```
React
```

```
Preact
```

```
Lit
```

```
Svelte
```

```
Others
```

```
,
```

3. Choose **JavaScript** as the framework variant to build this Vite project using JavaScript and Vue.

```
`bash
```

```
? Select a variant: » - Use arrow-keys. Return to submit.
```

```
JavaScript
```

```
TypeScript
```

```
Customize with create-vue ↗
```

```
Nuxt ↗
```

```
,
```

4. Upon completing the aforementioned steps to create the **my-project**, run the following command to install its dependencies:

```
`bash
```

```
cd my-project
```

```
npm install
```

```
,
```

```
or
```

```
`bash
```

```
cd my-project
```

```
yarn install
```

```
,
```

Now that `my-project` is ready to run with default settings, let's add Syncfusion components to the project.

### Add Syncfusion Vue packages

Syncfusion Vue component packages are available at [npmjs.com](https://www.npmjs.com). To use Syncfusion Vue components in the project, install the corresponding npm package.

This article uses the [Vue Breadcrumb component](#) as an example. To use the Vue Breadcrumb component in the project, the `@syncfusion/ej2-vue-navigations` package needs to be installed using the following command:

```
`bash
```

```
npm install @syncfusion/ej2-vue-navigations --save
```

```
`
```

or

```
`bash
```

```
yarn add @syncfusion/ej2-vue-navigations
```

```
`
```

### Import Syncfusion CSS styles

You can import themes for the Syncfusion Vue component in various ways, such as using CSS or SASS styles from npm packages, CDN, [CRG](#) and [Theme Studio](#). Refer to [themes topic](#) to know more about built-in themes and different ways to refer to themes in a Vue project.

In this article, `Material` theme is applied using CSS styles, which are available in installed packages. The necessary `Material` CSS styles for the Breadcrumb component and its dependents were imported into the `<style>` section of `src/App.vue` file.

#### ~/SRC/APP.VUE

```
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-navigations/styles/material.css";
</style>
```

The order of importing CSS styles should be in line with its dependency graph.

### Add Syncfusion Vue component

Follow the below steps to add the Vue Breadcrumb component using `Composition API` or `Options API`:

1.First, import and register the Breadcrumb component and its child directives in the `script` section of the `src/App.vue` file. If you are using the `Composition API`, you should add the `setup` attribute to the `script` tag to indicate that Vue will be using the `Composition API`.

#### COMPOSITION API (~/SRC/APP.VUE)

```
<script setup>
import { BreadcrumbComponent as EjsBreadcrumb } from "@syncfusion/ej2-vue-navigations";
</script>
```

**OPTIONS API (~SRC/APP.VUE)**

```
<script>
import { BreadcrumbComponent } from "@syncfusion/ej2-vue-navigations";
//Component registration
export default {
  name: "App",
  components: {
    'ejs-breadcrumb': BreadcrumbComponent
  }
}
</script>
```

2. In the **template** section, define the Breadcrumb component with the [enableRtl](#) property and item definitions.

**~/SRC/APP.VUE**

```
<template>
<div>
<ejs-breadcrumb :enableRtl='enableRtl'>
<e-breadcrumb-items>
<e-breadcrumb-item iconCss= 'e-icons e-home' url=
'https://ej2.syncfusion.com/vue/demos/'></e-breadcrumb-item>
<e-breadcrumb-item text= 'Components' url=
'https://ej2.syncfusion.com/vue/demos/datagrid/overview'></e-breadcrumb-
item>
<e-breadcrumb-item text= 'Navigations' url=
'https://ej2.syncfusion.com/vue/demos/menu/default'></e-breadcrumb-item>
<e-breadcrumb-item text= 'Breadcrumb' url=
'https://ej2.syncfusion.com/vue/demos/breadcrumb/default'></e-breadcrumb-
item>
</e-breadcrumb-items>
</ejs-breadcrumb>
</div>
</template>
```

3. Declare the values for the **enableNavigation** property in the **script** section.

**COMPOSITION API (~SRC/APP.VUE)**

```
<script setup>
const enableRtl = 'true';
</script>
```

**OPTIONS API (~SRC/APP.VUE)**

```
<script>
data() {
  return {
    enableRtl: 'true',
  };
}
</script>
```

Here is the summarized code for the above steps in the **src/App.vue** file:

### **COMPOSITION API (~SRC/APP.VUE)**

```
<template>
<div>
<ejs-breadcrumb :enableRtl='enableRtl'>
<e-breadcrumb-items>
<e-breadcrumb-item iconCss= 'e-icons e-home' url=
'https://ej2.syncfusion.com/vue/demos/'></e-breadcrumb-item>
<e-breadcrumb-item text= 'Components' url=
'https://ej2.syncfusion.com/vue/demos/datagrid/overview'></e-breadcrumb-
item>
<e-breadcrumb-item text= 'Navigations' url=
'https://ej2.syncfusion.com/vue/demos/menu/default'></e-breadcrumb-item>
<e-breadcrumb-item text= 'Breadcrumb' url=
'https://ej2.syncfusion.com/vue/demos/breadcrumb/default'></e-breadcrumb-
item>
</e-breadcrumb-items>
</ejs-breadcrumb>
</div>
</template>
<script setup>
import { BreadcrumbComponent as EjsBreadcrumb, ItemsDirective as
EBreadcrumbItems, ItemDirective as EBreadcrumbItem } from "@syncfusion/ej2-
vue-navigations";
const enableRtl='true'
</script>
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-navigations/styles/material.css";
</style>
```

### **OPTIONS API (~SRC/APP.VUE)**

```
<template>
<div>
<ejs-breadcrumb :enableRtl='enableRtl'>
<e-breadcrumb-items>
<e-breadcrumb-item iconCss= 'e-icons e-home' url=
'https://ej2.syncfusion.com/vue/demos/'></e-breadcrumb-item>
<e-breadcrumb-item text= 'Components' url=
'https://ej2.syncfusion.com/vue/demos/datagrid/overview'></e-breadcrumb-
item>
<e-breadcrumb-item text= 'Navigations' url=
'https://ej2.syncfusion.com/vue/demos/menu/default'></e-breadcrumb-item>
<e-breadcrumb-item text= 'Breadcrumb' url=
'https://ej2.syncfusion.com/vue/demos/breadcrumb/default'></e-breadcrumb-
item>
</e-breadcrumb-items>
</ejs-breadcrumb>
</div>
</template>
<script>
import { BreadcrumbComponent, ItemsDirective, ItemDirective } from
"@syncfusion/ej2-vue-navigations";
// Component registration
```

```
export default {
  name: "App",
  // Declaring component and its directives
  components: {
    'ejs-breadcrumb': BreadcrumbComponent,
    'e-breadcrumb-items': ItemsDirective,
    'e-breadcrumb-item': ItemDirective
  },
  // Bound properties declarations
  data() {
    return {
      enableRtl: 'true',
    };
  }
};
</script>
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-navigations/styles/material.css";
</style>
```

### Run the project

To run the project, use the following command:

```
`bash
```

```
npm run dev
```

```
,
```

or

```
`bash
```

```
yarn run dev
```

```
,
```

The output will appear as follows:

**Breadcrumb** / Navigations / Components / 

For migrating from Vue 2 to Vue 3, refer to the [migration](#) documentation.

See also

- [Getting Started with Vue UI Components using Composition API and TypeScript](#)
- [Getting Started with Vue UI Components using Options API and TypeScript](#)

## Data binding in Vue Breadcrumb component

### Items as tag directive

The React Breadcrumb contains `<e-breadcrumb-items>` and `<e-breadcrumb-item>` tags to render items for the component. To bind items, use `<e-breadcrumb-items>` tag and `<e-breadcrumb-item>` tag to bind properties for breadcrumb items.

#### APP.VUE

```
<template>
<div>
<ejs-breadcrumb :enableNavigation='false'>
  <e-breadcrumb-items>
    <e-breadcrumb-item iconCss= 'e-icons e-home' url=
'https://ej2.syncfusion.com/vue/demos/'></e-breadcrumb-item>
    <e-breadcrumb-item text= 'Components' url=
'https://ej2.syncfusion.com/vue/demos/datagrid/overview'></e-breadcrumb-
item>
    <e-breadcrumb-item text= 'Navigations' url=
'https://ej2.syncfusion.com/vue/demos/menu/default'></e-breadcrumb-item>
    <e-breadcrumb-item text= 'Breadcrumb' url=
'https://ej2.syncfusion.com/vue/demos/breadcrumb/default'></e-breadcrumb-
item>
  </e-breadcrumb-items>
</ejs-breadcrumb>
</div>
</template>
<script>
import Vue from 'vue';
import { BreadcrumbPlugin } from "@syncfusion/ej2-vue-navigations";
Vue.use(BreadcrumbPlugin);
export default {
  data: function() {
    return {};
  }
}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-navigations/styles/material.css";
body {
  margin-top: 100px;
  text-align: center;
}
</style>
```

{% previewsample "page.domainurl/code-snippet/breadcrumb/getting-started-cs1" %}

### Items based on current URL

The breadcrumb items can be generated from the current URL of the page, if the [url](#) property is not provided or when the user does not specify the breadcrumb items using the `<e-breadcrumb-item>` tag. The following example shows the breadcrumb items that are generated based on the current URL.

#### APP.VUE

```
<template>
```



```

<div>
<ejs-breadcrumb :enableNavigation='false'></ejs-breadcrumb>
</div>
</template>
<script>
import Vue from 'vue';
import { BreadcrumbPlugin } from "@syncfusion/ej2-vue-navigations";
Vue.use(BreadcrumbPlugin);
export default {
  data: function() {
    return {};
  }
}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-navigations/styles/material.css";
body {
  margin-top: 100px;
  text-align: center;
}
</style>

```

{% previewsample "page.domainurl/code-snippet/breadcrumb/getting-started-cs2" %}

This sample is hosted in different location, so the Breadcrumb Component is rendered with different location instead of the actual location.

### Static URL

The breadcrumb items can be generated by providing the [url](#) property in the component, if the user does not specify the breadcrumb items using the `<e-breadcrumb-item>` tag. The following example shows the breadcrumb items generated from the provided url in the component.

### APP.VUE

```

<template>
<div>
<ejs-breadcrumb :enableNavigation='false'
url="https://ej2.syncfusion.com/demos/breadcrumb/bind-to-location"></ejs-
breadcrumb>
</div>
</template>
<script>
import Vue from 'vue';
import { BreadcrumbPlugin } from "@syncfusion/ej2-vue-navigations";
Vue.use(BreadcrumbPlugin);
export default {
  data: function() {
    return {};
  }
}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-navigations/styles/material.css";
body {

```

```
margin-top: 100px;
text-align: center;
}
</style>
```

{% previewsample "page.domainurl/code-snippet/breadcrumb/getting-started-cs3" %}

### Customize text when generated items using Url

The breadcrumb items text can be customized by using the [beforeItemRender](#) event. In the following example, **bind-to-location** text was customized as **location**.

#### APP.VUE

```
<template>
<div>
<ejs-breadcrumb :enableNavigation='false'
url="https://ej2.syncfusion.com/demos/breadcrumb/bind-to-location"
:beforeItemRender="beforeItemRenderHandler">
</ejs-breadcrumb>
</div>
</template>
<script>
import Vue from 'vue';
import { BreadcrumbPlugin } from "@syncfusion/ej2-vue-navigations";
Vue.use(BreadcrumbPlugin);
export default {
  data: function() {
    return {};
  },
  methods: {
    beforeItemRenderHandler: function(args) {
      if (args.item.text === 'bind-to-location') {
        args.item.text = 'location';
      }
    }
  }
}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-navigations/styles/material.css";
body {
  margin-top: 100px;
  text-align: center;
}
</style>
```

{% previewsample "page.domainurl/code-snippet/breadcrumb/getting-started-cs4" %}

### Icons in Vue Breadcrumb component

#### Loading Icons in Breadcrumb Item

To load the icon on the breadcrumb item, set the [iconCss](#) property.

*Breadcrumb with font icons*

To place the font icon on the breadcrumb item, set the `iconCss` property to `e-icons` with the required icon CSS. By default, the icon is positioned to the left side of the item.

**APP.VUE**

```
<template>
<div>
<ejs-breadcrumb :enableNavigation='false'>
  <e-breadcrumb-items>
    <e-breadcrumb-item iconCss= 'e-icons e-home' url=
'https://ej2.syncfusion.com/vue/demos/'></e-breadcrumb-item>
    <e-breadcrumb-item text= 'Components' url=
'https://ej2.syncfusion.com/vue/demos/datagrid/overview'></e-breadcrumb-
item>
    <e-breadcrumb-item text= 'Navigations' url=
'https://ej2.syncfusion.com/vue/demos/menu/default'></e-breadcrumb-item>
    <e-breadcrumb-item text= 'Breadcrumb' url=
'https://ej2.syncfusion.com/vue/demos/breadcrumb/default'></e-breadcrumb-
item>
  </e-breadcrumb-items>
</ejs-breadcrumb>
</div>
</template>
<script>
import Vue from 'vue';
import { BreadcrumbPlugin } from "@syncfusion/ej2-vue-navigations";
Vue.use(BreadcrumbPlugin);
export default {
  data: function() {
    return {};
  }
}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-navigations/styles/material.css";
body {
  margin-top: 100px;
  text-align: center;
}
</style>
```

{% previewsample "page.domainurl/code-snippet/breadcrumb/getting-started-cs8" %}

*Breadcrumb with Image*

In the Breadcrumb component, images can be added for the items using the `iconCss` property. In the following example, an image was added to the breadcrumb item by using the iconCss class `e-image-home` and specifying height and width.

**APP.VUE**

```
<template>
<div>
<ejs-breadcrumb :enableNavigation='false'>
  <e-breadcrumb-items>
```

```

    <e-breadcrumb-item iconCss= 'e-image-home' url=
'https://ej2.syncfusion.com/vue/demos/'></e-breadcrumb-item>
    <e-breadcrumb-item text= 'Components' url=
'https://ej2.syncfusion.com/vue/demos/datagrid/overview'></e-breadcrumb-
item>
    <e-breadcrumb-item text= 'Navigations' url=
'https://ej2.syncfusion.com/vue/demos/menu/default'></e-breadcrumb-item>
    <e-breadcrumb-item text= 'Breadcrumb' url=
'https://ej2.syncfusion.com/vue/demos/breadcrumb/default'></e-breadcrumb-
item>
  </e-breadcrumb-items>
</ejs-breadcrumb>
</div>
</template>
<script>
import Vue from 'vue';
import { BreadcrumbPlugin } from "@syncfusion/ej2-vue-navigations";
Vue.use(BreadcrumbPlugin);
export default {
  data: function() {
    return {};
  }
}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-navigations/styles/material.css";
body {
  margin-top: 100px;
  text-align: center;
}
.e-image-home {
  background-image: url('home.png');
  height: 20px;
  width: 20px;
}
</style>

```

{% previewsample "page.domainurl/code-snippet/breadcrumb/getting-started-cs9" %}

### Breadcrumb with SVG Image

In the Breadcrumb component, SVG image can be added for the items using the [iconCss](#) property. In the following example, SVG image was added to the breadcrumb item by using the iconCss class `e-svg-home` and specifying height and width.

### APP.VUE

```

<template>
<div>
<ejs-breadcrumb :enableNavigation='false'>
  <e-breadcrumb-items>
    <e-breadcrumb-item iconCss= 'e-svg-home' url=
'https://ej2.syncfusion.com/vue/demos/'></e-breadcrumb-item>
    <e-breadcrumb-item text= 'Components' url=
'https://ej2.syncfusion.com/vue/demos/datagrid/overview'></e-breadcrumb-
item>

```

```

    <e-breadcrumb-item text= 'Navigations' url=
'https://ej2.syncfusion.com/vue/demos/menu/default'></e-breadcrumb-item>
    <e-breadcrumb-item text= 'Breadcrumb' url=
'https://ej2.syncfusion.com/vue/demos/breadcrumb/default'></e-breadcrumb-
item>
  </e-breadcrumb-items>
</ejs-breadcrumb>
</div>
</template>
<script>
import Vue from 'vue';
import { BreadcrumbPlugin } from "@syncfusion/ej2-vue-navigations";
Vue.use(BreadcrumbPlugin);
export default {
  data: function() {
    return {};
  }
}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-navigations/styles/material.css";
body {
  margin-top: 100px;
  text-align: center;
}
.e-svg-home {
  background-image: url('home.svg');
  height: 20px;
  width: 20px;
}
</style>

```

{% previewsample "page.domainurl/code-snippet/breadcrumb/getting-started-cs10" %}

### Icon Position

By default, the icon is positioned to the left side of the item in the Breadcrumb component. If you need to add the icon right to the breadcrumb item, add the `e-icon-right` class to the required item. In the following example, the `e-icon-right` class was added to the breadcrumb items using the [beforeItemRender](#) event.

### APP.VUE

```

<template>
<div id="breadcrumb-control">
<div class='header'><b>IconPosition- left</b></div><br />
<ejs-breadcrumb>
  <e-breadcrumb-items>
    <e-breadcrumb-item text="Program Files" iconCss="e-bicons e-folder"></e-
breadcrumb-item>
    <e-breadcrumb-item text="Services" iconCss="e-bicons e-folder"></e-
breadcrumb-item>
    <e-breadcrumb-item text="Config.json" iconCss="e-bicons e-file"></e-
breadcrumb-item>
  </e-breadcrumb-items>
</ejs-breadcrumb>

```

```

<br />
<br/>
<div class='header'><b>IconPosition- Right</b></div><br />
<ejs-breadcrumb :beforeItemRender="beforeItemRenderHandler">
  <e-breadcrumb-items >
    <e-breadcrumb-item text="Program Files" iconCss="e-bicons e-folder"></e-
breadcrumb-item>
    <e-breadcrumb-item text="Services" iconCss="e-bicons e-folder"></e-
breadcrumb-item>
    <e-breadcrumb-item text="Config.json" iconCss="e-bicons e-file"></e-
breadcrumb-item>
  </e-breadcrumb-items>
</ejs-breadcrumb>
</div>
</template>
<script>
import Vue from 'vue';
import { BreadcrumbPlugin } from "@syncfusion/ej2-vue-navigations";
Vue.use(BreadcrumbPlugin);
export default {
  data: function() {
    return {};
  },
  methods: {
    beforeItemRenderHandler: function(args) {
      if (args.element) {
        args.element.classList.add('e-icon-right');
      }
    }
  }
}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-navigations/styles/material.css";
body {
  margin-top: 100px;
  text-align: center;
}
@font-face {
  font-family: 'e-bicons';
  src:
    url(data:application/x-font-ttf;charset=utf-
8;base64,AAEAAAAKAIAAAwAgTlMvMj1wSfkAAAEoAAAAVmNtYXNHOdpAAABmAAAAD5nbHlmSRv
kRAAAAEgAAANoaGVhZB2Xb78AADQAAANmhoZWEIUQQHAAARAAAACRobXR4GAAAAAAAYAAAAA
YbG9jYQSCAv4AAAHYAAADm1heHABFwEfAAABCAAAACBuYW11Xj/4/wAABVAAAAILcG9zdE4LDlo
AAAd4AAAAEgABAAAEAAAAAFwEAAAAAAD9AABAAAAAABAAAAAABgABAAAAQAA6q1k4F8
PPPUACwQAAAAAN1ClWcAAAAA3UKVZwAAAAAD9AP0AAAAACAACAAAAAABAAAAAAGARMABwAAAA
AAgAAAAoACgAAAP8AAAAAQAQAAZAABQAAAokCzAAAAI8CiQLMAAAB6wAyAQgAAAIABQMAAAA
AAAAAABAAAAAABAAAAAABAAAAAABAAAAAABAAAAAABAAAAAABAAAAAABAAAAAABAAAAA
EAAAABAAAAAQAABAAAAAABAAAAAABAAAAAABAAAAAABAAAAAABAAAAAABAAAAAABAAAAA
D//8AAABAAQAAAAABAAIABQADAAQAAAAAABTgFqAYABlAG0AAAABwAAAAADdwP0AB8AXwCfAOM
A5gDsARIAAAEVDwUrAS8FPQE/BTsBHwUHFR8OPw8vDisBDw0XDw8jLw4/Dx8OJxUPAycHFW8EJwc
fBACXNx8EBxc3HwE/Ahc3Jz8DFzcnPwUnBy8DNycHLwQ1JyM/ASERIREzJREVHwgzITM/CDURNS8
IIyECGAICAwQEBAUFbQQDAwMBAQMDAwQFBQUEBAQDAgJvAgIDAwUFBQcGBwgICakJCQgJCAcHBwY
GBQQEAwIBAQEBAGMEBAUGBgCHBwgJCAkJCQgICAcGBwUFBQMDAgLeAQIDBQUHCAkJCwsMDA0NDg4
ODQwMCwoKCQgGBgUDAgEBAGMFBgYICQoKCwwMDQ4ODg0NDAwLCwkJCAcFBQMCohYTEhIiKyIOBQo

```

```

IBDQJNAEDBQYvHDANDg8IDBQ0FBQUDw8IFDQTEg8NEDAcLwUFBAEBNAo0BwgKECIqIg0RERMLuHF
xPgGW/ZDa/ucBAGUGCQoLBgYHAnAHBgYLCgkGBQIBAQIFBgkKCwYGB/4+AaIFBAQEAWICAgIDBAQ
EBQUFBAMDawEBawMDBAUFCQgJCAcHBwYGBQQEAWIBAQBAGMEBAUGBgCHBwgJCAkJCQgICAcGBwU
FBQMDAgICAgMDBQUFBwYHCAgICQkODQ0MDAsLCQkIBwYEAwIBAwMEBgcICAoLCwwMDQ0ODg0NDQw
LCGoJBwCGBAQCAQECAwUGBwcJCgoLDA0NDew2BQUICikkkRIIERILCTcKGBQTEhwwHA8MDAUGOBM
4AwEBAQI4EzclCwwRHTEcdRETEw0JOAkUEBAUKSQpBwgGBQI2fHEt/JQCKC39QwYGBgsKCQYFAGe
BAgUGCQoLBgYGA2wGBgYLCgkGBQIBAAACAAAAAPzA0wAAwALAAA3IRMhAzMTITUhJyFSAuq4/QP
rDrgCaf4uOv7dtAG9/kMB8Sh/AAAAAEAAAAAAxcD9AAFAAAATCQEXCQHpaCn+NzMB+/4FA8H+P/4
/MwH0AfQAAAAAAQAAAAAD9A0AAAUAAAEEnBwkBJwFZ52YBTQKbZwFM52b+sgKbZwAAAAIAAAAA/Q
DngAIAA4AABMRMzUhFTMRJQUVCQE1AYzuAQnx/pL+BgH6Ae7+EgHT/o/09AFx84NwAVv+rnEBUQA
AABIA3gABAAAAAAAEAAAAABAAAAAABAACAAQABAAAAAAACAACAABAAAAAADAAcADwABAAA
AAAAEAACAFgABAAAAAAAFaAsAHQABAAAAAAAGAACAABAAAAAAAKACwALwABAAAAAAALABIawWA
DAAEECQAAAAIAbQADAAEECQABAA4AbwADAAEECQACAA4AfQADAAEECQADAA4AiWADAAEECQAEAA4
AmQADAAEECQAFABYApwADAAEECQAGAA4AvQADAAEECQAKAFgAywADAAEECQALACQBIyB1LWJjb25
zUmVndWxhcmUtYmNvbNlLWJjb25zVmVyc2lvbiAxLjB1LWJjb25zRm9udCBnZW51cmF0ZWQgdXN
pbmcgU3luY2Zlc2lvbiBNZXRYbyBTdHVkaW93d3cuc3luY2Zlc2lvbi5jb20AIABlAC0AYgBjAG8
AbgBzAFIAZQBnAHUAbABhAHIAZQAtAGIAYwBvAG4AcwBlAC0AYgBjAG8AbgBzAFYAZQByAHMAaQB
vAG4AIAAxAC4AMABlAC0AYgBjAG8AbgBzAEYAbwBuAHQAIAbnAGUAbgBlAHIAIYQB0AGUAZAAGAHU
AcwBpAG4AZwAgAFMAEQBuAGMAZgB1AHMAaQBvAG4AIAbnAGUAdABYAG8AIABTAHQAdQBkAGkAbwB
3AHcAdwAuAHMAEQBuAGMAZgB1AHMAaQBvAG4ALgBjAG8AbQAAAAACAAAAAAAOAAAAAA
AAAAAAAAAAAAAAAAAYBAGEDAQQBBQEGAQCAE2RvY3VtZW50LXNldHRpbmctd2YOZm9sZGVyLW9
wZW4tMDERY2hldnJvbilyaWdodF8wMy0KY2hlY2stbWFiawhob3VzZS0wNAAAAA=)
format('truetype');
font-weight: normal;
font-style: normal;
}
.e-bicons {
font-family: 'e-bicons' !important;
font-size: 14px;
}
.e-folder:before {
content: "\e704";
}
.e-file:before {
content: "\e703";
}
#breadcrumb-control {
margin-left: auto;
margin-right: auto;
width: 60%;
}
#breadcrumb-control .header {
text-align: left;
padding-left: 10px;
}
#breadcrumb-control .e-control.e-breadcrumb {
text-align: left;
}
</style>

```

{% previewsample "page.domainurl/code-snippet/breadcrumb/getting-started-cs11" %}

### Icon only

To display only icons for the items, add icons using the [iconCss](#) property. In the following example, breadcrumb items were demonstrated with only icons by providing the [iconCss](#) property.

## APP.VUE

```
<template>
<div>
<ejs-breadcrumb>
  <e-breadcrumb-items>
    <e-breadcrumb-item iconCss="e-icons e-home"></e-breadcrumb-item>
    <e-breadcrumb-item iconCss="e-bicons e-folder"></e-breadcrumb-item>
    <e-breadcrumb-item iconCss="e-bicons e-file"></e-breadcrumb-item>
  </e-breadcrumb-items>
</ejs-breadcrumb>
</div>
</template>
<script>
import Vue from 'vue';
import { BreadcrumbPlugin } from "@syncfusion/ej2-vue-navigations";
Vue.use(BreadcrumbPlugin);
export default {
  data: function() {
    return {};
  }
}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-navigations/styles/material.css";
body {
  margin-top: 100px;
  text-align: center;
}
@font-face {
  font-family: 'e-bicons';
  src:
url(data:application/x-font-ttf;charset=utf-
8;base64,AAEAAAAKAIAAAwAgTlMvMj1lwSfkAAAEoAAAAVmNtYXDNH0dpAAABmAAAAD5nbHlMsrV
kRAAAAEgAAANoAGVhZB2Xb78AADQAAAAANmhoZWEIuUQqHAAAArAAAACRobXR4GAAAAAAAAAYAAAAA
YbG9jYQSCAv4AAAHYAAAAADm1heHABFwEfAAABCAAAACBuYW11Xj/4/wAABVAAAAIlcG9zdE4LDl0
AAAd4AAAAegABAAAEAAAAAFwEAAAAAAD9AABAAAAAAAAAAAAAAAAAAAAABgABAAAAQA6q1k4F8
PPPUACwQAAAAAN1C1WcAAAAA3UKVZwAAAAAD9AP0AAAAACAACAAAAAAAAAAAAEAAAAGARMABwAAAAA
AAgAAAAoACgAAAP8AAAAAAAAAAQQAAZAABQAAAokCzAAAAI8CiQLMAAAB6wAyAQgAAAIABQMAAAA
AAAAAAAAAAAAAAAAAAAAAUGZFZABA5wPnBwQAAAAAXAQAAAAAAAAABAAAAAAAAABAAAAAQAAAA
EAAAABAAAAAQAAAAEAAAAAAAAGAAAAAMAAAAUAMAAQAAABQABAAqAAAAABAAEAAEAEOcH//8AAOc
D//8AAAABAAQAAAAABAAIABQADAAQAAAAAAAABTgFqAYABlAG0AAAABwAAAAADdwP0AB8AXwCfAOM
A5gDsARIAAAAEVDwUrAS8FPQE/BTsBHwUHFR8OPw8vDisBDw0XDw8jLw4/Dx8OJxUPAYcHFw8EJwc
fBACXNx8EBxc3HwE/Ahc3Jz8DFzcnPwUnBy8DNycHLwQ1JyM/ASERIREzJREvHWgzITM/CDURNS8
IIyECGAICAwQEBAUFBBQDdAwMBAQMDAwQFBQUEBAQDAgJvAgIDAwUFBQcGBWgICakJCQgJCAcHBWY
GBQQEAwIBAQBEBagMEBAUGBgCHBwgJCAkJCQgICAcGBwUFBQMDAgLeAQIDBQUHCAkJCwsMDA0NDg4
ODQwMCwoKCQqGBgUDAgEBAgMFBgYICQoKCwwMDQ4ODg0NDAdwLCwkJCAcFBQMCOHYTEHiIKyIOBQo
IBDQJNAEDBQYvHDANDg8IDBQ0FBQUDw8IFDQTEg8NEDAcLwUFBAEBNAo0BwgKECIqIg0RERMLuHF
xPgGW/ZDa/ucBAgUGCQoLBgYHAnAHBgYLCgkGBQIBAQIFBgkKCwYGB/4+AAIFBAQEAWICAgIDBAQ
EBQUFBAMDawEBawMDBAUFCQgJCAcHBWYGBQQEAwIBAQBEBagMEBAUGBgCHBwgJCAkJCQgICAcGBwU
FBQMDAgICAgMDBQUFBWYHCAGICQKODQ0MDAsLCQkIBWYEAwIBAwMEBgICAO LCwwMDQ0ODg0NDQw
LCgoJBwcGBAQCAQECAwUGBwcJCgoLDA0NDew2BQUICikkKRIIERILCTcKGBQTEHwwHA8MDAUGOBM
4AwEBAQI4EzcLCwwRHTEcDRETEw0JOakUEBAUKSQpBwgGBQI2fHEt/JQcKc39QwYGBgsKCQYFAgE
BAGUGCQoLBgYGA2wGBgYLCgkGBQIBAAACAAAAAPzA0wAAwALAAA3IRMhAzMTITUhJyFSAuq4/QP
rDrgCaf4uOv7dtAG9/kMB8Sh/AAAAAAEAAAAAAxcD9AAFAAAATCQEXCQHpaCn+NzMB+/4FA8H+P/4
/MwH0AfQAAAAAAQAAAAAD9A0AAAAUAAAEEnBwkBJwFZ52YBTQKbZwFM52b+sqKbZwAAAAIAAAAAA/
```



```

DngAIAA4AABMRMzUhFTMRJQUVCQE1AYzuAQnx/pL+BgH6Ae7+EgHT/o/09AFx84NwAVv+rnEBUQA
AABIA3gABAAAAAAAAAAAAEAAAABAAAAAAAAABAACAAQABAAAAAAAAACAACAABAAAAAAAAADAACADwABAAA
AAAAEAACAFgABAAAAAAAAFAAsAHQABAAAAAAAAAGAAcAKAABAAAAAAAAAKACwALwABAAAAAAAAALABIWwA
DAAEECQAAAAIAbQADAAEECQABAA4AbwADAAEECQACAA4AfQADAAEECQADAA4AiwADAAEECQAEAA4
AmQADAAEECQAFABYApwADAAEECQAGAA4AvQADAAEECQAKAFgAywADAAEECQALACQBIyB1LWJjb25
zUmVndWxhcmUtYmNvbN1LWJjb25zVmVyc2lvbiAxLjB1LWJjb25zRm9udCBnZW51cmF0ZWQgdXN
pbmcgU3luY2Z1c2lvbiBNZXRYbyBTdHVkaW93d3cuc3luY2Z1c2lvbi5jb20AIABlAC0AYgBjAG8
AbgBzAFIAZQBnAHUAbABhAHIAZQAtAGIAYwBvAG4AcwBlAC0AYgBjAG8AbgBzAFYAZQByAHMAaQB
vAG4AIAAxAC4AMABlAC0AYgBjAG8AbgBzAEYAbwBuAHQAIAbnAGUAbgBlAHIAIYQB0AGUAZAAGAHU
AcwBpAG4AZwAgAFMAeQBvAGMAZgBlAHMAaQBvAG4AIAbnAGUAdABYAG8AIABTAHQAdQBkAGkAbwB
3AHcAdwAuAHMAeQBvAGMAZgBlAHMAaQBvAG4ALgBjAG8AbQAAAAACAAAAAoooooAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAYBAGEDAQQBBQEGAQcAE2RvY3VtZW50LXNldHRpbmctd2YOZm9sZGVyLW9
wZW4tMDERY2hlbnJvbi1yaWdodF8wMy0KY2h1Y2stbWFWYawhob3VzZS0wNAAAAA=)
format('truetype');
  font-weight: normal;
  font-style: normal;
}
.e-bicons {
  font-family: 'e-bicons' !important;
  font-size: 14px;
}
.e-folder:before {
  content: "\e704";
}
.e-file:before {
  content: "\e703";
}
</style>

```

{% previewsample "page.domainurl/code-snippet/breadcrumb/getting-started-cs12" %}

### Show icon only for first item

To show icon only for the first item in the Breadcrumb component, add icons to the first item using the [iconCss](#) property. In the following example, the icon was provided only for the first item by setting the [iconCss](#) property.

### APP.VUE

```

<template>
<div>
<ejs-breadcrumb :enableNavigation='false'>
  <e-breadcrumb-items>
    <e-breadcrumb-item iconCss= 'e-icons e-home' url=
'https://ej2.syncfusion.com/vue/demos/'></e-breadcrumb-item>
    <e-breadcrumb-item text= 'Components' url=
'https://ej2.syncfusion.com/vue/demos/datagrid/overview'></e-breadcrumb-
item>
    <e-breadcrumb-item text= 'Navigations' url=
'https://ej2.syncfusion.com/vue/demos/menu/default'></e-breadcrumb-item>
    <e-breadcrumb-item text= 'Breadcrumb' url=
'https://ej2.syncfusion.com/vue/demos/breadcrumb/default'></e-breadcrumb-
item>
  </e-breadcrumb-items>
</ejs-breadcrumb>
</div>
</template>

```

```

<script>
import Vue from 'vue';
import { BreadcrumbPlugin } from "@syncfusion/ej2-vue-navigations";
Vue.use(BreadcrumbPlugin);
export default {
  data: function() {
    return {};
  }
}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-navigations/styles/material.css";
body {
  margin-top: 100px;
  text-align: center;
}
</style>

```

{% previewsample "page.domainurl/code-snippet/breadcrumb/getting-started-cs13" %}

## Navigation in Vue Breadcrumb component

The breadcrumb item navigates to the path while clicking the item. To enable navigation, [url](#) property was bound to the items.

### URL

In the Breadcrumb component, the item represents the url. The breadcrumb items can be provided with either relative or absolute URL.

### Relative URL

The breadcrumb items with relative URL contain only the path but do not locate the path or server. The following example represents the breadcrumb items with relative url.

### APP.VUE

```

<template>
<div>
<ejs-breadcrumb :enableNavigation='false'>
  <e-breadcrumb-items>
    <e-breadcrumb-item text= 'Home' url= './breadcrumb'></e-breadcrumb-item>
    <e-breadcrumb-item text= 'Breadcrumb' url= './breadcrumb'></e-
breadcrumb-item>
    <e-breadcrumb-item text= 'Default' url= './breadcrumb/default'></e-
breadcrumb-item>
    <e-breadcrumb-item text= 'Icons' url= './breadcrumb/icons'></e-
breadcrumb-item>
    <e-breadcrumb-item text= 'Navigations' url=
'./breadcrumb/navigations'></e-breadcrumb-item>
    <e-breadcrumb-item text= 'Overflow' url= './breadcrumb/overflow'></e-
breadcrumb-item>
  </e-breadcrumb-items>
</ejs-breadcrumb>
</div>
</template>
<script>
import Vue from 'vue';

```

```
import { BreadcrumbPlugin } from "@syncfusion/ej2-vue-navigations";
Vue.use(BreadcrumbPlugin);
export default {
  data: function() {
    return {};
  }
}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-navigations/styles/material.css";
body {
  margin-top: 100px;
  text-align: center;
}
</style>
```

{% previewsample "page.domainurl/code-snippet/breadcrumb/getting-started-cs14" %}

### Absolute URL

The breadcrumb items with Absolute URL contain the path and locate to the resource if the absolute url is bound to the breadcrumb item. The following example represents the breadcrumb items with absolute url.

### APP.VUE

```
<template>
<div>
<ejs-breadcrumb :enableNavigation='false'>
  <e-breadcrumb-items>
    <e-breadcrumb-item text= 'Home' url=
'https://ej2.syncfusion.com/documentation/breadcrumb/introduction'></e-
breadcrumb-item>
    <e-breadcrumb-item text= 'Getting' url=
'https://ej2.syncfusion.com/documentation/breadcrumb/getting-started'></e-
breadcrumb-item>
    <e-breadcrumb-item text= 'Icons' url=
'https://ej2.syncfusion.com/documentation/breadcrumb/icons'></e-breadcrumb-
item>
    <e-breadcrumb-item text= 'Navigation' url=
'https://ej2.syncfusion.com/documentation/breadcrumb/navigation'></e-
breadcrumb-item>
    <e-breadcrumb-item text= 'Overflow' url=
'https://ej2.syncfusion.com/documentation/breadcrumb/overflow'></e-
breadcrumb-item>
  </e-breadcrumb-items>
</ejs-breadcrumb>
</div>
</template>
<script>
import Vue from 'vue';
import { BreadcrumbPlugin } from "@syncfusion/ej2-vue-navigations";
Vue.use(BreadcrumbPlugin);
export default {
  data: function() {
    return {};
  }
}
```

```

    }
  }
</script>
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-navigations/styles/material.css";
body {
    margin-top: 100px;
    text-align: center;
}
</style>

```

{% previewsample "page.domainurl/code-snippet/breadcrumb/getting-started-cs15" %}

### Enable navigation for last Breadcrumb item

The feature enables the last item of the Breadcrumb component by setting the [enableActiveItemNavigation](#) property to true. In the following example, the last item of the Breadcrumb was enabled.

### APP.VUE

```

<template>
<div>
<ejs-breadcrumb :enableNavigation='false' enableActiveItemNavigation='true'>
  <e-breadcrumb-items>
    <e-breadcrumb-item text= 'Home' url=
'https://ej2.syncfusion.com/documentation/breadcrumb/introduction'></e-
breadcrumb-item>
    <e-breadcrumb-item text= 'Getting' url=
'https://ej2.syncfusion.com/documentation/breadcrumb/getting-started'></e-
breadcrumb-item>
    <e-breadcrumb-item text= 'Icons' url=
'https://ej2.syncfusion.com/documentation/breadcrumb/icons'></e-breadcrumb-
item>
    <e-breadcrumb-item text= 'Navigation' url=
'https://ej2.syncfusion.com/documentation/breadcrumb/navigation'></e-
breadcrumb-item>
    <e-breadcrumb-item text= 'Overflow' url=
'https://ej2.syncfusion.com/documentation/breadcrumb/overflow'></e-
breadcrumb-item>
  </e-breadcrumb-items>
</ejs-breadcrumb>
</div>
</template>
<script>
import Vue from 'vue';
import { BreadcrumbPlugin } from "@syncfusion/ej2-vue-navigations";
Vue.use(BreadcrumbPlugin);
export default {
  data: function() {
    return {};
  }
}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";

```

```
@import "../node_modules/@syncfusion/ej2-navigations/styles/material.css";
body {
    margin-top: 100px;
    text-align: center;
}
</style>
```

{% previewsample "page.domainurl/code-snippet/breadcrumb/getting-started-cs16" %}

### Open URL in new page or tab

To open the breadcrumb item in a new page or tab, set the target property of the required item url to blank in the Breadcrumb component. In the following example, the target property of items url was set to blank by using the [beforeItemRender](#) event which locates to the path in the new tab.

### APP.VUE

```
<template>
<div>
<ejs-breadcrumb :beforeItemRender="beforeItemRenderHandler">
    <e-breadcrumb-items>
        <e-breadcrumb-item text= 'Home' url=
'https://ej2.syncfusion.com/documentation/breadcrumb/introduction'></e-
breadcrumb-item>
        <e-breadcrumb-item text= 'Getting' url=
'https://ej2.syncfusion.com/documentation/breadcrumb/getting-started'></e-
breadcrumb-item>
        <e-breadcrumb-item text= 'Icons' url=
'https://ej2.syncfusion.com/documentation/breadcrumb/icons'></e-breadcrumb-
item>
        <e-breadcrumb-item text= 'Navigation' url=
'https://ej2.syncfusion.com/documentation/breadcrumb/navigation'></e-
breadcrumb-item>
        <e-breadcrumb-item text= 'Overflow' url=
'https://ej2.syncfusion.com/documentation/breadcrumb/overflow'></e-
breadcrumb-item>
    </e-breadcrumb-items>
</ejs-breadcrumb>
</div>
</template>
<script>
import Vue from 'vue';
import { BreadcrumbPlugin } from "@syncfusion/ej2-vue-navigations";
Vue.use(BreadcrumbPlugin);
export default {
    data: function() {
        return {};
    },
    methods: {
        beforeItemRenderHandler: function(args) {
            if (args.element.children[0]) {
                args.element.children[0].target = "_blank";
            }
        }
    }
}
</script>
```

```

<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-navigations/styles/material.css";
body {
    margin-top: 100px;
    text-align: center;
}
</style>

```

{% previewsample "page.domainurl/code-snippet/breadcrumb/getting-started-cs17" %}

### Template in Vue Breadcrumb component

The Breadcrumb component provides a way to customize the items using [itemTemplate](#) and the separators using [separatorTemplate](#) properties.

#### Item Template

In the following example, Shopping Cart details are used as breadcrumb Items and the items are customized by the chips component using [itemTemplate](#).

#### APP.VUE

```

<template>
<div>
<ejs-breadcrumb cssClass="e-breadcrumb-chips" :itemTemplate="chipTemplate">
    <e-breadcrumb-items>
        <e-breadcrumb-item text="Cart"></e-breadcrumb-item>
        <e-breadcrumb-item text="Billing"></e-breadcrumb-item>
        <e-breadcrumb-item text="Shipping"></e-breadcrumb-item>
        <e-breadcrumb-item text="Payment"></e-breadcrumb-item>
    </e-breadcrumb-items>
</ejs-breadcrumb>
</div>
</template>
<script>
import Vue from 'vue';
import { BreadcrumbPlugin } from "@syncfusion/ej2-vue-navigations";
import { ChipListPlugin } from "@syncfusion/ej2-vue-buttons";
Vue.use(BreadcrumbPlugin);
Vue.use(ChipListPlugin);
export default {
    data: function() {
        return {
            chipTemplate:() => {
                return {
                    template : Vue.component('itemTemplate', {
                        template:
                            `<ejs-chiplist id="chip-default">
                                <e-chips>
                                    <e-chip :text="data.text"></e-chip>
                                </e-chips>
                            </ejs-chiplist>`
                    })
                }
            }
        },
    };

```

```

    }
  }
</script>
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-navigations/styles/material.css";
@import "../node_modules/@syncfusion/ej2-buttons/styles/material.css";
body {
    margin-top: 100px;
    text-align: center;
}
</style>

```

{% previewsample "page.domainurl/code-snippet/breadcrumb/getting-started-cs24" %}

### Separator Template

In the following example, the separators are customized with icons using [separatorTemplate](#).

#### APP.VUE

```

<template>
<div>
<ejs-breadcrumb :separatorTemplate= "separatorTemplate">
    <e-breadcrumb-items>
        <e-breadcrumb-item text="Cart"></e-breadcrumb-item>
        <e-breadcrumb-item text="Billing"></e-breadcrumb-item>
        <e-breadcrumb-item text="Shipping"></e-breadcrumb-item>
        <e-breadcrumb-item text="Payment"></e-breadcrumb-item>
    </e-breadcrumb-items>
</ejs-breadcrumb>
</div>
</template>
<script>
import Vue from 'vue';
import { BreadcrumbPlugin } from "@syncfusion/ej2-vue-navigations";
Vue.use(BreadcrumbPlugin);
export default {
    data: function() {
        return {
            separatorTemplate: '<span class="e-icons e-arrow"></span>',
        };
    }
}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-navigations/styles/material.css";
@import "../node_modules/@syncfusion/ej2-buttons/styles/material.css";
body {
    margin-top: 100px;
    text-align: center;
}
@font-face {
    font-family: 'e-bicons';
    src:

```

```
url(data:application/x-font-ttf;charset=utf-
8;base64,AAEAAAAKIAAAAwAgTlMvMj1wSfkAAAEoAAAAVmNtYXNHOdpAAABmAAAAAD5nbHlmsRv
kRAAAAEgAAANoaGVhZB2Xb78AAADQAAAAANmhoZWEIUQQHAAAArAAAACRobXR4GAAAAAAAYAAAA
YbG9jYQSCAv4AAAHYAAAAADm1heHABFwEfAAABCAAAACBuYW1lXj/4/wAABVAAAAIlcG9zdE4LDlo
AAAD4AAAAegABAAAEAAAAAFwEAAAAAAD9AABAAAAAABAAAAAABAAAAAQA6q1k4F8
PPPUACwQAAAAAAN1ClWcAAAAA3UKVZwAAAAAD9AP0AAAAACAACAAAAAABAAAAAGARMABwAAAA
AAgAAAAoACgAAAP8AAAAAQAQAAZAAABQAAAokCzAAAAI8CiQLMAAAB6wAyAQgAAAIABQMAAAA
AAAAAAAAAAAAAAAAAAAAAUGZFZABA5wPnBwQAAAAAXAQAAAAAABAAAAAABAAAAAQAQAAA
EAAAABAAAAQAQAAAEAAAAAAGAAAAAUAAMAAQAAABQABAAqAAAAABAAEAAEAEOch//8AAOc
D//8AAAABAAQAAAAABAIABQADAAQAAAAAABTgFqAYABLAG0AAAABwAAAAADdwP0AB8AXwCfAOM
A5gDsARIAAAEVDwUrAS8FPQE/BTsBHwUHFR8OPw8vDisBDw0XDw8jLw4/Dx8OJxUPAYcHFW8EJwc
fBACXNx8EBxc3HwE/Ahc3Jz8DFzcnPwUnBy8DNycHLwQ1JyM/ASERIREzJREVHWgzITM/CDURNS8
IIyECGAICAwQEBAUFBBQDAwMBAQMDAwQFBQQUEBAQDAgJvAgIDAUFBBQcGBwgICAKJCQgJCAcHBwY
GBQQEAWIBAQEBAgMEBAUGBgCHBwgJCAkJCQgICAcGBwUFBQMDAgLeAQIDBQUHCAkJCwsMDA0NDg4
ODQwMCwoKCQgGBgUDAgEBAgMFBgYICQoKCwwMDQ4ODg0NDAwLCwkJCAcFBQMCohYTEhIiKyIOBQo
IBDQJNAEDBQYvHDANDg8IDBQ0FBQUDw8IFDQTEg8NEDAcLwUFBAEBNAo0BwgKECIqIg0RERMLuHF
xPgGW/ZDa/ucBAgUGCQoLBgYHAnAHBgYLCgkGBQIBAQIFBgkKCwYGB/4+AaIFBAQEAWICAgIDBAQ
EBQUFBAMDawEBawMDBAUFCQgJCAcHBwYGBQQEAWIBAQEBAgMEBAUGBgCHBwgJCAkJCQgICAcGBwU
FBQMDAgICAgMDBQUFBwYHCAGICQkODQ0MDAsLCQkIBwYEAWIBAwMEBgICAOlCwwMDQ0ODg0NDQw
LCgoJBwCGBAQCAQECawUGBwCJCgoLDA0NDew2BQUICikkkRIIERILCTcKGBQTEhwwHA8MDAUGOBM
4AwEBAQI4EzcLCwwRHTEcDRETEw0JOakUEBAUKSQpBwgGBQI2fHEt/JQCKC39QwYGBgSKCQYFAgE
BAgUGCQoLBgYGA2wGBgYLCgkGBQIBAAACAAAAAPzA0wAAwALAA3IRMhAzMTITUhJyFSAuq4/QP
rDrgCaf4uOv7dtAG9/kMB8Sh/AAAAAEAAAAAxcD9AAFAAAATCQEXCQHAcn+NzMB+/4FA8H+P/4
/MwH0AfQAAAAAQAQAAAD9A0AAAAUAAEnBwkBJwFZ52YBTQKbZwFM52b+sgKbZwAAAAIAAAAA/Q
DngAIAA4AABMRmZUhFTMRJQUVCQE1AYzuAQnx/pL+BgH6Ae7+EgHT/o/09AFx84NwAVv+rnEBUQA
AABIA3gABAAAAAABAAAAAABAAAAAABAAcAAQABAAAAAACAACAABAAAAAADAACADwABAAA
AAAAEAACAFgABAAAAAFAAsAHQABAAAAAAGAACAABAAAAAABKACwALwABAAAAAALABIWwA
DAAEECQAAAAIAbQADAAEECQABAA4AbwADAAEECQACAA4AfQADAAEECQADAA4AiWADAAEECQAEAA4
AmQADAAEECQAFABYApwADAAEECQAGAA4AvQADAAEECQAKAFgAywADAAEECQALACQBIyBlLWJjb25
zUmVndWxhcmtUtYmNvbnNlLWJjb25zVmVyc2lubiAxLjBlLWJjb25zRm9udCBnZW5lcmF0ZWQgdXN
pbmctcgU3luY2Zlc2lubiBNZXRYbyBTdHVkaW93d3cuc3luY2Zlc2lubi5jb20AIBABAC0AYgBjAG8
AbgBzAFIAZQBnAHUAbAbhAHIAZQAtAGIAYwBvAG4AcwBlAC0AYgBjAG8AbgBzAFYAZQByAHMAaQB
vAG4AIAAxAAC4AMABlAC0AYgBjAG8AbgBzAEYABwBuAHQAIAABnAGUAbgBlAHIAAYQB0AGUAZAAGAHU
AcwBpAG4AZwAgAFMAeQBvAGMAZgBlAHMAaQBvAG4AIAABNAGUAdABYAG8AIAABTAHQAdQBkAGkABwB
3AHcAdwAuAHMAeQBvAGMAZgBlAHMAaQBvAG4ALgBjAG8AbQAAAAACAAAAAABAAAAAABAAAA
AAAAAAAAAAAAAAAAAYBAGEDAQQBBQEGAQCAE2RvY3VtZW50LXNldHRpbmctd2YOZm9sZGVyLW9
wZW4tMDERY2hlndnJvbilyaWdodF8wMy0KY2hlY2stbWFyYXhob3VzZS0wNAAAAA=)
format('trueType');
font-weight: normal;
font-style: normal;
}
.e-bicons {
font-family: 'e-bicons' !important;
font-size: 14px;
}
.e-arrow:before {
content: "\e253";
font-weight: 800;
font-size: 21px;
}
</style>
```

{% previewsample "page.domainurl/code-snippet/breadcrumb/getting-started-cs25" %}



### Customize Specific Item Template

The specific breadcrumb item can be customizable using `itemTemplate` with conditional rendering. In the following example, added the `span` element only for the `breadcrumb` text in breadcrumb item.

#### APP.VUE

```
<template>
<div>
<ejs-breadcrumb :enableNavigation='false' cssClass="e-specific-item-
template" :itemTemplate="specificItemTemplate">
  <e-breadcrumb-items>
    <e-breadcrumb-item text= 'Home' url=
'https://ej2.syncfusion.com/vue/demos/'></e-breadcrumb-item>
    <e-breadcrumb-item text= 'Components' url=
'https://ej2.syncfusion.com/vue/demos/datagrid/overview'></e-breadcrumb-
item>
    <e-breadcrumb-item text= 'Navigations' url=
'https://ej2.syncfusion.com/vue/demos/menu/default'></e-breadcrumb-item>
    <e-breadcrumb-item text= 'Breadcrumb' url=
'https://ej2.syncfusion.com/vue/demos/breadcrumb/default'></e-breadcrumb-
item>
  </e-breadcrumb-items>
</ejs-breadcrumb>
</div>
</template>
<script>
import Vue from 'vue';
import { BreadcrumbPlugin } from "@syncfusion/ej2-vue-navigations";
Vue.use(BreadcrumbPlugin);
export default {
  data: function() {
    return {
      specificItemTemplate: () => {
        return {
          template: Vue.component('specificItemTemplate', {
            template: `<div>
              <span v-if="data.text == 'Breadcrumb'" class="e-searchfor-
text"><span style="margin-right: 5px">Search for:</span>
              <a class="e-breadcrumb-text" :href="data.url" onclick="return
false">{{data.text}}</a></span>
              <a v-else class="e-breadcrumb-text" :href="data.url"
onclick="return false">{{data.text}}</a>
            </div>`
          })
        },
      },
    };
  }
}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-navigations/styles/material.css";
@import "../node_modules/@syncfusion/ej2-buttons/styles/material.css";
body {
  margin-top: 100px;
  text-align: center;
}
```

```
}
</style>
```

```
{% previewsample "page.domainurl/code-snippet/breadcrumb/getting-started-cs26" %}
```

## Overflow in Vue Breadcrumb component

In the Breadcrumb component, [maxItems](#) and [overflowMode](#) properties were used to limit the number of breadcrumb items to be displayed.

In the following example, the [maxItems](#) is set as 3 with [overflowMode](#) as Default. To prevent breadcrumb item navigation, the [enableNavigation](#) property has been set to false in the Breadcrumb component.

### APP.VUE

```
<template>
<div>
<ejs-breadcrumb maxItems='3' :enableNavigation='false'
:separatorTemplate="separatorTemplate">
  <e-breadcrumb-items>
    <e-breadcrumb-item text= 'Home' url= '../'></e-breadcrumb-item>
    <e-breadcrumb-item text= 'Breadcrumb' url= './breadcrumb'></e-
breadcrumb-item>
    <e-breadcrumb-item text= 'Default' url= './breadcrumb/default'></e-
breadcrumb-item>
    <e-breadcrumb-item text= 'Icons' url= './breadcrumb/icons'></e-
breadcrumb-item>
    <e-breadcrumb-item text= 'Navigations' url=
'./breadcrumb/navigations'></e-breadcrumb-item>
    <e-breadcrumb-item text= 'Overflow' url= './breadcrumb/overflow'></e-
breadcrumb-item>
  </e-breadcrumb-items>
</ejs-breadcrumb>
</div>
</template>
<script>
import Vue from 'vue';
import { BreadcrumbPlugin } from "@syncfusion/ej2-vue-navigations";
Vue.use(BreadcrumbPlugin);
export default {
  data: function() {
    return {
      separatorTemplate: '<span class="e-bicons e-arrow"></span>',
    };
  }
}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-navigations/styles/material.css";
@import "../node_modules/@syncfusion/ej2-buttons/styles/material.css";
body {
  margin-top: 100px;
  text-align: center;
}
@font-face {
```

```

font-family: 'e-bicons';
src:
url(data:application/x-font-ttf;charset=utf-
8;base64,AAEAAAAKIAAAAwAgTlMvMj1wSfkAAAEoAAAAVmNtYXDNHODpAAABmAAAAAD5nbHlmSRv
kRAAAAEgAAANoaGVhZB2Xb78AAADQAAAAANmhoZWEIUQQHAAAArAAAACRobXR4GAAAAAAAYAAAA
YbG9jYQSCAv4AAAHYAAAAADm1heHABFwEFAAABCAAAACBuYW11Xj/4/wAABVAAAAI1cG9zdE4LDlo
AAAd4AAAAegABAAAEAAAAAFwEAAAAAAD9AABAAAAAABAAAAAABgABAAAAQAA6q1k4F8
PPPUACwQAAAAAAN1ClWcAAAAA3UKVZWAAAAAD9AP0AAAACAACAAAAAABAAAAAGARMABwAAAA
AAgAAAAoACgAAAP8AAAAAQAQAAZAABQAAaokCzAAAAI8CiQLMAAAB6wAyAQgAAAIABQMAAAA
AAAAAAAAAAAAAAAAAAAAAUGZFZABA5wPnBwQAAAAAXAQAAAAAABAAAAAABAAAAAQAQAAA
EAAAABAAAAQAQAAAEAAAAAAGAAAAAUAAMAAQAAABQABAAqAAAABAAEAAEAOC//8AAOC
D//8AAAAABAAQAAAAABAAIABQADAAQAAAAAABATgFqAYABLAG0AAAABwAAAAADdwP0AB8AXwCfAOM
A5gDsARIAAAAEVDwUrAS8FPQE/BTsBHWUHF8OPw8vDisBDw0XDw8jLw4/Dx8OJxUPAychFw8EJwc
fBacXNx8EBxc3HwE/Ahc3Jz8DFzcnPwUnBy8DNyCHLwQ1JyM/ASERIREzJREVHwgzITM/CDURNS8
IIyECGAICAwQEBAUFBBQDawMBAQMDAwQFBQUEBAQDAgJvAgIDAwUFBQcGBwgICakJCQgJCAChBwY
GBQQEAWIBAQBAGMEBAUGBgCHBwgJCAkJCQgICACGBwUFBQMDAgLeAQIDBQUHCAkJCwsMDA0NDg4
ODQwMCwoKCQgGBgUDAgEBAGMFBgYICQoKCwwMDQ4ODg0NDawLCwkJCACFBQMCohYTEhIiKyIOBQo
IBDQJNAEDBQYvHDANDg8IDBQ0FBQUDw8IFDQTEg8NEDAcLwUFBAEBNAo0BwgKECIqIg0RERMLuHF
xPgGW/ZDa/ucBAGUGCQoLBgYHAnAHBgYLCgkGBQIBAQIFBgkKCwYGB/4+AaIFBAQEAWICAgIDBAQ
EBQUFBAMDawEBawMDBAUFCQgJCAChBwYGBQQEAWIBAQBAGMEBAUGBgCHBwgJCAkJCQgICACGBwU
FBQMDAgICAgMDBQUFBwYHCAgICQkODQ0MDAsLCQkIBwYEAWIBawMEBgICaOLCwwMDQ0ODg0NDQw
LCgoJBwCGBAQCAQECaWUGBwcJCgoLDA0NDew2BQUICikKRIIERILCTcKGBQTEhwwHA8MDAUGOBM
4AwEBAQI4EzCLCwwRHTECDRETEw0JOakUEBAUKSQpBwgGBQI2fHEt/JQCKC39QwYGBgsKCQYFAgE
BAgUGCQoLBgYGA2wGBgYLCgkGBQIBAAACAAAAAPzA0wAAwALAA3IRMhAzMTITUhJyFSAuq4/QP
rDrgCaf4uOv7dtAG9/kMB8Sh/AAAAAEAAAAAAxcD9AAFAAAATCQEXCQHpaCn+NzMB+/4FA8H+P/4
/MwH0AfQAAAAAAQAAAAAD9A0AAAUAAAEEnBwkBJwFZ52YBTQKbZwFM52b+sgKbZwAAAAIAAAAA/Q
DngAIAA4AABMRmZUhFTMRJQUVCQE1AYzuAQnx/pL+BgH6Ae7+EgHT/o/09AFx84NwAVv+rnEBUQA
AABIA3gABAAAAAABAAAAAABAAcAAQABAAAAAACAACAABAAAAAADAACADwABAAA
AAAAEAACAFgABAAAAAFAAAsAHQABAAAAAAGAACAABAAAAAABAAcAwALwABAAAAAALABIAWwA
DAAEECQAAAAIAbQADAAEECQABAA4AbwADAAEECQAGAA4AfQADAAEECQADAA4AiWADAAEECQAEAA4
AmQADAAEECQAFABYApwADAAEECQAGAA4AvQADAAEECQAKAFgAywADAAEECQALACQBIyB1LWJjb25
zUmVndWxhcmUtYmNvbN1LWJjb25zVmVyc2lvbiAxLjB1LWJjb25zRm9udCBnZW51cmF0ZWQgdXN
pbmcgU3luY2Z1c2lvbiBNZXRybyBTdHVkaW93d3cuc3luY2Z1c2lvbi5jb20AIABlAC0AYgBjAG8
AbgBzAFIAZQBnAHUAbABhAHIAZQAtAGIAYwBvAG4AcwBlAC0AYgBjAG8AbgBzAFYAZQByAHMAaQB
vAG4AIAAxAC4AMABlAC0AYgBjAG8AbgBzAEYABwBuAHQAIAbnAGUAbgBlAHIAZQB0AGUAZAAGAHU
AcwBpAG4AZwAgAFMAEQBuAGMAZgB1AHMAaQBvAG4AIAbnAGUAdABYAG8AIAbTAHQAdQBkAGkAbwB
3AHcAdwAuAHMAEQBuAGMAZgB1AHMAaQBvAG4ALgBjAG8AbQAAAAACAAAAAABAAAAAABAAAA
AAAAAAAAAAAAAAAAAYBAGEDAQQBBQEGAQcAE2RvY3VtZW50LXNldHRpbmctd2YOZm9sZGVyLW9
wZW4tMDERY2hldnJvbilyaWdodF8wMy0KY2hlY2stbWYyYXVhob3VzZS0wNAAAAA=)
format('trueType');
font-weight: normal;
font-style: normal;
}
.e-bicons {
font-family: 'e-bicons' !important;
font-size: 14px;
}
.e-breadcrumb .e-breadcrumb-separator + .e-breadcrumb-separator {
padding-left: 0;
margin-left: -8px;
}
.e-bicons.e-arrow {
margin-top: -2px;
}
.e-arrow:before {
content: "\e706";
font-weight: 800;

```

```
font-size: 9px;
}
</style>
```

{% previewsample "page.domainurl/code-snippet/breadcrumb/getting-started-cs18" %}

The following overflow modes are available in the Breadcrumb component.

- Collapsed
- Menu
- Wrap
- Scroll
- Hidden
- None

### Collapsed

Collapsed mode shows the first and last Breadcrumb items and hides the remaining items with a collapsed icon. When the collapsed icon is clicked, all items become visible and navigable.

#### APP.VUE

```
<template>
<div>
<ejs-breadcrumb maxItems='3' :enableNavigation='false'
overflowMode="Collapsed">
  <e-breadcrumb-items>
    <e-breadcrumb-item text= 'Home' url= '../'></e-breadcrumb-item>
    <e-breadcrumb-item text= 'Breadcrumb' url= './breadcrumb'></e-
breadcrumb-item>
    <e-breadcrumb-item text= 'Default' url= './breadcrumb/default'></e-
breadcrumb-item>
    <e-breadcrumb-item text= 'Icons' url= './breadcrumb/icons'></e-
breadcrumb-item>
    <e-breadcrumb-item text= 'Navigations' url=
'./breadcrumb/navigations'></e-breadcrumb-item>
    <e-breadcrumb-item text= 'Overflow' url= './breadcrumb/overflow'></e-
breadcrumb-item>
  </e-breadcrumb-items>
</ejs-breadcrumb>
</div>
</template>
<script>
import Vue from 'vue';
import { BreadcrumbPlugin } from "@syncfusion/ej2-vue-navigations";
Vue.use(BreadcrumbPlugin);
export default {
  data: function() {
    return {
    };
  }
}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
```

```
@import "../node_modules/@syncfusion/ej2-navigations/styles/material.css";
@import "../node_modules/@syncfusion/ej2-buttons/styles/material.css";
body {
    margin-top: 100px;
    text-align: center;
}
</style>
```

{% previewsample "page.domainurl/code-snippet/breadcrumb/getting-started-cs19" %}

## Menu

Menu mode shows the number of Breadcrumb items that can be accommodated within the container space and creates a submenu with the remaining items.

## APP.VUE

```
<template>
<div>
<ejs-breadcrumb maxItems='3' :enableNavigation='false' overflowMode="Menu">
    <e-breadcrumb-items>
        <e-breadcrumb-item text= 'Home' url= '../'></e-breadcrumb-item>
        <e-breadcrumb-item text= 'Breadcrumb' url= './breadcrumb'></e-
breadcrumb-item>
        <e-breadcrumb-item text= 'Default' url= './breadcrumb/default'></e-
breadcrumb-item>
        <e-breadcrumb-item text= 'Icons' url= './breadcrumb/icons'></e-
breadcrumb-item>
        <e-breadcrumb-item text= 'Navigations' url=
'./breadcrumb/navigations'></e-breadcrumb-item>
        <e-breadcrumb-item text= 'Overflow' url= './breadcrumb/overflow'></e-
breadcrumb-item>
    </e-breadcrumb-items>
</ejs-breadcrumb>
</div>
</template>
<script>
import Vue from 'vue';
import { BreadcrumbPlugin } from "@syncfusion/ej2-vue-navigations";
Vue.use(BreadcrumbPlugin);
export default {
    data: function() {
        return {
        };
    }
}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-navigations/styles/material.css";
@import "../node_modules/@syncfusion/ej2-buttons/styles/material.css";
body {
    margin-top: 100px;
    text-align: center;
}
</style>
```

```
{% previewsample "page.domainurl/code-snippet/breadcrumb/getting-started-cs20" %}
```

### Wrap

Wrap mode wraps the items to multiple lines when the Breadcrumb's width exceeds the container space.

### APP.VUE

```
<template>
<div>
<ejs-breadcrumb maxItems='3' :enableNavigation='false' overflowMode="Wrap">
  <e-breadcrumb-items>
    <e-breadcrumb-item text= 'Home' url= '../'></e-breadcrumb-item>
    <e-breadcrumb-item text= 'Breadcrumb' url= './breadcrumb'></e-
breadcrumb-item>
    <e-breadcrumb-item text= 'Default' url= './breadcrumb/default'></e-
breadcrumb-item>
    <e-breadcrumb-item text= 'Icons' url= './breadcrumb/icons'></e-
breadcrumb-item>
    <e-breadcrumb-item text= 'Navigations' url=
'./breadcrumb/navigations'></e-breadcrumb-item>
    <e-breadcrumb-item text= 'Overflow' url= './breadcrumb/overflow'></e-
breadcrumb-item>
  </e-breadcrumb-items>
</ejs-breadcrumb>
</div>
</template>
<script>
import Vue from 'vue';
import { BreadcrumbPlugin } from "@syncfusion/ej2-vue-navigations";
Vue.use(BreadcrumbPlugin);
export default {
  data: function() {
    return {
    };
  }
}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-navigations/styles/material.css";
@import "../node_modules/@syncfusion/ej2-buttons/styles/material.css";
body {
  margin-top: 100px;
  text-align: center;
}
</style>
```

```
{% previewsample "page.domainurl/code-snippet/breadcrumb/getting-started-cs21" %}
```

### Scroll

Scroll mode shows an HTML scroll bar when the Breadcrumb's width exceeds the container space.

### APP.VUE

```
<template>
<div>
```

```

<ejs-breadcrumb maxItems='3' :enableNavigation='false'
overflowMode="Scroll">
  <e-breadcrumb-items>
    <e-breadcrumb-item text= 'Home' url= '../'></e-breadcrumb-item>
    <e-breadcrumb-item text= 'Breadcrumb' url= './breadcrumb'></e-
breadcrumb-item>
    <e-breadcrumb-item text= 'Default' url= './breadcrumb/default'></e-
breadcrumb-item>
    <e-breadcrumb-item text= 'Icons' url= './breadcrumb/icons'></e-
breadcrumb-item>
    <e-breadcrumb-item text= 'Navigations' url=
'./breadcrumb/navigations'></e-breadcrumb-item>
    <e-breadcrumb-item text= 'Overflow' url= './breadcrumb/overflow'></e-
breadcrumb-item>
  </e-breadcrumb-items>
</ejs-breadcrumb>
</div>
</template>
<script>
import Vue from 'vue';
import { BreadcrumbPlugin } from "@syncfusion/ej2-vue-navigations";
Vue.use(BreadcrumbPlugin);
export default {
  data: function() {
    return {
    };
  }
}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-navigations/styles/material.css";
@import "../node_modules/@syncfusion/ej2-buttons/styles/material.css";
body {
  margin-top: 100px;
  text-align: center;
}
</style>

```

{% previewsample "page.domainurl/code-snippet/breadcrumb/getting-started-cs22" %}

### Hidden

Hidden mode shows the maximum number of items possible in the container space and hides the remaining items. Clicking on a previous item will make the hidden item visible.

### APP.VUE

```

<template>
<div>
<ejs-breadcrumb maxItems='3' :enableNavigation='false'
overflowMode="Hidden">
  <e-breadcrumb-items>
    <e-breadcrumb-item text= 'Home' url= '../'></e-breadcrumb-item>
    <e-breadcrumb-item text= 'Breadcrumb' url= './breadcrumb'></e-
breadcrumb-item>

```

```

    <e-breadcrumb-item text= 'Default' url= './breadcrumb/default'></e-
breadcrumb-item>
    <e-breadcrumb-item text= 'Icons' url= './breadcrumb/icons'></e-
breadcrumb-item>
    <e-breadcrumb-item text= 'Navigations' url=
'./breadcrumb/navigations'></e-breadcrumb-item>
    <e-breadcrumb-item text= 'Overflow' url= './breadcrumb/overflow'></e-
breadcrumb-item>
  </e-breadcrumb-items>
</ejs-breadcrumb>
</div>
</template>
<script>
import Vue from 'vue';
import { BreadcrumbPlugin } from "@syncfusion/ej2-vue-navigations";
Vue.use(BreadcrumbPlugin);
export default {
  data: function() {
    return {
    };
  }
}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-navigations/styles/material.css";
@import "../node_modules/@syncfusion/ej2-buttons/styles/material.css";
body {
  margin-top: 100px;
  text-align: center;
}
</style>

```

{% previewsample "page.domainurl/code-snippet/breadcrumb/getting-started-cs23" %}

None

None mode shows all the items on a single line.

### Accessibility in Vue Breadcrumb component

The Breadcrumb component followed the accessibility guidelines and standards, including [ADA](#), [Section 508](#), [WCAG 2.2](#) standards, and [WCAG roles](#) that are commonly used to evaluate accessibility.

The accessibility compliance for the Breadcrumb component is outlined below.

| Accessibility Criteria | Compatibility |

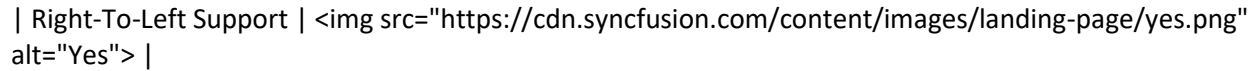
| -- | -- |

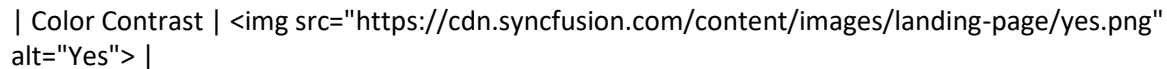
| [WCAG 2.2](#) Support |  |

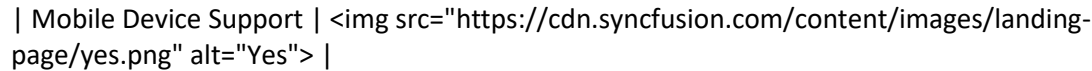
| [Section 508](#) Support |  |

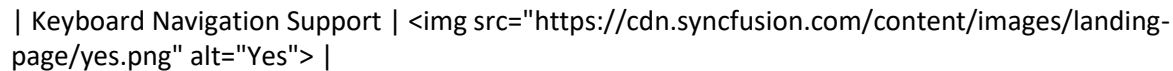
| Screen Reader Support |  |

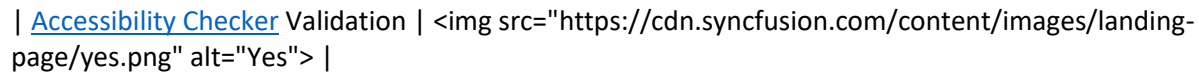


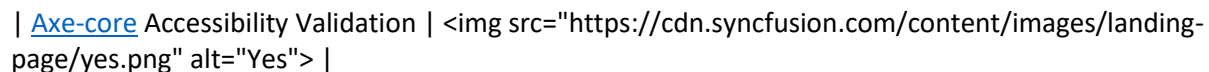
| Right-To-Left Support |  |

| Color Contrast |  |

| Mobile Device Support |  |

| Keyboard Navigation Support |  |

| [Accessibility Checker](#) Validation |  |

| [Axe-core](#) Accessibility Validation |  |

<style>

```
.post .post-content img {  
display: inline-block;  
margin: 0.5em 0;  
}
```

</style>

<div><img alt="Yes" data-bbox="112 485 844 519"/> - All features of the component meet the requirement.</div>

<div><img alt="Intermediate" data-bbox="112 526 802 561"/> - Some features of the component do not meet the requirement.</div>

<div><img alt="No" data-bbox="112 568 844 602"/> - The component does not meet the requirement.</div>

### WAI-ARIA attributes

The Breadcrumb component followed the [WAI-ARIA](#) patterns to meet the accessibility. The following ARIA attributes are used in the Breadcrumb component:

| Attributes | Purpose |

| --- | --- |

| `aria-label` | Indicates the breadcrumb item text. |

| `aria-disabled` | Indicates the state of breadcrumb item whether it is disabled. |

### Keyboard interaction

The Breadcrumb component followed the [keyboard interaction](#) guideline, making it easy for people who use assistive technologies (AT) and those who completely rely on keyboard navigation. The following keyboard shortcuts are supported by the Breadcrumb component.

| **Press** | **To do this** |

| --- | --- |

| Tab | Navigate to the next item and also next item in the popup of menu type overflow. |

| Shift + Tab | Navigate to the previous item also previous item in the popup of menu type overflow. |

| Enter key in normal mode | Select the breadcrumb item. |

| Enter key in normal mode | To open the popup of menu type overflow mode when you press enter on collapsed button and It will expand the items of collapsed type overflow mode when you press enter on collapsed button. |

### Ensuring accessibility

The Breadcrumb component's accessibility levels are ensured through an [accessibility-checker](#) and [axe-core](#) software tools during automated testing.

The accessibility compliance of the Breadcrumb component is shown in the following sample. Open the [sample](#) in a new window to evaluate the accessibility of the Breadcrumb component with accessibility tools.

```
{% previewsample "page.domainurl/code-snippet/breadcrumb/accessibility-cs1" %}
```

See also

- [Accessibility in Syncfusion Vue components](#)

## Bullet Chart

### Getting Started with the Vue Bullet chart Component in Vue 2

This article provides a step-by-step guide for setting up a Vue 2 project using [Vue-CLI](#) and integrating the Syncfusion Vue Bullet chart component

### Prerequisites

[System requirements for Syncfusion Vue UI components](#)

### Dependencies

Below is the list of minimum dependencies required to use the Bullet Chart component.

```
`javascript
|-- @syncfusion/ej2-vue-charts
|-- @syncfusion/ej2-base
|-- @syncfusion/ej2-data
|-- @syncfusion/ej2-pdf-export
|-- @syncfusion/ej2-file-utils
|-- @syncfusion/ej2-compression
|-- @syncfusion/ej2-charts
|-- @syncfusion/ej2-vue-base
|-- @syncfusion/ej2-svg-base
`
```

### Setting up the Vue 2 project

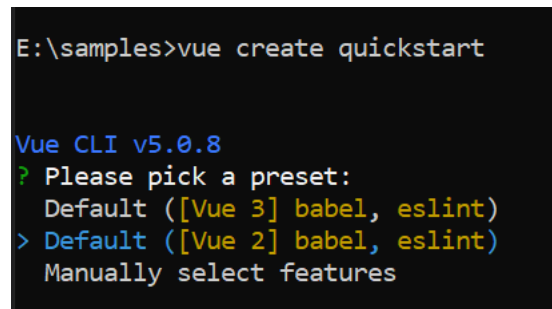
To generate a Vue 2 project using Vue-CLI, use the [vue create](#) command. Follow these steps to install Vue CLI and create a new project:

```
`bash
npm install -g @vue/cli
vue create quickstart
cd quickstart
npm run serve
`
```

or

```
`bash
yarn global add @vue/cli
vue create quickstart
cd quickstart
yarn run serve
`
```

When creating a new project, choose the option `Default ([Vue 2] babel, eslint)` from the menu.



```
E:\samples>vue create quickstart

Vue CLI v5.0.8
? Please pick a preset:
  Default ([Vue 3] babel, eslint)
> Default ([Vue 2] babel, eslint)
  Manually select features
```

Once the `quickstart` project is set up with default settings, proceed to add Syncfusion components to the project.

### Add Syncfusion Vue packages

Syncfusion packages are available at [npmjs.com](https://www.npmjs.com). To use Vue components, install the required npm package.

This article uses the [Vue Bullet chart component](#) as an example. Install the `@syncfusion/ej2-vue-charts` package by running the following command:

```
`bash
npm install @syncfusion/ej2-vue-charts --save
`

or
```

```
`bash
```

```
yarn add @syncfusion/ej2-vue-charts
```

```
,
```

The **--save** will instruct NPM to include the Bullet Chart package inside of the **dependencies** section of the **package.json**.

### Add Syncfusion Vue component

Follow the below steps to add the Vue Bullet chart component using:

1\ First, import and register the Bullet chart component in the **script** section of the **src/App.vue** file.

#### ~/SRC/APP.VUE

```
<script>
import { BulletChartComponent } from '@syncfusion/ej2-vue-charts';
export default {
  components: {
    'ejs-bulletchart': BulletChartComponent
  }
}
</script>
```

2\ In the **template** section, define the Bullet chart component.

#### ~/SRC/APP.VUE

```
<template>
<div>
<ejs-bulletchart id="bulletChart"> </ejs-bulletchart>
</div>
</template>
```

Here is the summarized code for the above steps in the **src/App.vue** file:

#### ~/SRC/APP.VUE

```
<template>
  <div>
    <ejs-bulletchart id="bulletChart"> </ejs-bulletchart>
  </div>
</template>
<script>
import { BulletChartComponent } from '@syncfusion/ej2-vue-charts';
export default {
  components: {
    'ejs-bulletchart': BulletChartComponent
  },
  data () {
    return {
    }
  }
}
</script>
```

### Run the project

To run the project, use the following command:

```
`bash
```

```
npm run serve
```

```
,
```

or

```
`bash
```

```
yarn run serve
```

```
,
```

```
{% previewsample "page.domainurl/code-snippet/bullet-chart/getting-started/initialize-cs1" %}
```

### Module Injection

Bullet Chart component are segregated into individual feature-wise modules. In order to use a particular feature, you need to inject its feature module using `provide` method. In the current application, we are going to use tooltip feature of the Bullet Chart.

- **BulletTooltip** - Inject this provider to use tooltip feature.

These modules should be injected to the provide method as follows,

```
`ts
```

```
import { BulletChartComponent, BulletTooltip } from "@syncfusion/ej2-vue-charts";
```

```
export default {
```

```
  components: {
```

```
    'ejs-bullectchart': BulletChartComponent
```

```
  },
```

```
  provide: {
```

```
    bulletChart: [BulletTooltip]
```

```
  }
```

```
};
```

```
,
```

### BulletChart With Data

This section explains how to plot local data to the Bullet Chart.

```
`ts
```

```
let data: any[] = [
```

```
  { value: 100, target: 80 },
```

```
  { value: 200, target: 180 },
```

```
  { value: 300, target: 280 },
```

```
{ value: 400, target: 380 },
{ value: 500, target: 480 },
];
,
```

Now assign the local data to [dataSource](#) property. **value** and **target** values should be mapped with [valueField](#) and [targetField](#) respectively.

#### ~/SRC/APP.VUE

```
<template>
  <div>
    <ejs-bulletchart id="bulletChart"
      :dataSource="data"
      valueField="value"
      targetField="target"
      height="300"
      :minimum="minimum"
      :maximum="maximum"
      :interval="interval"
    > </ejs-bulletchart>
  </div>
</template>
<script>
import { BulletChartComponent } from '@syncfusion/ej2-vue-charts';
export default {
  components: {
    'ejs-bulletchart': BulletChartComponent
  },
  data () {
    return {
      data: [
        { value: 100, target: 80 },
        { value: 200, target: 180 },
        { value: 300, target: 280 },
        { value: 400, target: 380 },
        { value: 500, target: 480 },
      ],
      minimum: 0, maximum: 300, interval: 50
    }
  }
}
</script>
```

{% previewsample "page.domainurl/code-snippet/bullet-chart/getting-started/datasource-cs1" %}

#### Add Bullet Chart Title

You can add a title using [title](#) property to the Bullet Chart to provide quick information to the user about the data plotted in the Bullet Chart.

#### ~/SRC/APP.VUE

```
<template>
  <div>
```

```

    <ejs-bulletchart id="bulletChart"
      :dataSource="data"
      valueField="value"
      targetField="target"
      :minimum="minimum"
      :maximum="maximum"
      :interval="interval"
      title="Revenue"
    > </ejs-bulletchart>
  </div>
</template>
<script>
import { BulletChartComponent } from '@syncfusion/ej2-vue-charts';
export default {
  components: {
    'ejs-bulletchart': BulletChartComponent
  },
  data () {
    return {
      data: [{ value: 270, target: 250 }],
      minimum: 0, maximum: 300, interval: 50
    }
  }
}
</script>

```

{% previewsample "page.domainurl/code-snippet/bullet-chart/getting-started/title-cs1" %}

### Ranges

You can add a range using `e-bullet-range` of the `e-bullet-range-collection`.

### ~/SRC/APP.VUE

```

<template>
  <div>
    <ejs-bulletchart id="bulletChart"
      :dataSource="data"
      valueField="value"
      targetField="target"
      :minimum="minimum"
      :maximum="maximum"
      :interval="interval"
      title="Revenue"
    >
      <e-bullet-range-collection>
        <e-bullet-range end="100" color="red"></e-bullet-range>
        <e-bullet-range end="200" color="blue"></e-bullet-range>
        <e-bullet-range end="300" color="green"></e-bullet-range>
      </e-bullet-range-collection>
    </ejs-bulletchart>
  </div>
</template>
<script>
import { BulletChartComponent, BulletRangeCollectionDirective,
BulletRangeDirective } from '@syncfusion/ej2-vue-charts';
export default {

```

```

components: {
  'ejs-bulletchart': BulletChartComponent,
  'e-bullet-range-collection': BulletRangeCollectionDirective,
  'e-bullet-range': BulletRangeDirective
},
data () {
  return {
    data: [{ value: 270, target: 250 }],
    minimum: 0, maximum: 300, interval: 50
  }
}
}
</script>

```

{% previewsample "page.domainurl/code-snippet/bullet-chart/getting-started/ranges-cs1" %}

### Tooltip

You can use tooltip for the Bullet Chart by setting the [enable](#) property to true in [tooltip](#) object and by injecting the `BulletTooltip` module using `provide` method.

### ~/SRC/APP.VUE

```

<template>
  <div>
    <ejs-bulletchart id="bulletChart"
      :dataSource="data"
      valueField="value"
      targetField="target"
      :minimum="minimum"
      :maximum="maximum"
      :interval="interval"
      title="Revenue"
      :tooltip="tooltip"
    > </ejs-bulletchart>
  </div>
</template>
<script>
import { BulletChartComponent, BulletTooltip } from '@syncfusion/ej2-vue-charts';
export default {
  components: {
    'ejs-bulletchart': BulletChartComponent
  },
  data () {
    return {
      data: [{ value: 270, target: 250 }],
      minimum: 0, maximum: 300, interval: 50,
      tooltip: { enable: true }
    }
  },
  provide: {
    bulletChart: [BulletTooltip]
  }
}
</script>

```



```
{% previewsample "page.domainurl/code-snippet/bullet-chart/getting-started/tooltip-cs1" %}
```

## Getting Started with the Vue Bullet Chart Component in Vue 3

This article provides a step-by-step guide for setting up a [Vite](#) project with a JavaScript environment and integrating the Syncfusion Vue Bullet Chart component using the [Composition API](#) / [Options API](#).

The **Composition API** is a new feature introduced in Vue.js 3 that provides an alternative way to organize and reuse component logic. It allows developers to write components as functions that use smaller, reusable functions called composition functions to manage their properties and behavior.

The **Options API** is the traditional way of writing Vue.js components, where the component logic is organized into a series of options that define the component's properties and behavior. These options include data, methods, computed properties, watchers, lifecycle hooks, and more.

### Prerequisites

#### [System requirements for Syncfusion Vue UI components](#)

#### Set up the Vite project

A recommended approach for beginning with Vue is to scaffold a project using [Vite](#). To create a new Vite project, use one of the commands that are specific to either NPM or Yarn.

```
`bash
```

```
npm create vite@latest
```

```
,
```

```
or
```

```
`bash
```

```
yarn create vite
```

```
,
```

Using one of the above commands will lead you to set up additional configurations for the project as below:

1. Define the project name: We can specify the name of the project directly. Let's specify the name of the project as **my-project** for this article.

```
`bash
```

```
? Project name: » my-project
```

```
,
```

2. Select **Vue** as the framework. It will create a Vue 3 project.

```
`bash
```

```
? Select a framework: » - Use arrow-keys. Return to submit.
```

```
Vanilla
```

```
Vue
```

```
React
```

```
Preact
```

Lit

Svelte

Others

,

3. Choose **JavaScript** as the framework variant to build this Vite project using JavaScript and Vue.

```
`bash
```

? Select a variant: » - Use arrow-keys. Return to submit.

JavaScript

TypeScript

Customize with create-vue ↗

Nuxt ↗

,

4. Upon completing the aforementioned steps to create the **my-project**, run the following command to install its dependencies:

```
`bash
```

```
cd my-project
```

```
npm install
```

,

or

```
`bash
```

```
cd my-project
```

```
yarn install
```

,

Now that **my-project** is ready to run with default settings, let's add Syncfusion components to the project.

#### Add Syncfusion Vue packages

Syncfusion Vue component packages are available at [npmjs.com](https://www.npmjs.com). To use Syncfusion Vue components in the project, install the corresponding npm package.

This article uses the [Vue Bullet Chart component](#) as an example. To use the Vue Bullet Chart component in the project, the **@syncfusion/ej2-vue-charts** package needs to be installed using the following command:

```
`bash
```

```
npm install @syncfusion/ej2-vue-charts --save
```

,

or

```
`bash
```

```
yarn add @syncfusion/ej2-vue-charts
```

```
,
```

### Add Syncfusion Vue component

Follow the below steps to add the Vue Bullet Chart component using **Composition API** or **Options API**:

1.First, import and register the Bullet Chart component and its child directives in the **script** section of the **src/App.vue** file. If you are using the **Composition API**, you should add the **setup** attribute to the **script** tag to indicate that Vue will be using the **Composition API**.

#### **COMPOSITION API (~SRC/APP.VUE)**

```
<script setup>
import { BulletChartComponent as EjsBulletchart, BulletTooltip,
BulletRangeCollectionDirective as EBulletRangeCollection,
BulletRangeDirective as EBulletRange } from "@syncfusion/ej2-vue-charts";
</script>
```

#### **OPTIONS API (~SRC/APP.VUE)**

```
<script>
import { BulletChartComponent, BulletTooltip,
BulletRangeCollectionDirective, BulletRangeDirective } from
"@syncfusion/ej2-vue-charts";
//Component registration
export default {
name: "App",
components: {
"ejs-bulletchart": BulletChartComponent,
"e-bullet-range-collection": BulletRangeCollectionDirective,
"e-bullet-range": BulletRangeDirective
}
}
</script>
```

2.In the **template** section, define the Bullet Chart component with the [dataSource](#) property.

#### **~/SRC/APP.VUE**

```
<template>
<ejs-bulletchart :dataSource="data" :valueField="valueField"
:tooltip="tooltip" :targetField="targetField" :height="height"
:title="title" :minimum="minimum" :maximum="maximum" :interval="interval">
<e-bullet-range-collection>
<e-bullet-range end="100" color="red"></e-bullet-range>
<e-bullet-range end="200" color="blue"></e-bullet-range>
<e-bullet-range end="300" color="green"></e-bullet-range>
</e-bullet-range-collection>
</ejs-bulletchart>
</template>
```

3.Declare the values for the **dataSource** property in the **script** section.

**COMPOSITION API (~SRC/APP.VUE)**

```
<script setup>
const data = [
  { value: 100, target: 80 },
  { value: 200, target: 180 },
  { value: 300, target: 280 },
  { value: 400, target: 180 },
  { value: 500, target: 230 }
];
const minimum = 0;
const maximum = 300;
const interval = 50;
const tooltip = { enable: true };
const title = 'Revenue';
const height = '300px';
const targetField = 'target';
const valueField = 'value';
</script>
```

**OPTIONS API (~SRC/APP.VUE)**

```
<script>
data() {
  return {
    data: [
      { value: 100, target: 80 },
      { value: 200, target: 180 },
      { value: 300, target: 280 },
      { value: 400, target: 180 },
      { value: 500, target: 230 }
    ],
    minimum: 0,
    maximum: 300,
    interval: 50,
    tooltip: { enable: true },
    title: 'Revenue',
    height: '300px',
    targetField: 'target',
    valueField: 'value'
  };
}
</script>
```

Here is the summarized code for the above steps in the **src/App.vue** file:

**COMPOSITION API (~SRC/APP.VUE)**

```
<template>
<ejs-bulletchart :dataSource="data" :valueField="valueField"
:tooltip="tooltip" :targetField="targetField" :height="height"
:title="title" :minimum="minimum" :maximum="maximum" :interval="interval">
<e-bullet-range-collection>
<e-bullet-range end="100" color="red"></e-bullet-range>
<e-bullet-range end="200" color="blue"></e-bullet-range>
<e-bullet-range end="300" color="green"></e-bullet-range>
</template>
```

```

</e-bullet-range-collection>
</ejs-bulletchart>
</template>
<script setup>
import { provide } from 'vue';
import { BulletChartComponent as EjsBulletchart, BulletTooltip,
BulletRangeCollectionDirective as EBulletRangeCollection,
BulletRangeDirective as EBulletRange } from "@syncfusion/ej2-vue-charts";
const data = [
{ value: 100, target: 80 },
{ value: 200, target: 180 },
{ value: 300, target: 280 },
{ value: 400, target: 180 },
{ value: 500, target: 230 }
];
const minimum = 0;
const maximum = 300;
const interval = 50;
const tooltip = { enable: true };
const title = 'Revenue';
const height = '300px';
const targetField = 'target';
const valueField = 'value';
const bulletChart = [ BulletTooltip ];
provide('bulletChart', bulletChart);
</script>

```

### OPTIONS API (~SRC/APP.VUE)

```

<template>
<ejs-bulletchart :dataSource="data" :valueField="valueField"
:tooltip="tooltip" :targetField="targetField" :height="height"
:title="title" :minimum="minimum" :maximum="maximum" :interval="interval">
<e-bullet-range-collection>
<e-bullet-range end="100" color="red"></e-bullet-range>
<e-bullet-range end="200" color="blue"></e-bullet-range>
<e-bullet-range end="300" color="green"></e-bullet-range>
</e-bullet-range-collection>
</ejs-bulletchart>
</template>
<script>
import { BulletChartComponent, BulletTooltip,
BulletRangeCollectionDirective, BulletRangeDirective } from
"@syncfusion/ej2-vue-charts";
//Component registration
export default {
name: "App",
components: {
"ejs-bulletchart": BulletChartComponent,
"e-bullet-range-collection": BulletRangeCollectionDirective,
"e-bullet-range": BulletRangeDirective
},
data() {
return {
data: [
{ value: 100, target: 80 },

```

```
{ value: 200, target: 180 },  
{ value: 300, target: 280 },  
{ value: 400, target: 180 },  
{ value: 500, target: 230 }  
],  
minimum: 0,  
maximum: 300,  
interval: 50,  
tooltip: { enable: true },  
title: 'Revenue',  
height: '300px',  
targetField: 'target',  
valueField: 'value'  
};  
},  
provide: {  
  bulletChart: [ BulletTooltip ]  
},  
};  
</script>
```

### Run the project

To run the project, use the following command:

```
`bash
```

```
npm run dev
```

```
,
```

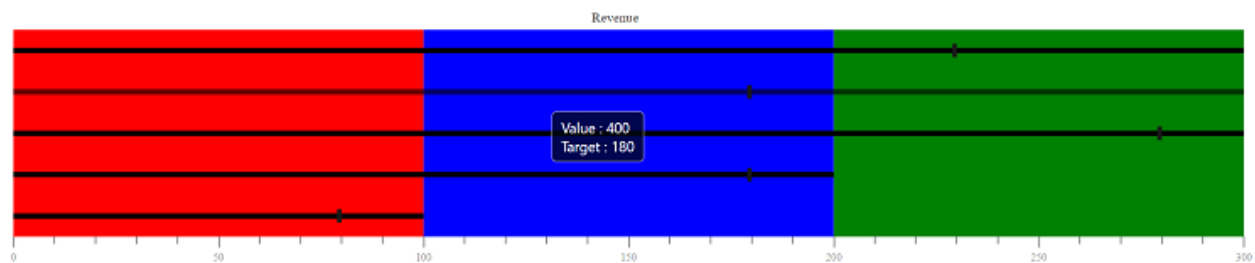
or

```
`bash
```

```
yarn run dev
```

```
,
```

The output will appear as follows:



**Sample:** [vue-3-bullet-chart-getting-started](#).

For migrating from Vue 2 to Vue 3, refer to the [migration](#) documentation.

See also

- [Getting Started with Vue UI Components using Composition API and TypeScript](#)
- [Getting Started with Vue UI Components using Options API and TypeScript](#)

## Bullet chart dimensions in Vue Bullet chart component

### Size for Container

The size of the Bullet Chart is determined by the container size, and it can be changed inline or via CSS as following.

```
,  
  
<style>  
bulletChart {  
width: 350px;  
}  
</style>  
  
<template>  
  <div>  
    <ejs-bulletchart id="bulletChart"> </ejs-bulletchart>  
  </div>  
</template>  
  
<script>  
import Vue from 'vue';  
import { BulletChartPlugin } from '@syncfusion/ej2-vue-charts';  
Vue.use(BulletChartPlugin);  
export default {  
  data() { /*  
    */  
  }  
}  
</script>  
,
```

### APP.VUE

```
<template>  
  <div>  
    <ejs-bulletchart id="bulletChart"  
      :dataSource="data"  
      valueField="value"  
      targetField="target"  
      :minimum="minimum"  
      :maximum="maximum"  
      :interval="interval"  
      title="Revenue"  
    >  
      <e-bullet-range-collection>
```

```

        <e-bullet-range end="100" color="red"></e-bullet-range>
        <e-bullet-range end="200" color="blue"></e-bullet-range>
        <e-bullet-range end="300" color="green"></e-bullet-range>
    </e-bullet-range-collection>
</ejs-bulletchart>
</div>
</template>
<script>
import Vue from 'vue';
import { BulletChartPlugin } from '@syncfusion/ej2-vue-charts';
Vue.use(BulletChartPlugin);
export default {
    data () {
        return {
            data: [{ value: 270, target: 250 }],
            minimum: 0, maximum: 300, interval: 50
        }
    }
}
</script>
<style>
    #bulletChart {
        width: 550px;
    }
</style>

```

```
{% previewsample "page.domainurl/code-snippet/bullet-chart/bullet-chart-dimensions/container-cs1" %}
```

### Size for Bullet Chart

The [width](#) and [height](#) properties are used to adjust the size of the Bullet Chart.

### APP.VUE

```

<template>
  <div>
    <ejs-bulletchart id="bulletChart"
      :dataSource="data"
      valueField="value"
      targetField="target"
      :minimum="minimum"
      :maximum="maximum"
      :interval="interval"
      title="Revenue"
      width="100%"
      height="100%"
    >
      <e-bullet-range-collection>
        <e-bullet-range end="100" color="red"></e-bullet-range>
        <e-bullet-range end="200" color="blue"></e-bullet-range>
        <e-bullet-range end="300" color="green"></e-bullet-range>
      </e-bullet-range-collection>
    </ejs-bulletchart>
  </div>
</template>
<script>

```



```
import Vue from 'vue';
import { BulletChartPlugin } from '@syncfusion/ej2-vue-charts';
Vue.use(BulletChartPlugin);
export default {
  data () {
    return {
      data: [{ value: 270, target: 250 }],
      minimum: 0, maximum: 300, interval: 50
    }
  }
}
</script>
```

```
{% previewsample "page.domainurl/code-snippet/bullet-chart/bullet-chart-dimensions/container-cs2"
%}
```

[Pixel](#)

Can set the size of the Bullet Chart in pixels as shown below.

### APP.VUE

```
<template>
  <div>
    <ejs-bulletchart id="bulletChart"
      :dataSource="data"
      valueField="value"
      targetField="target"
      :minimum="minimum"
      :maximum="maximum"
      :interval="interval"
      title="Revenue"
      width="600px"
      height="120px"
    >
      <e-bullet-range-collection>
        <e-bullet-range end="100" color="red"></e-bullet-range>
        <e-bullet-range end="200" color="blue"></e-bullet-range>
        <e-bullet-range end="300" color="green"></e-bullet-range>
      </e-bullet-range-collection>
    </ejs-bulletchart>
  </div>
</template>
<script>
import Vue from 'vue';
import { BulletChartPlugin } from '@syncfusion/ej2-vue-charts';
Vue.use(BulletChartPlugin);
export default {
  data () {
    return {
      data: [{ value: 270, target: 250 }],
      minimum: 0, maximum: 300, interval: 50
    }
  }
}
</script>
```

{% previewsample "page.domainurl/code-snippet/bullet-chart/bullet-chart-dimensions/container-cs3"%}

### Percentage

By setting a value in percentage, the Bullet Chart gets its dimension with respect to its container. For example, when the height is **50%**, the Bullet Chart renders to half of the container's height.

### APP.VUE

```
<template>
  <div>
    <ejs-bulletchart id="bulletChart"
      :dataSource="data"
      valueField="value"
      targetField="target"
      :minimum="minimum"
      :maximum="maximum"
      :interval="interval"
      title="Revenue"
      width="100%"
      height="100%"
    >
      <e-bullet-range-collection>
        <e-bullet-range end="100" color="red"></e-bullet-range>
        <e-bullet-range end="200" color="blue"></e-bullet-range>
        <e-bullet-range end="300" color="green"></e-bullet-range>
      </e-bullet-range-collection>
    </ejs-bulletchart>
  </div>
</template>
<script>
import Vue from 'vue';
import { BulletChartPlugin } from '@syncfusion/ej2-vue-charts';
Vue.use(BulletChartPlugin);
export default {
  data () {
    return {
      data: [{ value: 270, target: 250 }],
      minimum: 0, maximum: 300, interval: 50
    }
  }
}
</script>
```

{% previewsample "page.domainurl/code-snippet/bullet-chart/bullet-chart-dimensions/container-cs4"%}

If the size is not specified, the Bullet Chart will be rendered with a height of **126px** and a width of the window.

### Axis customization in Vue Bullet chart component

#### MajorTickLines and MinorTickLines Customization

The following properties can be used to customize [majorTicklines](#) and [minorTicklines](#).

- **width** - Specifies the width of ticklines.

- **height** - Specifies the height of ticklines.
- **color** - Specifies the color of ticklines.
- **useRangeColor** - Specifies the color of ticklines and represents the color from corresponding range colors.

**APP.VUE**

```

<template>
  <div>
    <ejs-bulletchart id="bulletChart"
      :dataSource="data"
      valueField="value"
      targetField="target"
      :minimum="minimum"
      :maximum="maximum"
      :interval="interval"
      title="Revenue"
      :majorTickLines="majorTickLines"
      :minorTickLines="minorTickLines"
    >
      <e-bullet-range-collection>
        <e-bullet-range end="100" color="red"></e-bullet-range>
        <e-bullet-range end="200" color="blue"></e-bullet-range>
        <e-bullet-range end="300" color="green"></e-bullet-range>
      </e-bullet-range-collection>
    </ejs-bulletchart>
  </div>
</template>
<script>
import Vue from 'vue';
import { BulletChartPlugin } from '@syncfusion/ej2-vue-charts';
Vue.use(BulletChartPlugin);
export default {
  data () {
    return {
      data: [{ value: 270, target: 250 }],
      minimum: 0, maximum: 300, interval: 50,
      majorTickLines: { color: 'blue', width: 5 },
      minorTickLines: { color: 'red', width: 5 }
    }
  }
}
</script>

```

{% previewsample "page.domainurl/code-snippet/bullet-chart/axis/custom-cs1" %}

**Tick Placement**

The major and the minor ticks can be placed **inside** or **outside** the ranges using the [tickPosition](#) property.

**APP.VUE**

```

<template>
  <div>
    <ejs-bulletchart id="bulletChart"

```

```

        :dataSource="data"
        valueField="value"
        targetField="target"
        :minimum="minimum"
        :maximum="maximum"
        :interval="interval"
        title="Revenue"
        tickPosition="Inside"
      >
      <e-bullet-range-collection>
        <e-bullet-range end="20" color="red"></e-bullet-range>
        <e-bullet-range end="25" color="blue"></e-bullet-range>
        <e-bullet-range end="30" color="green"></e-bullet-range>
      </e-bullet-range-collection>
    </ejs-bulletchart>
  </div>
</template>
<script>
import Vue from 'vue';
import { BulletChartPlugin } from '@syncfusion/ej2-vue-charts';
Vue.use(BulletChartPlugin);
export default {
  data () {
    return {
      data: [{ value: 23, target: 22 }],
      minimum: 0, maximum: 30, interval: 5
    }
  }
}
</script>

```

{% previewsample "page.domainurl/code-snippet/bullet-chart/axis/custom-cs2" %}

### Label Format

Axis numeric labels can be formatted by using the [labelFormat](#) property. Axis labels support all globalize formats.

### APP.VUE

```

<template>
  <div>
    <ejs-bulletchart id="bulletChart"
      :dataSource="data"
      valueField="value"
      targetField="target"
      :minimum="minimum"
      :maximum="maximum"
      :interval="interval"
      title="Revenue"
      labelFormat="c"
    >
    <e-bullet-range-collection>
      <e-bullet-range end="500" color="red"></e-bullet-range>
      <e-bullet-range end="1000" color="blue"></e-bullet-range>
      <e-bullet-range end="1500" color="green"></e-bullet-range>
    </e-bullet-range-collection>
  </div>
</template>

```

```

    </ejs-bulletchart>
  </div>
</template>
<script>
import Vue from 'vue';
import { BulletChartPlugin } from '@syncfusion/ej2-vue-charts';
Vue.use(BulletChartPlugin);
export default {
  data () {
    return {
      data: [{ value: 1100, target: 950 }],
      minimum: 0, maximum: 1500, interval: 500
    }
  }
}
</script>

```

{% previewsample "page.domainurl/code-snippet/bullet-chart/axis/custom-cs3" %}

The following table describes the result of applying some commonly used formats to numeric axis labels.

<!-- markdownlint-disable MD033 -->

Label Value	Label Format property value	Result	Description
1000	n1	1000.0	The Number is rounded to 1 decimal place
1000	n2	1000.00	The Number is rounded to 2 decimal place
1000	n3	1000.000	The Number is rounded to 3 decimal place
0.01	p1	1.0%	The Number is converted to percentage with 1 decimal place
0.01	p2	1.00%	The Number is converted to percentage with 2 decimal place
0.01	p3	1.000%	The Number is converted to percentage with 3 decimal place
1000	c1	\$1000.0	The Currency symbol is appended to number and number is rounded to 1 decimal place
1000	c2	\$1000.00	The Currency symbol is appended to number and number is rounded to 2 decimal place

### GroupingSeparator

To separate the groups of thousands, set the [enableGroupSeparator](#) property to **true**.

### APP.VUE

```

<template>
  <div>
    <ejs-bulletchart id="bulletChart"
      :dataSource="data"

```

```

        valueField="value"
        targetField="target"
        :minimum="minimum"
        :maximum="maximum"
        :interval="interval"
        title="Revenue"
        enableGroupSeparator=true
      >
      <e-bullet-range-collection>
        <e-bullet-range end="500" color="red"></e-bullet-range>
        <e-bullet-range end="1000" color="blue"></e-bullet-range>
        <e-bullet-range end="1500" color="green"></e-bullet-range>
      </e-bullet-range-collection>
    </ejs-bulletchart>
  </div>
</template>
<script>
import Vue from 'vue';
import { BulletChartPlugin } from '@syncfusion/ej2-vue-charts';
Vue.use(BulletChartPlugin);
export default {
  data () {
    return {
      data: [{ value: 1100, target: 950 }],
      minimum: 0, maximum: 1500, interval: 500
    }
  }
}
</script>

```

{% previewsample "page.domainurl/code-snippet/bullet-chart/axis/custom-cs4" %}

### Custom Label Format

Using the [labelFormat](#) property, axis labels can be specified with a custom defined format in addition to the axis value. The label format uses a placeholder such as `${value}K`, which represents the axis label.

### APP.VUE

```

<template>
  <div>
    <ejs-bulletchart id="bulletChart"
      :dataSource="data"
      valueField="value"
      targetField="target"
      :minimum="minimum"
      :maximum="maximum"
      :interval="interval"
      title="Revenue"
      labelFormat="${value}K"
    >
    <e-bullet-range-collection>
      <e-bullet-range end="100" color="red"></e-bullet-range>
      <e-bullet-range end="200" color="blue"></e-bullet-range>
      <e-bullet-range end="300" color="green"></e-bullet-range>
    </e-bullet-range-collection>
  </ejs-bulletchart>

```

```

    </div>
  </template>
  <script>
    import Vue from 'vue';
    import { BulletChartPlugin } from '@syncfusion/ej2-vue-charts';
    Vue.use(BulletChartPlugin);
    export default {
      data () {
        return {
          data: [{ value: 270, target: 250 }],
          minimum: 0, maximum: 300, interval: 50
        }
      }
    }
  </script>

```

{% previewsample "page.domainurl/code-snippet/bullet-chart/axis/custom-cs5" %}

### Label Placement

Label can be placed **Inside** or **Outside** of the ranges using the [labelPosition](#) property.

### APP.VUE

```

<template>
  <div>
    <ejs-bulletchart id="bulletChart"
      :dataSource="data"
      valueField="value"
      targetField="target"
      :minimum="minimum"
      :maximum="maximum"
      :interval="interval"
      title="Revenue"
      labelPosition="Inside"
    >
      <e-bullet-range-collection>
        <e-bullet-range end="100" color="red"></e-bullet-range>
        <e-bullet-range end="200" color="blue"></e-bullet-range>
        <e-bullet-range end="300" color="green"></e-bullet-range>
      </e-bullet-range-collection>
    </ejs-bulletchart>
  </div>
</template>
<script>
  import Vue from 'vue';
  import { BulletChartPlugin } from '@syncfusion/ej2-vue-charts';
  Vue.use(BulletChartPlugin);
  export default {
    data () {
      return {
        data: [{ value: 270, target: 250 }],
        minimum: 0, maximum: 300, interval: 50
      }
    }
  }
</script>

```

{% previewsample "page.domainurl/code-snippet/bullet-chart/axis/custom-cs6" %}

### Opposed Position

To place an axis opposite to its original position, set the [opposedPosition](#) property to **true**.

#### APP.VUE

```
<template>
  <div>
    <ejs-bulletchart id="bulletChart"
      :dataSource="data"
      valueField="value"
      targetField="target"
      :minimum="minimum"
      :maximum="maximum"
      :interval="interval"
      title="Revenue"
      opposedPosition=true
    >
      <e-bullet-range-collection>
        <e-bullet-range end="100" color="red"></e-bullet-range>
        <e-bullet-range end="200" color="blue"></e-bullet-range>
        <e-bullet-range end="300" color="green"></e-bullet-range>
      </e-bullet-range-collection>
    </ejs-bulletchart>
  </div>
</template>
<script>
import Vue from 'vue';
import { BulletChartPlugin } from '@syncfusion/ej2-vue-charts';
Vue.use(BulletChartPlugin);
export default {
  data () {
    return {
      data: [{ value: 270, target: 250 }],
      minimum: 0, maximum: 300, interval: 50
    }
  }
}
</script>
```

{% previewsample "page.domainurl/code-snippet/bullet-chart/axis/custom-cs7" %}

### Category Label

The Bullet Chart supports X-axis label by specifying the property from the data source to the [categoryField](#). It helps to understand the input data in a more efficient way.

#### APP.VUE

```
<template>
  <div>
    <ejs-bulletchart id="bulletChart"
      :dataSource="data"
      valueField="value"
      targetField="target"
```



```

      :minimum="minimum"
      :maximum="maximum"
      :interval="interval"
      title="Revenue"
      categoryField="category"
    >
    <e-bullet-range-collection>
      <e-bullet-range end="100" color="red"></e-bullet-range>
      <e-bullet-range end="200" color="blue"></e-bullet-range>
      <e-bullet-range end="300" color="green"></e-bullet-range>
    </e-bullet-range-collection>
  </ejs-bulletchart>
</div>
</template>
<script>
import Vue from 'vue';
import { BulletChartPlugin } from '@syncfusion/ej2-vue-charts';
Vue.use(BulletChartPlugin);
export default {
  data () {
    return {
      data: [{ value: 270, target: 250, category: "Product A" }],
      minimum: 0, maximum: 300, interval: 50
    }
  }
}
</script>

```

{% previewsample "page.domainurl/code-snippet/bullet-chart/axis/custom-cs8" %}

### Category Label Customization

The label color, opacity, font size, font family, font weight, and font style can be customized by using the [categoryLabelStyle](#) setting for category and the [labelStyle](#) setting for axis label. The [useRangeColor](#) property specifies the color of the axis label and represents the color from the corresponding range colors.

### APP.VUE

```

<template>
  <div>
    <ejs-bulletchart id="bulletChart"
      :dataSource="data"
      valueField="value"
      targetField="target"
      :minimum="minimum"
      :maximum="maximum"
      :interval="interval"
      title="Revenue"
      categoryField="category"
      :categoryLabelStyle="categoryStyle"
    >
    <e-bullet-range-collection>
      <e-bullet-range end="100" color="red"></e-bullet-range>
      <e-bullet-range end="200" color="blue"></e-bullet-range>
      <e-bullet-range end="300" color="green"></e-bullet-range>
    </e-bullet-range-collection>
  </div>
</template>

```

```

    </ejs-bulletchart>
  </div>
</template>
<script>
import Vue from 'vue';
import { BulletChartPlugin } from '@syncfusion/ej2-vue-charts';
Vue.use(BulletChartPlugin);
export default {
  data () {
    return {
      data: [{ value: 270, target: 250, category: "Product A" }],
      minimum: 0, maximum: 300, interval: 50,
      categoryStyle: { color: "Orange" }
    }
  }
}
</script>

```

{% previewsample "page.domainurl/code-snippet/bullet-chart/axis/custom-cs9" %}

### Data binding in Vue Bullet chart component

The [dataSource](#) property accepts a collection of values as input that helps to display measures, and compares them to a target bar. To display the actual and target bar, specify the property from the datasource into the [valueField](#) and [targetField](#) respectively.

#### APP.VUE

```

<template>
  <div>
    <ejs-bulletchart id="bulletChart"
      :dataSource="localData"
      valueField="value"
      targetField="comparativeMeasureValue"
      :minimum="minimum"
      :maximum="maximum"
      :interval="interval"
      title="Profit in %"
    >
      <e-bullet-range-collection>
        <e-bullet-range end="5" color="red"></e-bullet-range>
        <e-bullet-range end="15" color="blue"></e-bullet-range>
        <e-bullet-range end="20" color="green"></e-bullet-range>
      </e-bullet-range-collection>
    </ejs-bulletchart>
  </div>
</template>
<script>
import Vue from 'vue';
import { BulletChartPlugin } from '@syncfusion/ej2-vue-charts';
Vue.use(BulletChartPlugin);
export default {
  data () {
    return {
      localData: [{ value: 5, comparativeMeasureValue: 7.5, category:
'2001' },
                  { value: 7, comparativeMeasureValue: 5, category: '2002' },

```

```

        { value: 10, comparativeMeasureValue: 6, category: '2003' },
        { value: 5, comparativeMeasureValue: 8, category: '2004' },
        { value: 12, comparativeMeasureValue: 5, category: '2005' },
        { value: 8, comparativeMeasureValue: 6, category: '2006' }
      ],
      minimum: 0, maximum: 20, interval: 5
    }
  }
}
</script>

```

```

{% previewsample "page.domainurl/code-snippet/bullet-chart/bullet-chart-dimensions/container-cs12"
%}

```

### Ranges in Vue Bullet chart component

Ranges represent the quality of a specific range such as **Good**, **Bad** and **Satisfactory** in the Bullet Chart scale. The ending point of a qualitative range is specified in the [end](#) property in [ranges](#). The [minimum](#) value of a quantitative scale is considered the starting point of the first range or the previous range end point.

#### APP.VUE

```

<template>
  <div>
    <ejs-bulletchart id="bulletChart"
      :dataSource="data"
      valueField="value"
      targetField="target"
      :minimum="minimum"
      :maximum="maximum"
      :interval="interval"
      title="Revenue"
      categoryField="category"
      :categoryLabelStyle="categoryStyle"
      height="400px"
    >
      <e-bullet-range-collection>
        <e-bullet-range end="35" color="darkred" opacity= 0.5 ></e-bullet-
range>
        <e-bullet-range end="50" color="red" opacity= 1 ></e-bullet-range>
        <e-bullet-range end="75" color="blue" opacity= 0.7 ></e-bullet-
range>
        <e-bullet-range end="90" color="lightgreen" opacity= 1 ></e-
bullet-range>
        <e-bullet-range end="100" color="green" opacity= 1 ></e-bullet-
range>
      </e-bullet-range-collection>
    </ejs-bulletchart>
  </div>
</template>
<script>
import Vue from 'vue';
import { BulletChartPlugin } from '@syncfusion/ej2-vue-charts';
Vue.use(BulletChartPlugin);
export default {
  data () {

```

```

    return {
      data: [{ value: 55, target: 75, category: "Year 1" },
        { value: 70, target: 70, category: "Year 2" },
        { value: 85, target: 75, category: "Year 3" }],
      minimum: 0, maximum: 100, interval: 10,
      categoryStyle: { color: "red", size: "13", fontWeight: "bold" }
    }
  }
}
</script>

```

{% previewsample "page.domainurl/code-snippet/bullet-chart/axis/custom-cs10" %}

### Color Customization

Enhance the readability of ranges with color and opacity. It can be applied using the [color](#) and [opacity](#) properties in [ranges](#).

### APP.VUE

```

<template>
  <div>
    <ejs-bulletchart id="bulletChart"
      :dataSource="data"
      valueField="value"
      targetField="target"
      :minimum="minimum"
      :maximum="maximum"
      :interval="interval"
      title="Revenue"
      categoryField="category"
      :categoryLabelStyle="categoryStyle"
      height="400px"
    >
      <e-bullet-range-collection>
        <e-bullet-range end="35" color="darkred" opacity= 0.5 ></e-bullet-
range>
        <e-bullet-range end="50" color="red" opacity= 1 ></e-bullet-range>
        <e-bullet-range end="75" color="blue" opacity= 0.7 ></e-bullet-
range>
        <e-bullet-range end="90" color="lightgreen" opacity= 1 ></e-
bullet-range>
        <e-bullet-range end="100" color="green" opacity= 1 ></e-bullet-
range>
      </e-bullet-range-collection>
    </ejs-bulletchart>
  </div>
</template>
<script>
import Vue from 'vue';
import { BulletChartPlugin } from '@syncfusion/ej2-vue-charts';
Vue.use(BulletChartPlugin);
export default {
  data () {
    return {
      data: [{ value: 55, target: 75, category: "Year 1" },
        { value: 70, target: 70, category: "Year 2" },

```

```

        { value: 85, target: 75, category: "Year 3" }],
        minimum: 0, maximum: 100, interval: 10,
        categoryStyle: { color: "red", size: "13", fontWeight: "bold" }
    }
}
}
</script>

```

{% previewsample "page.domainurl/code-snippet/bullet-chart/axis/custom-cs11" %}

### Value bar in Vue Bullet chart component

To display the primary data or the current value of the data being measured known as the **Feature Measure** that should be encoded as a bar. This is called as the **Actual Bar** or the **Feature Bar** in the Bullet Chart, and to display the actual bar the [valueField](#) should be mapped to the appropriate field from the data source.

### APP.VUE

```

<template>
  <div>
    <ejs-bulletchart id="bulletChart"
      :dataSource="data"
      valueField="value"
      targetField="target"
      :minimum="minimum"
      :maximum="maximum"
      :interval="interval"
      title="Sales Rate"
      categoryField="category"
    >
      <e-bullet-range-collection>
        <e-bullet-range end="35" color="red"></e-bullet-range>
        <e-bullet-range end="50" color="blue"></e-bullet-range>
        <e-bullet-range end="100" color="green"></e-bullet-range>
      </e-bullet-range-collection>
    </ejs-bulletchart>
  </div>
</template>
<script>
import Vue from 'vue';
import { BulletChartPlugin } from '@syncfusion/ej2-vue-charts';
Vue.use(BulletChartPlugin);
export default {
  data () {
    return {
      data: [{ value: 55, target: 75, category: "Year 1" }],
      minimum: 0, maximum: 100, interval: 20
    }
  }
}
</script>

```

{% previewsample "page.domainurl/code-snippet/bullet-chart/bullet-chart-dimensions/container-cs24" %}

### Types of actual bar

The shape of the actual bar can be customized using the [type](#) property of the Bullet Chart. The actual bar contains **Rect** and **Dot** shapes. By default, the actual bar shape is Rect.

#### APP.VUE

```
<template>
  <div>
    <ejs-bulletchart id="bulletChart"
      :dataSource="data"
      valueField="value"
      targetField="target"
      :minimum="minimum"
      :maximum="maximum"
      :interval="interval"
      title="Sales Rate"
      categoryField="category"
      type="Dot"
    >
    <e-bullet-range-collection>
      <e-bullet-range end="35" color="red"></e-bullet-range>
      <e-bullet-range end="50" color="blue"></e-bullet-range>
      <e-bullet-range end="100" color="green"></e-bullet-range>
    </e-bullet-range-collection>
    </ejs-bulletchart>
  </div>
</template>
<script>
import Vue from 'vue';
import { BulletChartPlugin } from '@syncfusion/ej2-vue-charts';
Vue.use(BulletChartPlugin);
export default {
  data () {
    return {
      data: [{ value: 55, target: 75, category: "Year 1" }],
      minimum: 0, maximum: 100, interval: 20
    }
  }
}
</script>
```

{% previewsample "page.domainurl/code-snippet/bullet-chart/bullet-chart-dimensions/container-cs25" %}

### Actual bar customization

#### Border customization

Using the [valueBorder](#) property of the bullet chart, you can customize the border [color](#) and [width](#) of the actual bar.

#### APP.VUE

```
<template>
  <div>
    <ejs-bulletchart id="bulletChart"
      :dataSource="data"
      valueField="value"
```

```

        targetField="target"
        :minimum="minimum"
        :maximum="maximum"
        :interval="interval"
        title="Sales Rate"
        categoryField="category"
        :valueBorder="valueBorder"
      >
      <e-bullet-range-collection>
        <e-bullet-range end="35" color="red"></e-bullet-range>
        <e-bullet-range end="50" color="blue"></e-bullet-range>
        <e-bullet-range end="100" color="green"></e-bullet-range>
      </e-bullet-range-collection>
    </ejs-bulletchart>
  </div>
</template>
<script>
import Vue from 'vue';
import { BulletChartPlugin } from '@syncfusion/ej2-vue-charts';
Vue.use(BulletChartPlugin);
export default {
  data () {
    return {
      data: [{ value: 55, target: 75, category: "Year 1" }],
      minimum: 0, maximum: 100, interval: 20, valueBorder: {
        color: "red", width: 3
      }
    }
  }
}
</script>

```

{% previewsample "page.domainurl/code-snippet/bullet-chart/bullet-chart-dimensions/container-cs26" %}

#### Fill color and height customization

Customize the fill color and height of the actual bar using the [valueFill](#) and [valueHeight](#) properties of the bullet chart. Also, you can bind the color for the actual bar from [dataSource](#) for the bullet chart using [valueFill](#) property.

#### APP.VUE

```

<template>
  <div>
    <ejs-bulletchart id="bulletChart"
      :dataSource="data"
      valueField="value"
      targetField="target"
      :minimum="minimum"
      :maximum="maximum"
      :interval="interval"
      title="Sales Rate"
      categoryField="category"
      valueFill="color"
    >
    <e-bullet-range-collection>

```

```

        <e-bullet-range end="35" color="red"></e-bullet-range>
        <e-bullet-range end="50" color="blue"></e-bullet-range>
        <e-bullet-range end="100" color="green"></e-bullet-range>
      </e-bullet-range-collection>
    </ejs-bulletchart>
  </div>
</template>
<script>
import Vue from 'vue';
import { BulletChartPlugin } from '@syncfusion/ej2-vue-charts';
Vue.use(BulletChartPlugin);
export default {
  data () {
    return {
      data: [{ value: 55, target: 75, category: "Year 1", color: "blue" }],
      minimum: 0, maximum: 100, interval: 20
    }
  }
}
</script>

```

{% previewsample "page.domainurl/code-snippet/bullet-chart/bullet-chart-dimensions/container-cs27"  
 %}

### Comparative bar in Vue Bullet chart component

The line marker that runs perpendicular to the orientation of the graph is known as the **Comparative Measure** and it is used as a target marker to compare against the feature measure value. This is also called as the **Target Bar** in the Bullet Chart. To display the target bar, the [targetField](#) should be mapped to the appropriate field from the datasource.

#### APP.VUE

```

<template>
  <div>
    <ejs-bulletchart id="bulletChart"
      :dataSource="data"
      valueField="value"
      targetField="target"
      :minimum="minimum"
      :maximum="maximum"
      :interval="interval"
      title="Sales Rate"
    >
      <e-bullet-range-collection>
        <e-bullet-range end="35" color="red"></e-bullet-range>
        <e-bullet-range end="50" color="blue"></e-bullet-range>
        <e-bullet-range end="100" color="green"></e-bullet-range>
      </e-bullet-range-collection>
    </ejs-bulletchart>
  </div>
</template>
<script>
import Vue from 'vue';
import { BulletChartPlugin } from '@syncfusion/ej2-vue-charts';
Vue.use(BulletChartPlugin);
export default {

```



```

data () {
  return {
    data: [{ value: 55, target: 75 }],
    minimum: 0, maximum: 100, interval: 20
  }
}
}
</script>

```

```

{% previewsample "page.domainurl/code-snippet/bullet-chart/bullet-chart-dimensions/container-cs5"
%}

```

### Types of target bar

The shape of the target bar can be customized using the [targetTypes](#) property and it supports **Circle**, **Cross**, and **Rect** shapes. The default type of the target bar is **Rect**.

### APP.VUE

```

<template>
  <div>
    <ejs-bulletchart id="bulletChart"
      :dataSource="data"
      valueField="value"
      targetField="target"
      :minimum="minimum"
      :maximum="maximum"
      :interval="interval"
      title="Sales Rate"
      :targetTypes="targetTypes"
      :animation="animation"
    >
      <e-bullet-range-collection>
        <e-bullet-range end="35" color="red"></e-bullet-range>
        <e-bullet-range end="50" color="blue"></e-bullet-range>
        <e-bullet-range end="100" color="green"></e-bullet-range>
      </e-bullet-range-collection>
    </ejs-bulletchart>
  </div>
</template>
<script>
import Vue from 'vue';
import { BulletChartPlugin } from '@syncfusion/ej2-vue-charts';
Vue.use(BulletChartPlugin);
export default {
  data () {
    return {
      data: [{ value: 55, target: 75 }],
      minimum: 0, maximum: 100, interval: 20, targetTypes: ["Circle"],
      animation: { enable: false }
    }
  }
}
</script>

```

```
{% previewsample "page.domainurl/code-snippet/bullet-chart/bullet-chart-dimensions/container-cs6"
%}
```

### Target bar customization

The following properties can be used to customize the target bar. Also, you can bind the color for the target bar from [dataSource](#) for the bullet chart.

- [targetColor](#) - Specifies the fill color of target bar.
- [targetWidth](#) - Specifies the width of target bar.

### APP.VUE

```
<template>
  <div>
    <ejs-bulletchart id="bulletChart"
      :dataSource="data"
      valueField="value"
      targetField="target"
      :minimum="minimum"
      :maximum="maximum"
      :interval="interval"
      title="Sales Rate"
      targetColor="color"
      targetWidth=15
    >
      <e-bullet-range-collection>
        <e-bullet-range end="35" color="red"></e-bullet-range>
        <e-bullet-range end="50" color="blue"></e-bullet-range>
        <e-bullet-range end="100" color="green"></e-bullet-range>
      </e-bullet-range-collection>
    </ejs-bulletchart>
  </div>
</template>
<script>
import Vue from 'vue';
import { BulletChartPlugin } from '@syncfusion/ej2-vue-charts';
Vue.use(BulletChartPlugin);
export default {
  data () {
    return {
      data: [{ value: 55, target: 75, color: 'red' }],
      minimum: 0, maximum: 100, interval: 20
    }
  }
}
</script>
```

```
{% previewsample "page.domainurl/code-snippet/bullet-chart/bullet-chart-dimensions/container-cs7"
%}
```

### Title in Vue Bullet chart component

#### Title

The title of the Bullet Chart displays the information about the data plotted by specifying it in the [title](#) property.

**APP.VUE**

```

<template>
  <div>
    <ejs-bulletchart id="bulletChart"
      :dataSource="data"
      valueField="value"
      targetField="target"
      :minimum="minimum"
      :maximum="maximum"
      :interval="interval"
      title="Sales Rate"
    >
      <e-bullet-range-collection>
        <e-bullet-range end="35" color="red"></e-bullet-range>
        <e-bullet-range end="50" color="blue"></e-bullet-range>
        <e-bullet-range end="100" color="green"></e-bullet-range>
      </e-bullet-range-collection>
    </ejs-bulletchart>
  </div>
</template>
<script>
import Vue from 'vue';
import { BulletChartPlugin } from '@syncfusion/ej2-vue-charts';
Vue.use(BulletChartPlugin);
export default {
  data () {
    return {
      data: [{ value: 55, target: 75 }],
      minimum: 0, maximum: 100, interval: 20
    }
  }
}
</script>

```

```
{% previewsample "page.domainurl/code-snippet/bullet-chart/bullet-chart-dimensions/container-cs15"
%}
```

**Subtitle**

To show additional information about the data plotted, the Bullet Chart can also be given a subtitle using the [subtitle](#) property.

**APP.VUE**

```

<template>
  <div>
    <ejs-bulletchart id="bulletChart"
      :dataSource="data"
      valueField="value"
      targetField="target"
      :minimum="minimum"
      :maximum="maximum"
      :interval="interval"
      title="Sales Rate in dollars"
      labelFormat="{value}"
      subtitle="(in dollars $)"
    >

```

```

    >
    <e-bullet-range-collection>
      <e-bullet-range end="35" color="red"></e-bullet-range>
      <e-bullet-range end="50" color="blue"></e-bullet-range>
      <e-bullet-range end="100" color="green"></e-bullet-range>
    </e-bullet-range-collection>
  </ejs-bulletchart>
</div>
</template>
<script>
import Vue from 'vue';
import { BulletChartPlugin } from '@syncfusion/ej2-vue-charts';
Vue.use(BulletChartPlugin);
export default {
  data () {
    return {
      data: [{ value: 55, target: 45 }],
      minimum: 0, maximum: 100, interval: 20
    }
  }
}
</script>

```

```
{% previewsample "page.domainurl/code-snippet/bullet-chart/bullet-chart-dimensions/container-cs16" %}
```

### Title and SubTitle Position

The title and the subtitle positions can be customized using the [titlePosition](#) property. Possible positions are **Left**, **Right**, **Top**, and **Bottom**.

#### Position as Left

By setting the [titlePosition](#) to **Left**, you can display the title and subtitle at the left side of the Bullet Chart.

### APP.VUE

```

<template>
  <div>
    <ejs-bulletchart id="bulletChart"
      :dataSource="data"
      valueField="value"
      targetField="target"
      :minimum="minimum"
      :maximum="maximum"
      :interval="interval"
      title="Sales Rate in dollars"
      titlePosition="Left"
      labelFormat="{value}"
      subtitle="(in dollars $)"
    >
    <e-bullet-range-collection>
      <e-bullet-range end="35" color="red"></e-bullet-range>
      <e-bullet-range end="50" color="blue"></e-bullet-range>
      <e-bullet-range end="100" color="green"></e-bullet-range>
    </e-bullet-range-collection>
  </ejs-bulletchart>

```

```

</div>
</template>
<script>
import Vue from 'vue';
import { BulletChartPlugin } from '@syncfusion/ej2-vue-charts';
Vue.use(BulletChartPlugin);
export default {
  data () {
    return {
      data: [{ value: 55, target: 45 }],
      minimum: 0, maximum: 100, interval: 20
    }
  }
}
</script>

```

{% previewsample "page.domainurl/code-snippet/bullet-chart/bullet-chart-dimensions/container-cs17"  
 %}

### Position as Right

By setting the [titlePosition](#) to **Right**, you can display the title and subtitle at the right side of the Bullet Chart.

### APP.VUE

```

<template>
  <div>
    <ejs-bulletchart id="bulletChart"
      :dataSource="data"
      valueField="value"
      targetField="target"
      :minimum="minimum"
      :maximum="maximum"
      :interval="interval"
      title="Sales Rate in dollars"
      titlePosition="Right"
      labelFormat="{value}"
      subtitle="(in dollars $)"
    >
      <e-bullet-range-collection>
        <e-bullet-range end="35" color="red"></e-bullet-range>
        <e-bullet-range end="50" color="blue"></e-bullet-range>
        <e-bullet-range end="100" color="green"></e-bullet-range>
      </e-bullet-range-collection>
    </ejs-bulletchart>
  </div>
</template>
<script>
import Vue from 'vue';
import { BulletChartPlugin } from '@syncfusion/ej2-vue-charts';
Vue.use(BulletChartPlugin);
export default {
  data () {
    return {
      data: [{ value: 55, target: 45 }],
      minimum: 0, maximum: 100, interval: 20
    }
  }
}

```

```

    }
  }
}
</script>

```

{% previewsample "page.domainurl/code-snippet/bullet-chart/bullet-chart-dimensions/container-cs18" %}

### Position as Top

By setting the [titlePosition](#) to **Top**, you can display the title and subtitle at the top of the Bullet Chart. The default title and subtitle positions of the Bullet Chart is **Top**.

### APP.VUE

```

<template>
  <div>
    <ejs-bulletchart id="bulletChart"
      :dataSource="data"
      valueField="value"
      targetField="target"
      :minimum="minimum"
      :maximum="maximum"
      :interval="interval"
      title="Sales Rate in dollars"
      titlePosition="Top"
      labelFormat="{value}"
      subtitle="(in dollars $)"
    >
      <e-bullet-range-collection>
        <e-bullet-range end="35" color="red"></e-bullet-range>
        <e-bullet-range end="50" color="blue"></e-bullet-range>
        <e-bullet-range end="100" color="green"></e-bullet-range>
      </e-bullet-range-collection>
    </ejs-bulletchart>
  </div>
</template>
<script>
import Vue from 'vue';
import { BulletChartPlugin } from '@syncfusion/ej2-vue-charts';
Vue.use(BulletChartPlugin);
export default {
  data () {
    return {
      data: [{ value: 55, target: 45 }],
      minimum: 0, maximum: 100, interval: 20
    }
  }
}
</script>

```

{% previewsample "page.domainurl/code-snippet/bullet-chart/bullet-chart-dimensions/container-cs19" %}

*Position as Bottom*

By setting the [titlePosition](#) to **Bottom**, you can display the title and subtitle at the bottom of the Bullet Chart.

**APP.VUE**

```
<template>
  <div>
    <ejs-bulletchart id="bulletChart"
      :dataSource="data"
      valueField="value"
      targetField="target"
      :minimum="minimum"
      :maximum="maximum"
      :interval="interval"
      title="Sales Rate in dollars"
      titlePosition="Bottom"
      labelFormat="{value}"
      subtitle="(in dollars $)"
    >
      <e-bullet-range-collection>
        <e-bullet-range end="35" color="red"></e-bullet-range>
        <e-bullet-range end="50" color="blue"></e-bullet-range>
        <e-bullet-range end="100" color="green"></e-bullet-range>
      </e-bullet-range-collection>
    </ejs-bulletchart>
  </div>
</template>
<script>
import Vue from 'vue';
import { BulletChartPlugin } from '@syncfusion/ej2-vue-charts';
Vue.use(BulletChartPlugin);
export default {
  data () {
    return {
      data: [{ value: 55, target: 45 }],
      minimum: 0, maximum: 100, interval: 20
    }
  }
}
</script>
```

```
{% previewsample "page.domainurl/code-snippet/bullet-chart/bullet-chart-dimensions/container-cs20"
%}
```

*Title Customization*

The title color, opacity, font size, font family, font weight, and font style can be customized using the [titleStyle](#) property.

**APP.VUE**

```
<template>
  <div>
    <ejs-bulletchart id="bulletChart"
      :dataSource="data"
      valueField="value"
```

```

        targetField="target"
        :minimum="minimum"
        :maximum="maximum"
        :interval="interval"
        title="Sales Rate in dollars"
        labelFormat="{value}"
        :titleStyle="titleStyle"
        subtitle="(in dollars $)"
      >
      <e-bullet-range-collection>
        <e-bullet-range end="35" color="red"></e-bullet-range>
        <e-bullet-range end="50" color="blue"></e-bullet-range>
        <e-bullet-range end="100" color="green"></e-bullet-range>
      </e-bullet-range-collection>
    </ejs-bulletchart>
  </div>
</template>
<script>
import Vue from 'vue';
import { BulletChartPlugin } from '@syncfusion/ej2-vue-charts';
Vue.use(BulletChartPlugin);
export default {
  data () {
    return {
      data: [{ value: 55, target: 45 }],
      minimum: 0, maximum: 100, interval: 20,
      titleStyle: { size: '22px', color: 'red', fontFamily: 'cursive',
fontWeight: 'Bold' }
    }
  }
}
</script>

```

{% previewsample "page.domainurl/code-snippet/bullet-chart/bullet-chart-dimensions/container-cs21"  
 %}

### SubTitle Customization

The sub-title color, opacity, font size, font family, font weight, and font style can be customized using the [subtitleStyle](#) property.

### APP.VUE

```

<template>
  <div>
    <ejs-bulletchart id="bulletChart"
      :dataSource="data"
      valueField="value"
      targetField="target"
      :minimum="minimum"
      :maximum="maximum"
      :interval="interval"
      title="Sales Rate in dollars"
      labelFormat="{value}"
      subtitle="(in dollars $)"
      :subtitleStyle="subtitleStyle"
    >

```



```

    <e-bullet-range-collection>
      <e-bullet-range end="35" color="red"></e-bullet-range>
      <e-bullet-range end="50" color="blue"></e-bullet-range>
      <e-bullet-range end="100" color="green"></e-bullet-range>
    </e-bullet-range-collection>
  </ejs-bulletchart>
</div>
</template>
<script>
import Vue from 'vue';
import { BulletChartPlugin } from '@syncfusion/ej2-vue-charts';
Vue.use(BulletChartPlugin);
export default {
  data () {
    return {
      data: [{ value: 55, target: 45 }],
      minimum: 0, maximum: 100, interval: 20,
      subtitleStyle: { size: '22px', color: 'red', fontFamily: 'cursive',
fontWeight: 'Bold' }
    }
  }
}
</script>

```

```
{% previewsample "page.domainurl/code-snippet/bullet-chart/bullet-chart-dimensions/container-cs22" %}
```

## Customization in Vue Bullet chart component

### Orientation

The Bullet Chart can be rendered in different orientations such as **Horizontal** or **Vertical** via the [orientation](#) property. By default, the Bullet Chart is rendered in the **Horizontal** orientation.

### APP.VUE

```

<template>
  <div>
    <ejs-bulletchart id="bulletChart"
      :dataSource="data"
      valueField="value"
      targetField="target"
      :minimum="minimum"
      :maximum="maximum"
      :interval="interval"
      title="Sales Rate in dollars"
      labelFormat="{value}"
      subtitle="(in dollars $)"
      orientation="vertical"
      :animation="animation"
      width="20%"
    >
    <e-bullet-range-collection>
      <e-bullet-range end="35" color="red"></e-bullet-range>
      <e-bullet-range end="50" color="blue"></e-bullet-range>
      <e-bullet-range end="100" color="green"></e-bullet-range>
    </e-bullet-range-collection>
  </div>
</template>

```

```

    </ejs-bulletchart>
  </div>
</template>
<script>
import Vue from 'vue';
import { BulletChartPlugin } from '@syncfusion/ej2-vue-charts';
Vue.use(BulletChartPlugin);
export default {
  data () {
    return {
      data: [{ value: 55, target: 45 }], label: '${value}',
      minimum: 0, maximum: 100, interval: 20, animation: { enable: false }
    }
  }
}
</script>

```

{% previewsample "page.domainurl/code-snippet/bullet-chart/bullet-chart-dimensions/container-cs8"  
 %}

### Right-to-left (RTL)

The Bullet Chart supports the right-to-left rendering that can be enabled by setting the [enableRtl](#) property to **true**.

### APP.VUE

```

<template>
  <div>
    <ejs-bulletchart id="bulletChart"
      :dataSource="data"
      valueField="value"
      targetField="target"
      :minimum="minimum"
      :maximum="maximum"
      :interval="interval"
      title="Sales Rate"
      enableRtl="true"
    >
      <e-bullet-range-collection>
        <e-bullet-range end="500" color="red"></e-bullet-range>
        <e-bullet-range end="1500" color="blue"></e-bullet-range>
        <e-bullet-range end="2000" color="green"></e-bullet-range>
      </e-bullet-range-collection>
    </ejs-bulletchart>
  </div>
</template>
<script>
import Vue from 'vue';
import { BulletChartPlugin } from '@syncfusion/ej2-vue-charts';
Vue.use(BulletChartPlugin);
export default {
  data () {
    return {
      data: [{ value: 1500, target: 1000 }],
      minimum: 0, maximum: 2000, interval: 200
    }
  }
}

```

```

    }
  }
</script>

```

```
{% previewsample "page.domainurl/code-snippet/bullet-chart/bullet-chart-dimensions/container-cs9"
%}
```

### Animation

The actual and the target bar supports the linear animation via the [animation](#) setting. The speed and the delay are controlled using the `duration` and `delay` properties respectively.

### APP.VUE

```

<template>
  <div>
    <ejs-bulletchart id="bulletChart"
      :dataSource="data"
      valueField="value"
      targetField="target"
      :minimum="minimum"
      :maximum="maximum"
      :interval="interval"
      title="Sales Rate"
      :animation="animation"
    >
    <e-bullet-range-collection>
      <e-bullet-range end="500" color="red"></e-bullet-range>
      <e-bullet-range end="1500" color="blue"></e-bullet-range>
      <e-bullet-range end="2000" color="green"></e-bullet-range>
    </e-bullet-range-collection>
    </ejs-bulletchart>
  </div>
</template>
<script>
import Vue from 'vue';
import { BulletChartPlugin } from '@syncfusion/ej2-vue-charts';
Vue.use(BulletChartPlugin);
export default {
  data () {
    return {
      data: [{ value: 1500, target: 1000 }],
      minimum: 0, maximum: 2000, interval: 200,
      animation: { enable: true }
    }
  }
}
</script>

```

```
{% previewsample "page.domainurl/code-snippet/bullet-chart/bullet-chart-dimensions/container-cs10"
%}
```

### Theme

The Bullet Chart supports different type of themes via the [theme](#) property.

### APP.VUE

```

<template>
  <div>
    <ejs-bulletchart id="bulletChart"
      :dataSource="data"
      valueField="value"
      targetField="target"
      :minimum="minimum"
      :maximum="maximum"
      :interval="interval"
      title="Profit in %"
      :theme="theme"
    >
    <e-bullet-range-collection>
      <e-bullet-range end="15" color="red"></e-bullet-range>
      <e-bullet-range end="50" color="blue"></e-bullet-range>
      <e-bullet-range end="100" color="green"></e-bullet-range>
    </e-bullet-range-collection>
    </ejs-bulletchart>
  </div>
</template>
<script>
import Vue from 'vue';
import { BulletChartPlugin } from '@syncfusion/ej2-vue-charts';
Vue.use(BulletChartPlugin);
export default {
  data () {
    return {
      data: [{ value: 50, target: 45 }],
      minimum: 0, maximum: 100, interval: 10, theme: "HighContrast"
    }
  }
}
</script>

```

{% previewsample "page.domainurl/code-snippet/bullet-chart/bullet-chart-dimensions/container-cs11"%}

### Data label in Vue Bullet chart component

Data Labels are used to identify the value of actual bar in the Bullet Chart component. The Data Labels will be shown by specifying the [dataLabel](#) setting's [enable](#) property to **true**.

#### APP.VUE

```

<template>
  <div>
    <ejs-bulletchart id="bulletChart"
      :dataSource="data"
      valueField="value"
      targetField="comparativeMeasureValue"
      :minimum="minimum"
      :maximum="maximum"
      :interval="interval"
      title="Profit in %"
      labelFormat="{value}%"
      :dataLabel="dataLabel"
      height="400px"
    >

```

```

    >
    <e-bullet-range-collection>
      <e-bullet-range end="5" color="red"></e-bullet-range>
      <e-bullet-range end="15" color="blue"></e-bullet-range>
      <e-bullet-range end="20" color="green"></e-bullet-range>
    </e-bullet-range-collection>
  </ejs-bulletchart>
</div>
</template>
<script>
import Vue from 'vue';
import { BulletChartPlugin } from '@syncfusion/ej2-vue-charts';
Vue.use(BulletChartPlugin);
export default {
  data () {
    return {
      data: [{ value: 5, comparativeMeasureValue: 7.5, category: '2001' },
        { value: 7, comparativeMeasureValue: 5, category: '2002' },
        { value: 10, comparativeMeasureValue: 6, category: '2003' },
        { value: 5, comparativeMeasureValue: 8, category: '2004' },
        { value: 12, comparativeMeasureValue: 5, category: '2005' },
        { value: 8, comparativeMeasureValue: 6, category: '2006' }
      ],
      minimum: 0, maximum: 20, interval: 5, dataLabel: { visible: true }
    }
  }
}
</script>

```

```
{% previewsample "page.domainurl/code-snippet/bullet-chart/bullet-chart-dimensions/container-cs13"
%}
```

### Data Label Customization

Data Labels color, opacity, font size, font family, font weight, and font style can be customized using the [labelStyle](#).

### APP.VUE

```

<template>
  <div>
    <ejs-bulletchart id="bulletChart"
      :dataSource="data"
      valueField="value"
      targetField="comparativeMeasureValue"
      :minimum="minimum"
      :maximum="maximum"
      :interval="interval"
      title="Profit in %"
      labelFormat="{value}%"
      :dataLabel="dataLabel"
      height="400px"
    >
    <e-bullet-range-collection>
      <e-bullet-range end="5" color="red"></e-bullet-range>
      <e-bullet-range end="15" color="blue"></e-bullet-range>
      <e-bullet-range end="20" color="green"></e-bullet-range>
    </e-bullet-range-collection>
  </div>
</template>

```

```

        </e-bullet-range-collection>
      </ejs-bulletchart>
    </div>
  </template>
  <script>
import Vue from 'vue';
import { BulletChartPlugin } from '@syncfusion/ej2-vue-charts';
Vue.use(BulletChartPlugin);
export default {
  data () {
    return {
      data: [{ value: 5, comparativeMeasureValue: 7.5, category: '2001' },
        { value: 7, comparativeMeasureValue: 5, category: '2002' },
        { value: 10, comparativeMeasureValue: 6, category: '2003' },
        { value: 5, comparativeMeasureValue: 8, category: '2004' },
        { value: 12, comparativeMeasureValue: 5, category: '2005' },
        { value: 8, comparativeMeasureValue: 6, category: '2006' }
      ],
      minimum: 0, maximum: 20, interval: 5,
      dataLabel: { enable: true, labelStyle: { color: "yellow", size: "20" }
    }
  }
}
  </script>

```

```
{% previewsample "page.domainurl/code-snippet/bullet-chart/bullet-chart-dimensions/container-cs14" %}
```

### Tool tip in Vue Bullet chart component

When the mouse is hovered over a bar in the Bullet Chart, the tooltip displays important summary about the actual and the target bar values.

#### Default Tooltip

The tooltip is not visible by default. To make it visible, set the [enable](#) property in the [tooltip](#) to **true** and injecting **BulletTooltip** module using **BulletChart.Inject(BulletTooltip)** and injecting **BulletTooltip** module using **BulletChart.Inject(BulletTooltip)**.

#### APP.VUE

```

<template>
  <div>
    <ejs-bulletchart id="bulletChart"
      :dataSource="data"
      valueField="value"
      targetField="comparativeMeasureValue"
      :minimum="minimum"
      :maximum="maximum"
      :interval="interval"
      title="Profit in %"
      height="400px"
    >
    <e-bullet-range-collection>
      <e-bullet-range end="5" color="red"></e-bullet-range>
      <e-bullet-range end="15" color="blue"></e-bullet-range>
    </e-bullet-range-collection>
  </div>
</template>

```

```

        <e-bullet-range end="20" color="green"></e-bullet-range>
      </e-bullet-range-collection>
    </ejs-bulletchart>
  </div>
</template>
<script>
import Vue from 'vue';
import { BulletChartPlugin } from '@syncfusion/ej2-vue-charts';
Vue.use(BulletChartPlugin);
export default {
  data () {
    return {
      data: [{ value: 5, comparativeMeasureValue: 7.5, category: '2001' },
        { value: 7, comparativeMeasureValue: 5, category: '2002' },
        { value: 10, comparativeMeasureValue: 6, category: '2003' },
        { value: 5, comparativeMeasureValue: 8, category: '2004' },
        { value: 12, comparativeMeasureValue: 5, category: '2005' },
        { value: 8, comparativeMeasureValue: 6, category: '2006' }
      ],
      minimum: 0, maximum: 20, interval: 5
    }
  }
}
</script>

```

```
{% previewsample "page.domainurl/code-snippet/bullet-chart/bullet-chart-dimensions/container-cs23"
%}
```

### Tooltip Template

Any HTML elements can be displayed in the tooltip by using the [template](#) property of the [tooltip](#). You can use the `#{target}` and `#{value}` as place holders in the HTML element to display the value and target values from the data source of corresponding data point.

### Customize the Appearance of Tooltip

#### Tooltip Customization

The following properties can be used to customize the Bullet Chart tooltip.

- [fill](#) - Specifies the color of tooltip.
- [border](#) - Specifies the tooltip border color and width.
- [textStyle](#) - Specifies the tooltip font family, font style, font weight, color and size.


### Accessibility in Vue Bullet chart component

The Bullet chart component followed the accessibility guidelines and standards, including [ADA, Section 508](#), [WCAG 2.2](#) standards, and [WCAG roles](#) that are commonly used to evaluate accessibility.

The accessibility compliance for the Bullet chart component is outlined below.

| Accessibility Criteria | Compatibility |

| -- | -- |

| [WCAG 2.2](#) Support |  alt="Yes" |

```

| Section 508 Support |  |

| Screen Reader Support |  |

| Right-To-Left Support |  |

| Color Contrast |  |

| Mobile Device Support |  |

| Keyboard Navigation Support |  |

| Accessibility Checker Validation |  |

| Axe-core Accessibility Validation |  |

<style>

.post .post-content img {
display: inline-block;
margin: 0.5em 0;
}

</style>

<div> - All
features of the component meet the requirement.</div>

<div> - Some features of the component do not meet the requirement.</div>

<div> - The component does not meet the requirement.</div>

```

#### WAI-ARIA attributes

The Bullet chart component followed the [WAI-ARIA](#) patterns to meet the accessibility. The following ARIA attributes are used in the Bullet chart component:

- `img (role)`
- `button (role)`
- `aria-label (attribute)`
- `aria-pressed (attribute)`



### Keyboard interaction

The Bullet chart component followed the [keyboard interaction](#) guideline, making it easy for people who use assistive technologies (AT) and those who completely rely on keyboard navigation. The following keyboard shortcuts are supported by the Bullet chart component.

| **Press** | **To do this** |

| --- | --- |

| **Tab** | Moves the focus to the next element in the Bullet chart. |

| **Shift + Tab** | Moves the focus to the previous element in the Bullet chart. |

| **Ctrl + P** | Prints the Bullet chart. |

### Ensuring accessibility

The Bullet chart component's accessibility levels are ensured through an [accessibility-checker](#) and [axe-core](#) software tools during automated testing.

The accessibility compliance of the Bullet chart component is shown in the following sample. Open the [sample](#) in a new window to evaluate the accessibility of the Bullet chart component with accessibility tools.

```
{% previewsample "https://ej2.syncfusion.com/accessibility/bullet-chart.html" %}
```

See also

- [Accessibility in Syncfusion Vue components](#)

## ButtonGroup

### Getting Started with the Vue Button group Component in Vue 2

This article provides a step-by-step guide for setting up a Vue 2 project using [Vue-CLI](#) and integrating the Syncfusion Vue Button group component using the [Composition API](#) / [Options API](#).

#### Prerequisites

[System requirements for Syncfusion Vue UI components](#)

#### Dependencies

The list of dependencies required to use the Button component in your application is given below:

```
`js
|-- @syncfusion/ej2-vue-splitbuttons
|-- @syncfusion/ej2-base
|-- @syncfusion/ej2-buttons
|-- @syncfusion/ej2-vue-base
`,`
```

#### Setting up the Vue 2 project

To generate a Vue 2 project using Vue-CLI, use the [vue create](#) command. Follow these steps to install Vue CLI and create a new project:

```
`bash
```

```
npm install -g @vue/cli
```

```
vue create quickstart
```

```
cd quickstart
```

```
npm run serve
```

```
,
```

```
or
```

```
`bash
```

```
yarn global add @vue/cli
```

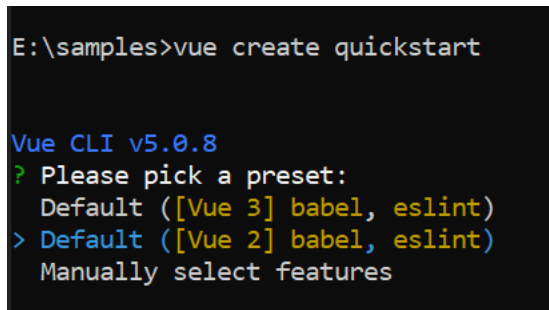
```
vue create quickstart
```

```
cd quickstart
```

```
yarn run serve
```

```
,
```

When creating a new project, choose the option `Default ([Vue 2] babel, eslint)` from the menu.



```
E:\samples>vue create quickstart

Vue CLI v5.0.8
? Please pick a preset:
  Default ([Vue 3] babel, eslint)
> Default ([Vue 2] babel, eslint)
  Manually select features
```

Once the `quickstart` project is set up with default settings, proceed to add Syncfusion components to the project.

#### Add Syncfusion Vue packages

Syncfusion packages are available at [npmjs.com](https://www.npmjs.com). To use Vue components, install the required npm package.

This article uses the [Vue Button group component](#) as an example. Install the `@syncfusion/ej2-splitbuttons` package by running the following command:

```
`bash
```

```
npm install @syncfusion/ej2-vue-splitbuttons --save
```

```
,
```

```
or
```

```
`bash
```

```
yarn add @syncfusion/ej2-vue-splitbuttons
```

```
,
```

### Import Syncfusion CSS styles

You can import themes for the Syncfusion Vue component in various ways, such as using CSS or SASS styles from npm packages, CDN, [CRG](#) and [Theme Studio](#). Refer to [themes topic](#) to know more about built-in themes and different ways to refer to themes in a Vue project.

In this article, the **Material** theme is applied using CSS styles, which are available in installed packages. The necessary Material CSS styles for the Button group component and its dependents were imported into the `<style>` section of **src/App.vue** file.

#### ~/SRC/APP.VUE

```
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-buttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-popups/styles/material.css";
@import "../node_modules/@syncfusion/ej2-splitbuttons/styles/material.css";
</style>
```

### Add Syncfusion Vue component

Follow the below steps to add the Vue Button group component using **Composition API** or **Options API**:

1\ First, import and register the Button group component in the **script** section of the **src/App.vue** file. If you are using the **Composition API**, you should add the **setup** attribute to the **script** tag to indicate that Vue will be using the **Composition API**.

#### COMPOSITION API (~/SRC/APP.VUE)

```
<script setup>
import { ButtonComponent as EjsButton } from '@syncfusion/ej2-vue-buttons';
</script>
```

#### OPTIONS API (~/SRC/APP.VUE)

```
<script>
import { ButtonComponent } from '@syncfusion/ej2-vue-buttons';
export default {
  components: {
    'ejs-button': ButtonComponent
  }
}
</script>
```

### Creating Vue Sample

Add a div element with class name as **e-btn-group** and add the button elements that needs to be group inside the **div** element in **App.vue** file.

To render button elements as EJ2 Vue Button, then add [Button dependencies](#) in **systemjs.config.js** file. Then import **ButtonPlugin** from **ej2-vue-buttons**.

#### ~/SRC/APP.VUE

```
<template>
```

```
<div id='app'>
  <div class="e-btn-group">
    <ejs-button>HTML</ejs-button>
    <ejs-button>CSS</ejs-button>
    <ejs-button>Javascript</ejs-button>
  </div>
</div>
</template>
```

Here is the summarized code for the above steps in the **src/App.vue** file:

#### **COMPOSITION API (~SRC/APP.VUE)**

```
<template>
  <div id='app'>
    <div class="e-btn-group">
      <ejs-button>HTML</ejs-button>
      <ejs-button>CSS</ejs-button>
      <ejs-button>Javascript</ejs-button>
    </div>
  </div>
</template>
<script setup>
import { ButtonComponent as EjsButton } from '@syncfusion/ej2-vue-buttons';
</script>
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-buttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-popups/styles/material.css";
@import "../node_modules/@syncfusion/ej2-splitbuttons/styles/material.css";
#app {
  margin: 20px;
}
.e-btn-group {
  margin: 25px 5px 20px 20px;
}
</style>
```

#### **OPTIONS API (~SRC/APP.VUE)**

```
<template>
  <div id='app'>
    <div class="e-btn-group">
      <ejs-button>HTML</ejs-button>
      <ejs-button>CSS</ejs-button>
      <ejs-button>Javascript</ejs-button>
    </div>
  </div>
</template>
<script>
import { ButtonComponent } from '@syncfusion/ej2-vue-buttons';
export default {
  components: {
    'ejs-button': ButtonComponent
  },
  name: 'app'
}
```

```

}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-buttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-popups/styles/material.css";
@import "../node_modules/@syncfusion/ej2-splitbuttons/styles/material.css";
#app {
  margin: 20px;
}
.e-btn-group {
  margin: 25px 5px 20px 20px;
}
</style>

```

### Run the project

To run the project, use the following command:

```
`bash
```

```
npm run serve
```

```
`
```

or

```
`bash
```

```
yarn run serve
```

```
`
```

```
{% previewsample "page.domainurl/code-snippet/button-group/getting-started-cs2" %}
```

### Orientation

ButtonGroup can be arranged in vertical and horizontal orientation. By default, it is horizontally aligned.

#### Vertical Orientation

ButtonGroup can be aligned vertically using the built-in CSS class `e-vertical` to the target element.

The following example illustrates how to achieve vertical orientation in ButtonGroup,

#### COMPOSITION API (~SRC/APP.VUE)

```

<template>
  <div id='app'>
    <div class="e-btn-group e-vertical">
      <ejs-button >HTML</ejs-button>
      <ejs-button >CSS</ejs-button>
      <ejs-button >Javascript</ejs-button>
    </div>
  </div>
</template>
<script setup>
import { ButtonComponent as EjsButton } from '@syncfusion/ej2-vue-buttons';
</script>
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-buttons/styles/material.css";

```

```
@import "../node_modules/@syncfusion/ej2-popups/styles/material.css";
@import "../node_modules/@syncfusion/ej2-splitbuttons/styles/material.css";
#app {
  margin: 20px;
}
.e-btn-group {
  margin: 25px 5px 20px 20px;
}
</style>
```

### OPTIONS API (~SRC/APP.VUE)

```
<template>
  <div id='app'>
    <div class="e-btn-group e-vertical">
      <ejs-button >HTML</ejs-button>
      <ejs-button >CSS</ejs-button>
      <ejs-button >Javascript</ejs-button>
    </div>
  </div>
</template>
<script>
import { ButtonComponent } from '@syncfusion/ej2-vue-buttons';
export default {
  components: {
    'ejs-button': ButtonComponent
  },
  name: 'app'
}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-buttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-popups/styles/material.css";
@import "../node_modules/@syncfusion/ej2-splitbuttons/styles/material.css";
#app {
  margin: 20px;
}
.e-btn-group {
  margin: 25px 5px 20px 20px;
}
</style>
```

{% previewsample "page.domainurl/code-snippet/button-group/getting-started-cs3" %}

ButtonGroup with SplitButton component nesting won't work in vertical orientation.

See Also

- [Initialize ButtonGroup using util function](#)

### Getting Started with the Vue ButtonGroup Component in Vue 3

This article provides a step-by-step guide for setting up a [Vite](#) project with a JavaScript environment and integrating the Syncfusion Vue ButtonGroup component using the [Composition API](#) / [Options API](#).

The **Composition API** is a new feature introduced in Vue.js 3 that provides an alternative way to organize and reuse component logic. It allows developers to write components as functions that use smaller, reusable functions called composition functions to manage their properties and behavior.

The **Options API** is the traditional way of writing Vue.js components, where the component logic is organized into a series of options that define the component's properties and behavior. These options include data, methods, computed properties, watchers, lifecycle hooks, and more.

#### Prerequisites

##### [System requirements for Syncfusion Vue UI components](#)

#### Set up the Vite project

A recommended approach for beginning with Vue is to scaffold a project using [Vite](#). To create a new Vite project, use one of the commands that are specific to either NPM or Yarn.

```
`bash
```

```
npm create vite@latest
```

```
,
```

or

```
`bash
```

```
yarn create vite
```

```
,
```

Using one of the above commands will lead you to set up additional configurations for the project as below:

1. Define the project name: We can specify the name of the project directly. Let's specify the name of the project as **my-project** for this article.

```
`bash
```

```
? Project name: » my-project
```

```
,
```

2. Select **Vue** as the framework. It will create a Vue 3 project.

```
`bash
```

```
? Select a framework: » - Use arrow-keys. Return to submit.
```

```
Vanilla
```

```
Vue
```

```
React
```

```
Preact
```

```
Lit
```

```
Svelte
```

```
Others
```

```
,
```

3. Choose `JavaScript` as the framework variant to build this Vite project using JavaScript and Vue.

```
`bash
```

? Select a variant: » - Use arrow-keys. Return to submit.

JavaScript

TypeScript

Customize with create-vue ↗

Nuxt ↗

,

4. Upon completing the aforementioned steps to create the `my-project`, run the following command to install its dependencies:

```
`bash
```

```
cd my-project
```

```
npm install
```

,

or

```
`bash
```

```
cd my-project
```

```
yarn install
```

,

Now that `my-project` is ready to run with default settings, let's add Syncfusion components to the project.

#### Add Syncfusion Vue packages

Syncfusion Vue component packages are available at [npmjs.com](https://www.npmjs.com). To use Syncfusion Vue components in the project, install the corresponding npm package.

This article uses the [Vue ButtonGroup component](#) as an example. To use the Vue ButtonGroup component in the project, the `@syncfusion/ej2-vue-buttons` package needs to be installed using the following command:

```
`bash
```

```
npm install @syncfusion/ej2-vue-buttons --save
```

,

or

```
`bash
```

```
yarn add @syncfusion/ej2-vue-buttons
```

,



### Import Syncfusion CSS styles

You can import themes for the Syncfusion Vue component in various ways, such as using CSS or SASS styles from npm packages, CDN, [CRG](#) and [Theme Studio](#). Refer to [themes topic](#) to know more about built-in themes and different ways to refer to themes in a Vue project.

In this article, **Material** theme is applied using CSS styles, which are available in installed packages. The necessary **Material** CSS styles for the Button component and its dependents were imported into the `<style>` section of **src/App.vue** file.

#### ~/SRC/APP.VUE

```
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-buttons/styles/material.css";
</style>
```

The order of importing CSS styles should be in line with its dependency graph.

### Add Syncfusion Vue component

Follow the below steps to add the Vue ButtonGroup component using **Composition API** or **Options API**:

1.First, import and register the ButtonGroup component and its child directives in the **script** section of the **src/App.vue** file. If you are using the **Composition API**, you should add the **setup** attribute to the **script** tag to indicate that Vue will be using the **Composition API**.

#### COMPOSITION API (~/SRC/APP.VUE)

```
<script setup>
import { ButtonComponent as EjsButton } from "@syncfusion/ej2-vue-buttons";
</script>
```

#### OPTIONS API (~/SRC/APP.VUE)

```
<script>
import { ButtonComponent } from "@syncfusion/ej2-vue-buttons";
//Component registration
export default {
  name: "App",
  components: {
    "ejs-button": ButtonComponent
  }
}
</script>
```

2.In the **template** section, define the ButtonGroup component.

#### ~/SRC/APP.VUE

```
<template>
<div id='app'>
<div class="e-btn-group">
<ejs-button>HTML</ejs-button>
<ejs-button>CSS</ejs-button>
</div>
</div>
</template>
```

```
<ejs-button>Javascript</ejs-button>
</div>
</div>
</template>
```

3. Here is the summarized code for the above steps in the **src/App.vue** file:

#### **COMPOSITION API (~SRC/APP.VUE)**

```
<template>
<div id='app'>
<div class="e-btn-group">
<ejs-button>HTML</ejs-button>
<ejs-button>CSS</ejs-button>
<ejs-button>Javascript</ejs-button>
</div>
</div>
</template>
<script setup>
import { ButtonComponent as EjsButton } from "@syncfusion/ej2-vue-buttons";
</script>
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-buttons/styles/material.css";
</style>
```

#### **OPTIONS API (~SRC/APP.VUE)**

```
<template>
<div id='app'>
<div class="e-btn-group">
<ejs-button>HTML</ejs-button>
<ejs-button>CSS</ejs-button>
<ejs-button>Javascript</ejs-button>
</div>
</div>
</template>
<script>
import { ButtonComponent } from '@syncfusion/ej2-vue-buttons';
//Component registration
export default {
name: "App",
components: {
"ejs-button": ButtonComponent
},
data() {
return {
};
}
}
</script>
```

#### Run the project

To run the project, use the following command:

```
`bash
```

```
npm run dev
```

```
,
```

```
or
```

```
`bash
```

```
yarn run dev
```

```
,
```

The output will appear as follows:

A screenshot showing three buttons arranged horizontally. The first button is labeled 'HTML', the second 'CSS', and the third 'JAVASCRIPT'. They are all in a light gray box with rounded corners.

**Sample:** [vue-3-button-group-getting-started](#).

For migrating from Vue 2 to Vue 3, refer to the [migration](#) documentation.

See also

- [Getting Started with Vue UI Components using Composition API and TypeScript](#)
- [Getting Started with Vue UI Components using Options API and TypeScript](#)

## Types and styles in Vue Button group component

This section explains about different types and styles of ButtonGroup.

### ButtonGroup types

#### *Outline ButtonGroup*

An Outline ButtonGroup has a border with transparent background. To create Outline ButtonGroup, `e-outline` class needs to be added to the target element and to the button element using [cssClass](#) property.

The following sample illustrates how to achieve outline ButtonGroup,

#### **APP.VUE**

```
<template>
  <div id='app'>
    <div class="e-btn-group e-outline">
      <ejs-button cssClass='e-outline'>HTML</ejs-button>
      <ejs-button cssClass='e-outline' >CSS</ejs-button>
      <ejs-button cssClass='e-outline' >Javascript</ejs-button>
    </div>
  </div>
</template>
<script>
import Vue from 'vue';
import { ButtonPlugin } from '@syncfusion/ej2-vue-buttons';
Vue.use(ButtonPlugin);
export default {
  name: 'app'
```

```

}
</script>
<style>
  @import '../node_modules/@syncfusion/ej2-base/styles/material.css';
  @import '../node_modules/@syncfusion/ej2-buttons/styles/material.css';
  @import '../node_modules/@syncfusion/ej2-popups/styles/material.css';
  @import '../node_modules/@syncfusion/ej2-splitbuttons/styles/material.css';
  #app {
    margin: 20px;
  }
  .e-btn-group {
    margin: 25px 5px 20px 20px;
  }
</style>

```

{% previewsample "page.domainurl/code-snippet/button-group/getting-started-cs16" %}

ButtonGroup does not have support for **flat** and **round** types.

### ButtonGroup styles

The Essential JS 2 ButtonGroup has the following predefined styles. This can be achieved by adding corresponding class name in each button elements using **cssClass** property.

Class	Description
-----	-----
e-primary	Used to represent a primary action.
e-success	Used to represent a positive action.
e-info	Used to represent an informative action.
e-warning	Used to represent an action with caution.
e-danger	Used to represent a negative action.

The following example illustrates how to achieve predefined styles in ButtonGroup,

### APP.VUE

```

<template>
  <div id='app'>
    <div class="e-btn-group">
      <ejs-button cssClass='e-info'>View</ejs-button>
      <ejs-button>Edit</ejs-button>
      <ejs-button cssClass='e-danger'>Delete</ejs-button>
    </div>
  </div>
</template>
<script>
import Vue from 'vue';
import { ButtonPlugin } from '@syncfusion/ej2-vue-buttons';
Vue.use(ButtonPlugin);
export default {
  name: 'app'
}

```

```

</script>
<style>
@import '../node_modules/@syncfusion/ej2-base/styles/material.css';
@import '../node_modules/@syncfusion/ej2-buttons/styles/material.css';
@import '../node_modules/@syncfusion/ej2-popups/styles/material.css';
@import '../node_modules/@syncfusion/ej2-splitbuttons/styles/material.css';
#app {
  margin: 20px;
}
.e-btn-group {
  margin: 25px 5px 20px 20px;
}
</style>

```

{% previewsample "page.domainurl/code-snippet/button-group/getting-started-cs17" %}

See Also

- [ButtonGroup with icons](#)
- [Create ButtonGroup with rounded corner](#)

## Selection and nesting in Vue Button group component

### Selection

#### Single selection

ButtonGroup supports radio type selection in which only one button can be selected. This can be achieved by adding input element along with `id` attribute with its corresponding label along with `for` attribute inside the target element. In this ButtonGroup, the type of the input element should be `radio` and `e-btn` is added to the `label` element.

The following example illustrates the single selection behavior in ButtonGroup,

#### APP.VUE

```

<template>
  <div id='app'>
    <div class='e-btn-group'>
      <input type="radio" id="radioleft" name="align" value="left"/>
      <label class="e-btn" for="radioleft">Left</label>
      <input type="radio" id="radiomiddle" name="align" value="middle"/>
      <label class="e-btn" for="radiomiddle">Center</label>
      <input type="radio" id="radiatoright" name="align" value="right"/>
      <label class="e-btn" for="radiatoright">Right</label>
    </div>
  </div>
</template>
<script>
import Vue from 'vue';
export default {
  name: 'app'
}
</script>
<style>
@import '../node_modules/@syncfusion/ej2-base/styles/material.css';
@import '../node_modules/@syncfusion/ej2-buttons/styles/material.css';

```

```
@import '../node_modules/@syncfusion/ej2-popups/styles/material.css';
@import '../node_modules/@syncfusion/ej2-splitbuttons/styles/material.css';
#app {
  margin: 20px;
}
.e-btn-group {
  margin: 25px 5px 20px 20px;
}
</style>
```

{% previewsample "page.domainurl/code-snippet/button-group/getting-started-cs12" %}

### Multiple selection

ButtonGroup supports checkbox type selection in which multiple button can be selected. This can be achieved by adding input element with its corresponding label. In this ButtonGroup, the type of the input element should be `checkbox` and `e-btn` is added to the `label` element.

The following example illustrates the multiple selection behavior in ButtonGroup,

### APP.VUE

```
<template>
  <div id='app'>
    <div class='e-btn-group'>
      <input type="checkbox" id="checkbold" name="font" value="bold"/>
      <label class="e-btn" for="checkbold">Bold</label>
      <input type="checkbox" id="checkitalic" name="font" value="italic" />
    </div>
    <label class="e-btn" for="checkitalic">Italic</label>
    <input type="checkbox" id="checkline" name="font" value="underline"/>
    <label class="e-btn" for="checkline">Underline</label>
  </div>
</template>
<script>
import Vue from 'vue';
export default {
  name: 'app'
}
</script>
<style>
@import '../node_modules/@syncfusion/ej2-base/styles/material.css';
@import '../node_modules/@syncfusion/ej2-buttons/styles/material.css';
@import '../node_modules/@syncfusion/ej2-popups/styles/material.css';
@import '../node_modules/@syncfusion/ej2-splitbuttons/styles/material.css';
#app {
  margin: 20px;
}
.e-btn-group {
  margin: 25px 5px 20px 20px;
}
</style>
```

{% previewsample "page.domainurl/code-snippet/button-group/getting-started-cs13" %}

### Nesting

Nesting with other components can be possible in ButtonGroup. The following components can be nested in ButtonGroup.

- DropDownButton
- SplitButton

For nesting support, [SplitButton dependencies](#) should be configured and it should be added in `system.config.js`.

#### DropDownButton

To initialize DropDownButton component refer [DropDownButton Getting Started documentation](#).

In the following example, DropDownButton component is added by importing `DropDownButtonPlugin` from `ej2-vue-splitbuttons`.

#### APP.VUE

```
<template>
  <div id='app'>
    <div class="e-btn-group">
      <div class='e-btn-group'>
        <ejs-button >HTML</ejs-button>
        <ejs-button >CSS</ejs-button>
        <ejs-button >Javascript</ejs-button>
        <ejs-dropdownbutton :items='items'>More</ejs-dropdownbutton>
      </div>
    </div>
  </div>
</template>
<script>
import Vue from 'vue';
import { ButtonPlugin } from '@syncfusion/ej2-vue-buttons';
import { DropDownButtonPlugin } from "@syncfusion/ej2-vue-splitbuttons";
Vue.use(ButtonPlugin);
Vue.use(DropDownButtonPlugin);
export default {
  name: 'app',
  data () {
    return {
      items:[
        {
          text: 'Learn SQL'
        },
        {
          text: 'Learn PHP'
        },
        {
          text: 'Learn Bootstrap'
        }
      ]
    };
  }
}
</script>
<style>
```

```
@import '../node_modules/@syncfusion/ej2-base/styles/material.css';
@import '../node_modules/@syncfusion/ej2-buttons/styles/material.css';
@import '../node_modules/@syncfusion/ej2-popups/styles/material.css';
@import '../node_modules/@syncfusion/ej2-splitbuttons/styles/material.css';
#app {
  margin: 20px;
}
</style>
```

{% previewsample "page.domainurl/code-snippet/button-group/getting-started-cs14" %}

### SplitButton

To initialize SplitButton component refer [SplitButton Getting Started documentation](#).

In the following example, SplitButton component is added by importing SplitButtonPlugin from ej2-vue-splitbuttons.

### APP.VUE

```
<template>
  <div id='app'>
    <div class="e-btn-group e-vertical">
      <div class='e-btn-group'>
        <ejs-button content='Cut'></ejs-button>
        <ejs-button content='Copy'></ejs-button>
        <ejs-splitbutton :items='items' content='Paste'></ejs-splitbutton>
      </div>
    </div>
  </div>
</template>
<script>
import Vue from 'vue';
import { SplitButtonPlugin } from "@syncfusion/ej2-vue-splitbuttons";
import { ButtonPlugin } from '@syncfusion/ej2-vue-buttons';
Vue.use(ButtonPlugin);
Vue.use(SplitButtonPlugin );
export default {
  name: 'app',
  data () {
    return {
      items:[
        {
          text: 'Paste'
        },
        {
          text: 'Paste Text'
        },
        {
          text: 'Paste Special'
        }
      ]
    };
  }
}
</script>
<style>
```



```
@import '../node_modules/@syncfusion/ej2-base/styles/material.css';
@import '../node_modules/@syncfusion/ej2-buttons/styles/material.css';
@import '../node_modules/@syncfusion/ej2-popups/styles/material.css';
@import '../node_modules/@syncfusion/ej2-splitbuttons/styles/material.css';
#app {
  margin: 20px;
}
</style>
```

{% previewsample "page.domainurl/code-snippet/button-group/getting-started-cs15" %}

See Also

- [Show ButtonGroup selected state on initial render](#)

### Accessibility in Vue Button group component

The Button group component followed the accessibility guidelines and standards, including [ADA, Section 508](#), [WCAG 2.2](#) standards, and [WCAG roles](#) that are commonly used to evaluate accessibility.

The accessibility compliance for the Button group component is outlined below.

| Accessibility Criteria | Compatibility |

| -- | -- |

| [WCAG 2.2](#) Support |  |

| [Section 508](#) Support |  |

| Screen Reader Support |  |

| Right-To-Left Support |  |

| Color Contrast |  |

| Mobile Device Support |  |

| Keyboard Navigation Support |  |

| [Accessibility Checker](#) Validation |  |

| [Axe-core](#) Accessibility Validation |  |

<style>

.post .post-content img {

display: inline-block;

```
margin: 0.5em 0;
```

```
}
```

```
</style>
```

```
<div> - All features of the component meet the requirement.</div>
```

```
<div> - Some features of the component do not meet the requirement.</div>
```

```
<div> - The component does not meet the requirement.</div>
```

### Keyboard interaction

The Button group component followed the [keyboard interaction](#) guideline, making it easy for people who use assistive technologies (AT) and those who completely rely on keyboard navigation. The following keyboard shortcuts are supported by the Button group component.

#### Normal behavior

Press	To do this
---   ---	
Tab	Focuses the next button in the ButtonGroup.
Enter/Space	Activates the focussed button in the ButtonGroup.

#### Checkbox behavior

Press	To do this
---   ---	
Tab	Focuses the next button in the ButtonGroup.
Space	Activates the focussed button in the ButtonGroup.

#### Radiobutton behavior

Press	To do this
---   ---	
Tab	Focuses the active button in the ButtonGroup.
Right	Activates next/previous button in the ButtonGroup.

### Ensuring accessibility

The Button group component's accessibility levels are ensured through an [accessibility-checker](#) and [axe-core](#) software tools during automated testing.

The accessibility compliance of the Button group component is shown in the following sample. Open the [sample](#) in a new window to evaluate the accessibility of the Button group component with accessibility tools.

```
{% previewsample "page.domainurl/code-snippet/button-group/getting-started-cs1" %}
```

See also

- [Accessibility in Syncfusion Vue components](#)

### Style and appearance in Vue Button group component

To modify the ButtonGroup appearance, you need to override the default CSS of ButtonGroup component. Please find the list of CSS classes and its corresponding section in ButtonGroup. Also, you have an option to create your own custom theme for the controls using our [Theme Studio](#).

#### CSS Class | Purpose of Class

- |.e-btn|To customize the ButtonGroup.
- |.e-btn: hover|To customize the ButtonGroup on hover.
- |.e-btn: focus|To customize the ButtonGroup on focus.
- |.e-btn: active|To customize the ButtonGroup on active.

### How To

#### Create buttongroup with icons in Vue Button group component

To create ButtonGroup with icons, [iconCss](#) property of Button component can be used.

The following example illustrates how to create ButtonGroup with icons,

#### APP.VUE

```
<template>
  <div id='app'>
    <div class="e-btn-group">
      <ejs-button content='Left' iconCss='e-icons e-left-icon'></ejs-
button>
      <ejs-button content='Center' iconCss='e-icons e-middle-icon'></ejs-
button>
      <ejs-button content='Right' iconCss='e-icons e-right-icon'></ejs-
button>
    </div>
  </div>
</template>
<script>
import Vue from 'vue';
import { ButtonPlugin } from '@syncfusion/ej2-vue-buttons';
Vue.use(ButtonPlugin);
export default {
  name: 'app'
}
</script>
<style>
@import '../node_modules/@syncfusion/ej2-base/styles/material.css';
@import '../node_modules/@syncfusion/ej2-buttons/styles/material.css';
@import '../node_modules/@syncfusion/ej2-popups/styles/material.css';
@import '../node_modules/@syncfusion/ej2-splitbuttons/styles/material.css';
#app {
  margin: 20px;
}
.e-btn-group {
  margin: 25px 5px 20px 20px;
}
```

```

    }
    .e-left-icon::before {
      content: '\e33a';
    }
    .e-right-icon::before {
      content: '\e34d';
    }
    .e-middle-icon::before {
      content: '\e35e';
    }
  }
</style>

```

{% previewsample "page.domainurl/code-snippet/button-group/getting-started-cs4" %}

### Create buttongroup with rounded corner in Vue Button group component

The ButtonGroup with rounded corner has round edges on both sides. To ButtonGroup with rounded corner, `e-round-corner` class is to be added to the target element.

The following example illustrates how to create ButtonGroup with rounded corner,

#### APP.VUE

```

<template>
  <div id='app'>
    <div class='e-btn-group e-round-corner'>
      <ejs-button content='HTML'></ejs-button>
      <ejs-button content='CSS'></ejs-button>
      <ejs-button content='Javascript'></ejs-button>
    </div>
  </div>
</template>
<script>
import Vue from 'vue';
import { ButtonPlugin } from '@syncfusion/ej2-vue-buttons';
import { enableRipple } from '@syncfusion/ej2-base';
enableRipple(true);
Vue.use(ButtonPlugin);
export default {
  name: 'app'
}
</script>
<style>
@import '../node_modules/@syncfusion/ej2-base/styles/material.css';
@import '../node_modules/@syncfusion/ej2-buttons/styles/material.css';
@import '../node_modules/@syncfusion/ej2-popups/styles/material.css';
@import '../node_modules/@syncfusion/ej2-splitbuttons/styles/material.css';
#app {
  margin: 20px;
}
.e-btn-group {
  margin: 25px 5px 20px 20px;
}
</style>

```

{% previewsample "page.domainurl/code-snippet/button-group/getting-started-cs5" %}

Disable in Vue Button group component

#### Particular button

To disable a particular button in a ButtonGroup, [disabled](#) attribute needs to be added to corresponding button element.

#### Whole ButtonGroup

To disable whole ButtonGroup, [disabled](#) attribute needs to be added to all the button elements.

The following example illustrates how to disable the particular and the whole ButtonGroup.

#### APP.VUE

```
<template>
  <div id='app'>
    <div class='e-btn-group'>
      <ejs-button >HTML</ejs-button>
      <ejs-button disabled='true'>CSS</ejs-button>
      <ejs-button >Javascript</ejs-button>
    </div>
    <div class='e-btn-group'>
      <ejs-button disabled='true'>HTML</ejs-button>
      <ejs-button disabled='true'>CSS</ejs-button>
      <ejs-button disabled='true'>Javascript</ejs-button>
    </div>
  </div>
</template>
<script>
import Vue from 'vue';
import { ButtonPlugin } from '@syncfusion/ej2-vue-buttons';
import { enableRipple } from '@syncfusion/ej2-base';
enableRipple(true);
Vue.use(ButtonPlugin);
export default {
  name: 'app'
}
</script>
<style>
@import '../node_modules/@syncfusion/ej2-base/styles/material.css';
@import '../node_modules/@syncfusion/ej2-buttons/styles/material.css';
@import '../node_modules/@syncfusion/ej2-popups/styles/material.css';
@import '../node_modules/@syncfusion/ej2-splitbuttons/styles/material.css';
#app {
  margin: 20px;
}
.e-btn-group {
  margin: 25px 5px 20px 20px;
}
</style>
```

{% previewsample "page.domainurl/code-snippet/button-group/getting-started-cs6" %}

To disable radio/checkbox type ButtonGroup, the `disabled` attribute should be added to the input element.

### Enable ripple in Vue Button group component

Ripple can be enabled by importing `rippleEffect` method from `ej2-base` and initialize rippleEffect with `.e-btn-group` element and selector as `e-btn`.

The following example illustrates how to enable ripple for ButtonGroup,

#### APP.VUE

```
<template>
  <div id='app'>
    <div class='e-btn-group'>
      <ejs-button>HTML</ejs-button>
      <ejs-button>CSS</ejs-button>
      <ejs-button>Javascript</ejs-button>
    </div>
  </div>
</template>
<script>
import Vue from 'vue';
import { ButtonPlugin } from '@syncfusion/ej2-vue-buttons';
import { enableRipple, rippleEffect } from '@syncfusion/ej2-base';
enableRipple(true);
Vue.use(ButtonPlugin);
export default {
  name: 'app',
  mounted() {
    var button= document.querySelector('.e-btn-group');
    rippleEffect(button, { selector: '.e-btn' });
  }
}
</script>
<style>
@import '../node_modules/@syncfusion/ej2-base/styles/material.css';
@import '../node_modules/@syncfusion/ej2-buttons/styles/material.css';
@import '../node_modules/@syncfusion/ej2-popups/styles/material.css';
@import '../node_modules/@syncfusion/ej2-splitbuttons/styles/material.css';
#app {
  margin: 20px;
}
.e-btn-group {
  margin: 25px 5px 20px 20px;
}
</style>
```

{% previewsample "page.domainurl/code-snippet/button-group/getting-started-cs7" %}

### Enable rtl in Vue Button group component

ButtonGroup supports RTL functionality. This can be achieved by adding `e-rtl` class to the target element.

The following example illustrates how to create ButtonGroup with RTL support.

#### APP.VUE

```
<template>
  <div id='app'>
    <div class='e-btn-group e-rtl'>
```

```

        <ejs-button content='HTML'></ejs-button>
        <ejs-button content='CSS'></ejs-button>
        <ejs-button content='Javascript'></ejs-button>
    </div>
</div>
</template>
<script>
import Vue from 'vue';
import { ButtonPlugin } from '@syncfusion/ej2-vue-buttons';
import { enableRipple } from '@syncfusion/ej2-base';
enableRipple(true);
Vue.use(ButtonPlugin);
export default {
    name: 'app'
}
</script>
<style>
@import '../node_modules/@syncfusion/ej2-base/styles/material.css';
@import '../node_modules/@syncfusion/ej2-buttons/styles/material.css';
@import '../node_modules/@syncfusion/ej2-popups/styles/material.css';
@import '../node_modules/@syncfusion/ej2-splitbuttons/styles/material.css';
#app {
    margin: 20px;
}
.e-btn-group {
    margin: 25px 5px 20px 20px;
}
</style>

```

{% previewsample "page.domainurl/code-snippet/button-group/getting-started-cs8" %}

### Form submit in Vue Button group component

The name attribute of the input element is used to group checkbox/radio type ButtonGroup. When the radio/checkbox type are grouped in form, the checked items value attribute will be post to server on form submit that can be retrieved through the name. The disabled radio/checkbox type value will not be sent to the server on form submit.

In the following code snippet, the radio type ButtonGroup is explained with male value as checked. Now, the value that is in checked state will be sent on form submit.

### APP.VUE

```

<template>
  <div id='app'>
    <form>
      <div class='e-btn-group'>
        <input type="radio" id="male" name="gender" value="male" checked/>
        <label class="e-btn" for="male">Male</label>
        <input type="radio" id="female" name="gender" value="female"/>
        <label class="e-btn" for="female">Female</label>
        <input type="radio" id="transgender" name="gender" value="transgender"/>
        <label class="e-btn" for="transgender">Transgender</label>
      </div>
      <ejs-button isPrimary='true'>Submit</ejs-button>
    </form>
  </div>
</template>

```

```

    </form>
  </div>
</template>
<script>
import Vue from 'vue';
import { ButtonPlugin } from '@syncfusion/ej2-vue-buttons';
import { enableRipple } from '@syncfusion/ej2-base';
enableRipple(true);
Vue.use(ButtonPlugin);
export default {
  name: 'app'
}
</script>
<style>
@import '../node_modules/@syncfusion/ej2-base/styles/material.css';
@import '../node_modules/@syncfusion/ej2-buttons/styles/material.css';
@import '../node_modules/@syncfusion/ej2-popups/styles/material.css';
@import '../node_modules/@syncfusion/ej2-splitbuttons/styles/material.css';
#app {
  margin: 20px;
}
.e-btn-group, button {
  margin: 20px 5px 20px 20px;
}
</style>

```

{% previewsample "page.domainurl/code-snippet/button-group/getting-started-cs9" %}

Initialize buttongroup using util function in Vue Button group component

Though it is a CSS component, for easy initialization of ButtonGroup `createButtonGroup` util function can be used.

To use `createButtonGroup` util function, [SplitButton dependencies](#) should be configured and it should be added in `system.config.js`.

Using `createButtonGroup` method, the Button options, selector, and `cssClass` is passed and the corresponding classes is added to the elements.

#### *For basic ButtonGroup*

To create basic ButtonGroup, the target element along with the button elements needs to be created and `createButtonGroup` is to be imported from `ej2-splitbuttons` package.

#### *For radio type ButtonGroup*

To create a radio type ButtonGroup, the target element along with the input elements should be created with type `radio`.

#### *For checkbox type ButtonGroup*

Checkbox type ButtonGroup creation is similar to radio type ButtonGroup, instead the type of the input elements should be `checkbox`.

The following example illustrates how to create ButtonGroup using `createButtonGroup` function for basic, checkbox and radio type behavior.

#### **APP.VUE**



```

<template>
  <div id='app'>
    <h5>Normal behavior</h5>
    <div id='basic'>
      <button></button>
      <button></button>
      <button></button>
    </div>
    <h5>Checkbox type behavior</h5>
    <div id='checkbox'>
      <input type="checkbox" id="checkbold" name="font" value='bold' />
      <input type="checkbox" id="checkitalic" name="font" value='italic' />
      <input type="checkbox" id="checkunderline" name="font" value='underline' />
    </div>
    <h5>Radiobutton type behavior</h5>
    <div id='radio'>
      <input type="radio" id="radioleft" name="align" value='left' />
      <input type="radio" id="radiomiddle" name="align" value='middle' />
      <input type="radio" id="radiatoright" name="align" value='right' />
    </div>
  </div>
</template>
<script>
import Vue from 'vue';
import { createButtonGroup } from '@syncfusion/ej2-splitbuttons';
import { ButtonPlugin } from '@syncfusion/ej2-vue-buttons';
Vue.use(ButtonPlugin);
export default {
  name: 'app',
  mounted() {
    createButtonGroup('#basic', {
      buttons: [
        { content: 'HTML' },
        { content: 'CSS' },
        { content: 'Javascript' }
      ]
    });
    createButtonGroup('#checkbox', {
      buttons: [
        { content: 'Bold' },
        { content: 'Italic' },
        { content: 'Undeline' }
      ]
    });
    createButtonGroup('#radio', {
      buttons: [
        { content: 'Left' },
        { content: 'Center' },
        { content: 'Right' }
      ]
    });
  }
}
</script>
<style>

```

```
@import '../node_modules/@syncfusion/ej2-base/styles/material.css';
@import '../node_modules/@syncfusion/ej2-vue-buttons/styles/material.css';
@import '../node_modules/@syncfusion/ej2-buttons/styles/material.css';
@import '../node_modules/@syncfusion/ej2-popups/styles/material.css';
@import '../node_modules/@syncfusion/ej2-splitbuttons/styles/material.css';
.e-btn-group {
  margin: 0 5px 5px 5px;
}
</style>
```

{% previewsample "page.domainurl/code-snippet/button-group/getting-started-cs10" %}

Show buttongroup selected state on initial render in Vue Button group component

To show selected state on initial render, `checked` property should be added to the corresponding input element.

The following example illustrates how to show selected state on initial render.

#### APP.VUE

```
<template>
  <div id='app'>
    <div class='e-btn-group'>
      <input type="checkbox" id="checkbold" name="font" value="bold" checked/>
      <label class="e-btn" for="checkbold">Bold</label>
      <input type="checkbox" id="checkitalic" name="font" value="italic" />
      <label class="e-btn" for="checkitalic">Italic</label>
      <input type="checkbox" id="checkline" name="font" value="underline" />
      <label class="e-btn" for="checkline">Underline</label>
    </div>
  </div>
</template>
<script>
import Vue from 'vue';
export default {
  name: 'app'
}
</script>
<style>
@import '../node_modules/@syncfusion/ej2-base/styles/material.css';
@import '../node_modules/@syncfusion/ej2-buttons/styles/material.css';
@import '../node_modules/@syncfusion/ej2-popups/styles/material.css';
@import '../node_modules/@syncfusion/ej2-splitbuttons/styles/material.css';
#app {
  margin: 20px;
}
.e-btn-group {
  margin: 25px 5px 20px 20px;
}
</style>
```

{% previewsample "page.domainurl/code-snippet/button-group/getting-started-cs11" %}

## Button

### Getting Started with the Vue Button Component in Vue 2

This article provides a step-by-step guide for setting up a Vue 2 project using [Vue-CLI](#) and integrating the Syncfusion Vue Button component using the [Composition API](#) / [Options API](#).

#### Prerequisites

##### [System requirements for Syncfusion Vue UI components](#)

#### Dependencies

The list of dependencies required to use the Button component in your application is given below :

```
`js
|-- @syncfusion/ej2-vue-buttons
|-- @syncfusion/ej2-base
|-- @syncfusion/ej2-buttons
|-- @syncfusion/ej2-vue-base
`,`
```

#### Setting up the Vue 2 project

To generate a Vue 2 project using Vue-CLI, use the [vue create](#) command. Follow these steps to install Vue CLI and create a new project:

```
`bash
npm install -g @vue/cli
vue create quickstart
cd quickstart
npm run serve
`,`
```

or

```
`bash
yarn global add @vue/cli
vue create quickstart
cd quickstart
yarn run serve
`,`
```

When creating a new project, choose the option **Default ([Vue 2] babel, eslint)** from the menu.

```
E:\samples>vue create quickstart

Vue CLI v5.0.8
? Please pick a preset:
  Default ([Vue 3] babel, eslint)
> Default ([Vue 2] babel, eslint)
  Manually select features
```

Once the `quickstart` project is set up with default settings, proceed to add Syncfusion components to the project.

### Add Syncfusion Vue packages

Install Syncfusion `Button` packages using below command.

Syncfusion packages are available at [npmjs.com](https://www.npmjs.com). To use Vue components, install the required npm package.

This article uses the [Vue Button component](#) as an example. Install the `@syncfusion/ej2-vue-buttons` package by running the following command:

```
`bash
```

```
npm install @syncfusion/ej2-vue-buttons --save
```

```
,
```

or

```
`bash
```

```
yarn add @syncfusion/ej2-vue-buttons
```

```
,
```

### Import Syncfusion CSS styles

You can import themes for the Syncfusion Vue component in various ways, such as using CSS or SASS styles from npm packages, CDN, [CRG](#) and [Theme Studio](#). Refer to [themes topic](#) to know more about built-in themes and different ways to refer to themes in a Vue project.

In this article, the `Material` theme is applied using CSS styles, which are available in installed packages. The necessary `Material` CSS styles for the `Button` component and its dependents were imported into the `<style>` section of `src/App.vue` file.

### ~SRC/APP.VUE

```
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-buttons/styles/material.css";
</style>
```

### Add Syncfusion Vue component

Follow the below steps to add the Vue Button component using `Composition API` or `Options API`:

1\ First, import and register the Button component in the `script` section of the `src/App.vue` file. If you are using the `Composition API`, you should add the `setup` attribute to the `script` tag to indicate that Vue will be using the `Composition API`.

#### **COMPOSITION API (~SRC/APP.VUE)**

```
<script setup>
import { ButtonComponent as EjsButton } from '@syncfusion/ej2-vue-buttons';
</script>
```

#### **OPTIONS API (~SRC/APP.VUE)**

```
<script>
import { ButtonComponent } from '@syncfusion/ej2-vue-buttons';
export default {
  components: {
    'ejs-button': ButtonComponent
  }
}
</script>
```

2\ In the `template` section, define the Button component.

#### **~/SRC/APP.VUE**

```
<template>
<ejs-button>Button</ejs-button>
</template>
```

Here is the summarized code for the above steps in the `src/App.vue` file:

#### **COMPOSITION API (~SRC/APP.VUE)**

```
<template>
<ejs-button>Button</ejs-button>
</template>
<script setup>
import { ButtonComponent as EjsButton } from '@syncfusion/ej2-vue-buttons';
import { enableRipple } from '@syncfusion/ej2-base';
enableRipple(true);
</script>
<style>
@import '../node_modules/@syncfusion/ej2-base/styles/material.css';
@import '../node_modules/@syncfusion/ej2-buttons/styles/material.css';
button {
  margin: 25px 5px 20px 20px;
}
</style>
```

#### **OPTIONS API (~SRC/APP.VUE)**

```
<template>
<ejs-button>Button</ejs-button>
</template>
<script>
```

```
import { ButtonComponent } from '@syncfusion/ej2-vue-buttons';
import { enableRipple } from '@syncfusion/ej2-base';
enableRipple(true);
export default {
  components: {
    'ejs-button': ButtonComponent
  }
}
</script>
<style>
@import '../node_modules/@syncfusion/ej2-base/styles/material.css';
@import '../node_modules/@syncfusion/ej2-buttons/styles/material.css';
button {
  margin: 25px 5px 20px 20px;
}
</style>
```

### Run the project

To run the project, use the following command:

```
`bash
```

```
npm run serve
```

```
,
```

or

```
`bash
```

```
yarn run serve
```

```
,
```

```
{% previewsample "page.domainurl/code-snippet/button/default-cs1" %}
```

### Change Button type

To change the default Button to flat Button, set the [cssClass](#) property to `e-flat` and text content of the Button is set using [content](#) property.

### COMPOSITION API (~SRC/APP.VUE)

```
<template>
<ejs-button cssClass='e-flat'>Button</ejs-button>
</template>
<script setup>
import { ButtonComponent as EjsButton } from '@syncfusion/ej2-vue-buttons';
import { enableRipple } from '@syncfusion/ej2-base';
enableRipple(true);
</script>
<style>
@import '../node_modules/@syncfusion/ej2-base/styles/material.css';
@import '../node_modules/@syncfusion/ej2-buttons/styles/material.css';
button {
  margin: 25px 5px 20px 20px;
}
</style>
```

**OPTIONS API (~SRC/APP.VUE)**

```
<template>
<ejs-button cssClass='e-flat'>Button</ejs-button>
</template>
<script>
import { ButtonComponent } from '@syncfusion/ej2-vue-buttons';
import { enableRipple } from '@syncfusion/ej2-base';
enableRipple(true);
export default {
  components: {
    'ejs-button': ButtonComponent
  }
}
</script>
<style>
@import '../node_modules/@syncfusion/ej2-base/styles/material.css';
@import '../node_modules/@syncfusion/ej2-buttons/styles/material.css';
button {
  margin: 25px 5px 20px 20px;
}
</style>
```

{% previewsample "page.domainurl/code-snippet/button/default-cs2" %}

See Also

- [Types of Button](#)

## Getting Started with the Vue Button Component in Vue 3

This article provides a step-by-step guide for setting up a [Vite](#) project with a JavaScript environment and integrating the Syncfusion Vue Button component using the [Composition API](#) / [Options API](#).

The **Composition API** is a new feature introduced in Vue.js 3 that provides an alternative way to organize and reuse component logic. It allows developers to write components as functions that use smaller, reusable functions called composition functions to manage their properties and behavior.

The **Options API** is the traditional way of writing Vue.js components, where the component logic is organized into a series of options that define the component's properties and behavior. These options include data, methods, computed properties, watchers, lifecycle hooks, and more.

### Prerequisites

#### [System requirements for Syncfusion Vue UI components](#)

#### Set up the Vite project

A recommended approach for beginning with Vue is to scaffold a project using [Vite](#). To create a new Vite project, use one of the commands that are specific to either NPM or Yarn.

```
`bash
```

```
npm create vite@latest
```

```
`
```

or

```
`bash
yarn create vite
`
```

Using one of the above commands will lead you to set up additional configurations for the project as below:

1. Define the project name: We can specify the name of the project directly. Let's specify the name of the project as `my-project` for this article.

```
`bash
? Project name: » my-project
`
```

2. Select `Vue` as the framework. It will create a Vue 3 project.

```
`bash
? Select a framework: » - Use arrow-keys. Return to submit.
Vanilla
Vue
React
Preact
Lit
Svelte
Others
`
```

3. Choose `JavaScript` as the framework variant to build this Vite project using JavaScript and Vue.

```
`bash
? Select a variant: » - Use arrow-keys. Return to submit.
JavaScript
TypeScript
Customize with create-vue ↗
Nuxt ↗
`
```

4. Upon completing the aforementioned steps to create the `my-project`, run the following command to install its dependencies:

```
`bash
cd my-project
npm install
```



```
`  
or  
`bash  
cd my-project  
yarn install  
`
```

Now that `my-project` is ready to run with default settings, let's add Syncfusion components to the project.

#### Add Syncfusion Vue packages

Syncfusion Vue component packages are available at [npmjs.com](https://www.npmjs.com). To use Syncfusion Vue components in the project, install the corresponding npm package.

This article uses the [Vue Button component](#) as an example. To use the Vue Button component in the project, the `@syncfusion/ej2-vue-buttons` package needs to be installed using the following command:

```
`bash  
npm install @syncfusion/ej2-vue-buttons --save  
`  
or  
`bash  
yarn add @syncfusion/ej2-vue-buttons  
`
```

#### Import Syncfusion CSS styles

You can import themes for the Syncfusion Vue component in various ways, such as using CSS or SASS styles from npm packages, CDN, [CRG](#) and [Theme Studio](#). Refer to [themes topic](#) to know more about built-in themes and different ways to refer to themes in a Vue project.

In this article, `Material` theme is applied using CSS styles, which are available in installed packages. The necessary `Material` CSS styles for the Button component and its dependents were imported into the `<style>` section of `src/App.vue` file.

#### ~/SRC/APP.VUE

```
<style>  
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";  
@import "../node_modules/@syncfusion/ej2-vue-buttons/styles/material.css";  
</style>
```

The order of importing CSS styles should be in line with its dependency graph.

#### Add Syncfusion Vue component

Follow the below steps to add the Vue Button component using `Composition API` or `Options API`:

1.First, import and register the Button component and its child directives in the `script` section of the `src/App.vue` file. If you are using the `Composition API`, you should add the `setup` attribute to the `script` tag to indicate that Vue will be using the `Composition API`.

#### **COMPOSITION API (~/SRC/APP.VUE)**

```
<script setup>
import { ButtonComponent as EjsButton } from "@syncfusion/ej2-vue-buttons";
</script>
```

#### **OPTIONS API (~/SRC/APP.VUE)**

```
<script>
import { ButtonComponent } from "@syncfusion/ej2-vue-buttons";
//Component registration
export default {
  name: "App",
  components: {
    "ejs-button": ButtonComponent
  }
}
</script>
```

2.In the `template` section, define the Button component with the `content` property.

#### **~/SRC/APP.VUE**

```
<template>
<ejs-button :content="content"></ejs-button>
</template>
```

3.Declare the value for the content property in the script section.

#### **COMPOSITION API (~/SRC/APP.VUE)**

```
<template>
<ejs-button :content="content"></ejs-button>
</template>
<script setup>
import { ButtonComponent as EjsButton } from "@syncfusion/ej2-vue-buttons";
const content = "Button";
</script>
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-buttons/styles/material.css";
</style>
```

#### **OPTIONS API (~/SRC/APP.VUE)**

```
<template>
<ejs-button :content="content"></ejs-button>
</template>
<script>
import { ButtonComponent } from '@syncfusion/ej2-vue-buttons';
//Component registration
```

```
export default {
  name: "App",
  components: {
    "ejs-button": ButtonComponent
  },
  data() {
    return {
      content: "Button"
    };
  }
}
```

### Run the project

To run the project, use the following command:

```
`bash
```

```
npm run dev
```

```
,
```

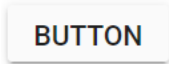
or

```
`bash
```

```
yarn run dev
```

```
,
```

The output will appear as follows:



**Sample:** [vue-3-button-getting-started](#).

For migrating from Vue 2 to Vue 3, refer to the [migration](#) documentation.

See also

- [Getting Started with Vue UI Components using Composition API and TypeScript](#)
- [Getting Started with Vue UI Components using Options API and TypeScript](#)

### Types and styles in Vue Button component

This section explains the different styles and types of Buttons.

#### Button styles

The Essential JS 2 Button has the following predefined styles that can be defined using the [cssClass](#) property.

Class	Description
-------	-------------

-----	-----
-------	-------

e-primary	Used to represent a primary action.
-----------	-------------------------------------

- | e-success | Used to represent a positive action. |
- | e-info | Used to represent an informative action. |
- | e-warning | Used to represent an action with caution. |
- | e-danger | Used to represent a negative action. |
- | e-link | Changes the appearance of the Button like a hyperlink. |

**APP.VUE**

```

<template>
<div>
  <ejs-button cssClass='e-primary'>Primary</ejs-button>
  <ejs-button cssClass='e-success'>Success</ejs-button>
  <ejs-button cssClass='e-info'>Info</ejs-button>
  <ejs-button cssClass='e-warning'>Warning</ejs-button>
  <ejs-button cssClass='e-danger'>Danger</ejs-button>
  <ejs-button cssClass='e-link'>Link</ejs-button>
</div>
</template>
<script>
import Vue from 'vue';
import { ButtonPlugin } from '@syncfusion/ej2-vue-buttons';
import { enableRipple } from '@syncfusion/ej2-base';
enableRipple(true);
Vue.use(ButtonPlugin);
export default {}
</script>
<style>
@import '../node_modules/@syncfusion/ej2-base/styles/material.css';
@import '../node_modules/@syncfusion/ej2-buttons/styles/material.css';
button {
  margin: 25px 5px 20px 20px;
}
</style>

```

{% previewsample "page.domainurl/code-snippet/button/default-cs12" %}

Predefined Button styles provide only the visual indication. So, Button content should define the Button style for the users of assistive technologies such as screen readers.

Primary action button can also be achieved by setting [isPrimary](#) property as `true`.

**Button types**

The types of Essential JS 2 Button are as follows:

- Basic types
- Flat Button
- Outline Button
- Round Button
- Toggle Button

*Basic types*

The basic Button types are explained below.

Type	Description
Button	Defines a click Button.
Submit	This Button submits the form data to the server.
Reset	This Button resets all the controls in the form elements to their initial values.

**APP.VUE**

```

<template>
  <form>
    <ejs-button type='submit'>Submit</ejs-button>
    <ejs-button type='reset'>Reset</ejs-button>
  </form>
</template>
<script>
import Vue from 'vue';
import { ButtonPlugin } from '@syncfusion/ej2-vue-buttons';
import { enableRipple } from '@syncfusion/ej2-base';
enableRipple(true);
Vue.use(ButtonPlugin);
export default {}
</script>
<style>
@import '../node_modules/@syncfusion/ej2-base/styles/material.css';
@import '../node_modules/@syncfusion/ej2-buttons/styles/material.css';
button {
  margin: 25px 5px 20px 20px;
}
</style>

```

```
{% previewsample "page.domainurl/code-snippet/button/default-cs13" %}
```

*Flat Button*

The Flat Button is styled with no background color. To create a flat Button, set the [cssClass](#) property to **e-flat**.

*Outline Button*

An outline Button has a border with transparent background. To create an outline Button, set the [cssClass](#) property to **e-outline**.

*Round Button*

A round Button is shaped like a circle. Usually, it contains an icon representing its action. To create a round Button, set the [cssClass](#) property to **e-round**.

**APP.VUE**

```

<template>
  <div>
    <ejs-button cssClass='e-flat'>Flat</ejs-button>
    <ejs-button cssClass='e-outline'>Outline</ejs-button>
    <ejs-button cssClass='e-round' iconCss='e-icons e-plus-icon'
isPrimary=true></ejs-button>
  </div>
</template>

```

```

<script>
import Vue from 'vue';
import { ButtonPlugin } from '@syncfusion/ej2-vue-buttons';
import { enableRipple } from '@syncfusion/ej2-base';
enableRipple(true);
Vue.use(ButtonPlugin);
export default {}
</script>
<style>
@import '../node_modules/@syncfusion/ej2-base/styles/material.css';
@import '../node_modules/@syncfusion/ej2-buttons/styles/material.css';
button {
    margin: 25px 5px 20px 20px;
}
.e-plus-icon::before {
    content: '\e823';
}
</style>

```

{% previewsample "page.domainurl/code-snippet/button/default-cs14" %}

### Toggle Button

A toggle Button allows you to change between the two states. The Button is active in toggled state and can be

recognized through the `e-active` class. The functionality of the toggle Button is handled by click event. To create a toggle Button, set the `isToggle` property to `true`. In the following code snippet, the toggle Button text changes to play/pause based on the state of the Button with the use of click event.

### APP.VUE

```

<template>
    <ejs-button ref="toggleBtn" cssClass='e-flat' iconCss='e-btn-sb-icon e-play-icon' isToggle=true v-on:click.native='btnClick'>Play</ejs-button>
</template>
<script>
import Vue from 'vue';
import { ButtonPlugin } from '@syncfusion/ej2-vue-buttons';
import { enableRipple } from '@syncfusion/ej2-base';
enableRipple(true);
Vue.use(ButtonPlugin);
export default {
    methods : {
        btnClick: function(event) {
            if (this.$refs.toggleBtn.$el.classList.contains('e-active')) {
                this.$refs.toggleBtn.content = 'Play';
                this.$refs.toggleBtn.iconCss = 'e-btn-sb-icon e-play-icon';
            } else {
                this.$refs.toggleBtn.content = 'Pause';
                this.$refs.toggleBtn.iconCss = 'e-btn-sb-icon e-pause-icon';
            }
        }
    }
}
</script>
<style>

```

```

@import '../node_modules/@syncfusion/ej2-base/styles/material.css';
@import '../node_modules/@syncfusion/ej2-buttons/styles/material.css';
button {
    margin: 25px 5px 20px 20px;
}
@font-face {
    font-family: 'btn-icon';
    src:
        url(data:application/x-font-ttf;charset=utf-
8;base64,AAEAAAAKAIAAAwAgTlMvMj1tSfgAAAEoAAAVmNtYXdhN+dzAAABoAAAAAEJnbHlm1v4
8pAAAAafgAAAYQYAGVhZBOPfZcAAADQAAAAANmhoZWEIuQQJAAAArAAAACRobXR4IAAAAAAAAAAYAAAA
gbG9jYQYN6ApQAAAHkAAAAEm1heHABFQCqAAABCAAAACBuYw1l07lFxAAABhAAAAIxG9zdK9uovo
AAAhEAAAAgAABAAAAEAAAAAFwEAAAAAAD9AABAAAAAAAAAAAAAAAAAAAAACAABAAAAAQAAJ1LUzF8
PPPUACwQAAAAAANg+nFMAAAAA2D6cUwAAAAAD9AP0AAAAACAACAAAAAAAAAAAAEAAAAIAJ4AAwAAAAA
AAgAAAAoACgAAAP8AAAAAAAAAAQAAAZAABQAAAokCzAAAAI8CiQLMAAAB6wAyAQgAAAIABQMAAAAA
AAAAAAAAAAAAAAAAAAAAAUGZFZABA5wDnBgQAAAAAXAQAAAAAAAABAAAAAAAAABAAAAQAAAA
EAAAAABAAAAAQAAAAEAAAABAAAAAQAAAAAAAACAAAAAwAAABQAAwABAAAAFAAEAC4AAAAEAAQAAQA
A5wb//wAA5wD//wAAAAEABAAAAEAAgADAAQABQAGAaCAAAAAAAAAADgAkADIAhAEuAEwCDAAAAAE
AAAAAA2ED9AACAAA3CQGeAsT9PAwB9AH0AAAACAAAAAAPHA/QAAwAHAAAAIRehASERIQJpAV7+ov3
QAV7+ogwD6PwYA+gAAAEAAAAAA4sD9AACAAATARF0AXgCAP4MA+gAAAAABAAAAAAP0A/QAQwAAExE
fDyE/DxEvDyEPDgwBAGMFBQcICQkLCwMDQ4NAtONDg0MDAsLCQkIBwUFAwIBAQIDBQUHCAkJCws
MDA00df0mDQ4NDawLCwkJCACFBQMCA239Jg4NDQ0LCwsJCQgHBQUdAgEBAGMFBQcICQkLCwsNDQ0
OAtODQ0NCwsLCQkIBwUFAwIBAQIDBQUHCAkJCwsLDQ0NAAIAAAAAA/MDxQADAIwAADczESMBdWm
VFw8METM3HwQ3Fz8KPQEvBT8LLwg3NT8INS8FNT8NNS8JBjYU/BDUvCyMPAQytrQH5AgoEAQEBAArg
hERESEyIJCsgQBiEHNQceOZPbdgUICw0LCQUDBAICBAkGAGABAQMOBAkIBgcDAwEBAQEDAwMJAgE
BAxYLBQQAeAwMCAgIEBAoBAQEECgcHBgUFBAMDAQEBAGQFBwkFBQUGef6tDwkEawIBAQMDCGwVawc
GDAsNBwdaAYcB3gEFAwN2HwoELDodGxwLwKIGwz+igEBHwMBAQECAQEDBgoKDAYICAgFCAkICwU
EBAQFAwYDBwgIDAgHCACGBgYFBQkEAgYCBawJBgUGBwkJCgkICAcLBAIFawIEBAQFBQcGBwgHBgY
GBgoJCAYCagEBAQFGMRkaGw0NDA0LIh4xBAQCBABEBAgADAAAAAOKA/MAHABCAJ0AAAEzHwIRDWm
hLwIDNzM/CjUTHwcVIwcVIy8HETcXMz8KNScxBxEfdjsSBHQEFDTMhMz8OES8PIz0BLw4ha0EDBQQ
DAQIEBf5eBQQCAW4RDg0LCQgGBQUDBAFEBAMDAwIBAQL7Y0EAWQCAgIBAYYKChEQDQsJCACBAU
CYt8BAQIDBAUFBQcHBwgICQgKjQECAGMEBAUFBgYHBgcIBwGcCAcHBWYGBgUFBQADAgIBAQEBAgI
DBAQFBQYGBgcHBwgMAQMDAwUFBgYHBwgICQkJ/tQCiwMEBf3XAwYEAgiEBgFoAQEDBQYGBwgIBw0
KhQEiAQEBAgMDAwTV+94BAQECAwMDBAGyAQECBAYHCAgJCgkQCAQC6/47CQkICQcIBWYGBQQEAWI
CUAgHBwcGBgYFBQQEAwMBAgIBAwMEBAUFBQcGBwCHCAImCAcHBWYGBgUFBQADAgIBADUJCQgICAQ
GBWYFBAQDagEBAAAAAAIAAAAAA6cD9AADAaAaADchNSElAQcJAScBESNZA078sgGB/uMuAXkBgdB
+1EwMTZcBCD3+ngFiPf7pAxMAAAAAABIA3gABAAAAAAAAAAAAEAAAABAAAAAABAAgAAQABAAAAAA
CAACACQABAAAAAADAAGAEABAAAAAAAEAAgAGAABAAAAAAFAAAsAIAABAAAAAAGAAGAKwABAAA
AAAAKACwAMwABAAAAAALABIAxwADAAEECCAAAAAIAcQADAAEECCABABAAAcwADAAEECCQACAA4AgwA
DAEECCQADABAAKQADAAEECCQAEBAAAOQADAAEECCQAFABYAsQADAAEECCQAGABAAxwADAAEECCQAKAFg
AlwADAAEECCQALACQALyBidG4taWNvblJlZ3VsYXJidG4taWNvbmJ0b1lpY29uVmVyc2lvbiAxLjB
idG4taWNvbkZvbnpQgZ2VuZXJhdGVkIHVzaW5nIFN5bmNmdXNpb24gTWV0cm8gU3RlZGlvd3d3LnN
5bmNmdXNpb24uY29tACAAYgB0AG4ALQBpAGMAbwBuAFIAZQBnAHUAbABhAHIAyYgB0AG4ALQBpAGM
AbwBuAGIAAdABuAC0AaQBjAG8AbgBWAGUAcgBzAGkAbwBuACAAMQAUADAAYgB0AG4ALQBpAGMAbwB
uAEYAbwBuAHQAIAbnAGUAbgBlAHIAyYQB0AGUAZAAGAHUAcwBpAG4AZwAgAFMAeQBwAGMAZgBlAHM
AaQBvAG4AIAbnAGUAdABYAG8AIAbTAHQAdQBkAGkAbwB3AHcAdwAuAHMAeQBwAGMAZgBlAHMAaQB
vAG4ALgBjAG8AbQAAAAACAAAAAaAAAOAAAAAaAAAgBAGEDAQBBQEGAQCBCAEJAAPtZWRpYS1wbGF5C21lZGhlLXBhdXNlDmFycm93aGVhZC1sZWZ0BHN0b3AJbGlrZS0
tLTAxBGNvcHkQLWRvd25sb2FkLTAYLXdmLQAA) format('truetype');
    font-weight: normal;
    font-style: normal;
}
.e-btn-sb-icon {
    font-family: 'btn-icon' !important;
    speak: none;
    font-size: 55px;
    font-style: normal;
}

```

```

    font-weight: normal;
    font-variant: normal;
    text-transform: none;
    line-height: 1;
    -webkit-font-smoothing: antialiased;
    -moz-osx-font-smoothing: grayscale;
  }
  /* Added when toggle button is in active state*/
  .e-play-icon::before {
    content: '\e700';
  }
  /* Added when toggle button is not in active state*/
  .e-pause-icon::before {
    content: '\e701';
  }
}
</style>

```

```
{% previewsample "page.domainurl/code-snippet/button/default-cs15" %}
```

## Icons

### Button with font icons

The Button can have an icon to provide the visual representation of the action. To place the icon on a Button,

set the [iconCss](#) property to `e-icons` with the required icon CSS. By default, the icon is positioned to the left side of the Button. You can customize the icon's position by using the [iconPosition](#) property.

## APP.VUE

```

<template>
  <div>
    <ejs-button iconCss='e-btn-sb-icon e-prev-icon'>Previous</ejs-
button>
    <ejs-button iconCss='e-btn-sb-icon e-stop-icon'
iconPosition='Right'>Stop</ejs-button>
  </div>
</template>
<script>
import Vue from 'vue';
import { ButtonPlugin } from '@syncfusion/ej2-vue-buttons';
import { enableRipple } from '@syncfusion/ej2-base';
enableRipple(true);
Vue.use(ButtonPlugin);
export default {}
</script>
<style>
@import '../node_modules/@syncfusion/ej2-base/styles/material.css';
@import '../node_modules/@syncfusion/ej2-buttons/styles/material.css';
button {
  margin: 25px 5px 20px 20px;
}
@font-face {
  font-family: 'btn-icon';
  src:
    url(data:application/x-font-ttf;charset=utf-
8;base64,AAEAAAKAIAAAwAgTlMvMj1tSfgAAAEoAAAAVmNtYXDnH+dzAAABoAAAAEJnbHlm1v4

```



```

8pAAAAfgAAQYAGVhZBOPfZcAAADQAAAAANmhoZWEIUQQJAAAArAAAACRobXR4IAAAAAAAAAAYAAAA
gbG9jYQN6ApQAAAHkAAAAEm1heHABFQCqAAABCAAAACBuYW1l071FxAABhAAAAIxcG9zdK9uovo
AAAhEAAAAGaABAAAEAAAAAFWEAAAAAAD9AABAAAAAAAAAAAAAAAAAAAAACAABAAAAAQAAJ1LUzF8
PPPUACwQAAAAAANG+nFMAAAAA2D6cUwAAAAAD9AP0AAAACAACAAAAAAAAAAAAEAAAAIAJ4AAwAAAA
AAgAAAAoACgAAAP8AAAAAAAAAAQAAZAAABQAAAokCzAAAAI8CiQLMAAAB6wAyAQgAAAIABQMAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAUGZFZABA5wDnBgQAAAAAXAQAAAAAABAAAAAABAAAAAQAAAA
EAAAABAAAAAQAAAAEAAAABAAAAAQAAAAAACAAAAwAAABQAAwABAAAAFAAEAC4AAAAEAAQAAQA
A5wb//wAA5wD//wAAAAEABAAAAEAAgADAAQABQAGAAcAAAAAAAAADgAkADIAhAEuAewCDAAAAAE
AAAAAA2ED9AACAAA3CQGeAsT9PAwB9AH0AAACAAAAAAPHa/QAAwAHAAALIREhASERIQJpAV7+ov3
QAV7+ogwD6PwYA+gAAAAEAAAAA4sD9AACAAATARF0AxgCAP4MA+gAAAAABAAAAAAP0A/QAQwAAExE
fDyE/DxEvDyEPDgwBAGMFBQcICQkLCwwMDQ4NAtONDg0MDAsLCQkIBwUFAwIBAQIDBQUHCAkJCws
MDA0ODf0mDQ4NDawLCwkJCACFBQMCA239Jg4NDQ0LCwsJCQgHBQUdAgEBAgMFBQcICQkLCwsNDQ0
OAtOODQ0NCwsLCQkIBwUFAwIBAQIDBQUHCAkJCwsLDQ0NAAIAAAAAA/MDxQADAIwAADczESMBDwM
VFw8METM3HwQ3Fz8KPQEvBT8LLwg3NT8INS8FNT8NNS8JBjU/BDUvCyMPAQytrQH5AgoEAQEBArg
hERESEyIJCsgQBiEHNQceOZPbDgUICw0LCQUDBAICBAkGAgEBAQMOBAkIBgcDAwEBAQEDAwMJAge
BAxYLBQQAeAwMCAgIEBAoBAQEECgcHBgUFBAMDAQEBAQQFBwkFBQUGef6tDwKEAwIBAQMDCgwVAwc
GDAsNBwdaAYcB3gEFAwN2HwoELDodGxwaLwkIGwz+igEBHwMBAQECAQEDBgoKDAYICagFCAkICwU
EBAQFAwYDBwgIDAgHCACGBgYFBQkEAgYCBawJBgUGBwkJCgkICacLBaIFawIEBAQFBQcGBwgHBgY
GBgoJCAYCagEBAQFGMRkaGw0NDA0LIh4xBAQCBAEBAgADAAAAAOKA/MAHABCAJ0AAAEzHwIRDwM
hLwIDNzM/CjUTHwcVIwcVIy8HETcXmZ8KNScxBxEfdjsSBHQefDTMhMz8OES8PIz0BLw4ha0EDBQQ
DAQIEbf5eBQQCAW4RDg0LCQgGBQUDBAFEBAMDawIBAQGL7Y0EawQCAgIBAYYKChEQDQsJCACBAU
CYt8BAQIDBAUFBQcHBwgICQgKjQECAGMEBAUFBgYHBgcIBwGcCAcHBwYGBgUFBAQDAgIBAQEBAgI
DBAQFBQYGBgCHBwgMAQMDawUFBgYHBwgICQkJ/tQCiwMEbf3XAwYEAgIEBgFoAQEDBQYGBwgIBw0
KhQEiAQEBAGMDawTV+94BAQECAwMDBAGyAQECBAYHCAgJCgkQCaQC6/47CQkICQcIBwYGBQQEAWI
CUAGHBwcGBgYFBQQAeAwMBAgIBAwMEBAUFBQcGBwCHCAImCAcHBwYGBgUFBAQDAgIBAdUJCQgICAg
GBwYFBAQDAgEBAAAAAIAAAAAA6cD9AADAawAADchNSElAQcJAScBESNZA078sgGB/uMuAXkBgDb
+1EwMTZcBCD3+ngFiPf7pAxMAAAAAABIA3gABAAAAAIAAAAAAABAAAAAABAAgAAQABAAAAAA
CAACACQABAAAAAADAAGAEABAAAAAEEAAGAGAABAAAAAFAAsAIAABAAAAAAGAAGAKwABAAA
AAAAKACwAMwABAAAAAALABIAxwADAAEECQAAAAIACQADAAEECQABABAacwADAAEECQACAA4AgwA
DAEEECQADABAakQADAAEECQAEABAAoQADAAEECQAFABYAsQADAAEECQAGABAaxwADAAEECQAKAFg
AlwADAAEECQALACQBLyBidG4taWNvblJlZ3VsYXJidG4taWNvbmJ0bi1pY29uVmVyc2lubiAxLjB
idG4taWNvbkZvbnQgZ2VuZXJhdGVkIHVzaW5nIFN5bmNmdXNpb24gTWV0cm8gU3R1ZG1vd3d3LnN
5bmNmdXNpb24uY29tACAAYgB0AG4ALQBpAGMAbwBuAFIAZQBnAHUAbABhAHIAyGBOAG4ALQBpAGM
AbwBuAGIAdABuAC0AaQBjAG8AbgBWAGUAcgBzAGkAbwBuACAAMQAuADAAYgB0AG4ALQBpAGMAbwB
uAEYAbwBuAHQAIABnAGUAbgBIAHIAyQB0AGUAZAAGAHUAcwBpAG4AZwAgAFMAeQBwAGMAZgB1AHM
AaQBvAG4AIAABNAGUAdABYAG8AIAABTAHQAdQBkAGkAbwB3AHcAdwAuAHMAeQBwAGMAZgB1AHMAaQB
vAG4ALgBjAG8AbQAAAAACAAAAAoooooAAAAAAAAAAAAAAAAAAAAAgBAGEDAQQBBQE
GAQcBCAEJAAPtZWRpYS1wbGF5C211ZG1hLXBhdXNlDmFycm93aGVhZC1sZWZ0BHN0b3AJbGlrZS0
tLTAXBGNvcHkQLWRvd25sb2FkLTAYLXdmLQAA) format('true');
font-weight: normal;
font-style: normal;
}
.e-btn-sb-icon {
font-family: 'btn-icon' !important;
speak: none;
font-size: 55px;
font-style: normal;
font-weight: normal;
font-variant: normal;
text-transform: none;
line-height: 1;
-webkit-font-smoothing: antialiased;
-moz-osx-font-smoothing: grayscale;
}
/*For Right Icon Button*/
.e-stop-icon::before {
content: '\e703';

```

```

}
/*For Left Icon Button*/
.e-prev-icon::before {
  content: '\e702';
}
</style>

```

{% previewsample "page.domainurl/code-snippet/button/default-cs16" %}

#### Button with SVG image

SVG image can be added to the Button using [iconCss](#) property.

In the following example, SVG image is added using the iconCss class `e-search-icon` by setting `height` and `width`.

#### APP.VUE

```

<template>
  <div>
    <ejs-button iconCss='e-search-icon'></ejs-button>
  </div>
</template>
<script>
import Vue from 'vue';
import { ButtonPlugin } from '@syncfusion/ej2-vue-buttons';
import { enableRipple } from '@syncfusion/ej2-base';
enableRipple(true);
Vue.use(ButtonPlugin);
export default {}
</script>
<style>
@import '../node_modules/@syncfusion/ej2-base/styles/material.css';
@import '../node_modules/@syncfusion/ej2-buttons/styles/material.css';
button {
  margin: 25px 5px 20px 20px;
}
.e-btn-icon.e-search-icon {
  background: url('../.../button/images/search_icon.svg');
  height: 25px;
  width: 25px;
}
</style>

```

{% previewsample "page.domainurl/code-snippet/button/default-cs17" %}

The Essential JS 2 provides a set of icons that can be loaded by applying `e-icons` class name to the element. You can also use third party icons on the Button using the [iconCss](#) property.

#### Button size

The two types of Button sizes are default and small. To change the size of the default Button to small Button,

set the [cssClass](#) property to `e-small`.

#### APP.VUE

```

<template>
  <div>
    <ejs-button cssClass='e-small'>Small</ejs-button>
    <ejs-button >Normal</ejs-button>
  </div>
</template>
<script>
import Vue from 'vue';
import { ButtonPlugin } from '@syncfusion/ej2-vue-buttons';
import { enableRipple } from '@syncfusion/ej2-base';
enableRipple(true);
Vue.use(ButtonPlugin);
export default {}
</script>
<style>
@import '../node_modules/@syncfusion/ej2-base/styles/material.css';
@import '../node_modules/@syncfusion/ej2-buttons/styles/material.css';
button {
  margin: 25px 5px 20px 20px;
}
</style>

```

{% previewsample "page.domainurl/code-snippet/button/default-cs18" %}

See Also

- [Customize Button appearance](#)
- [How to create block button](#)
- [How to create repeat button](#)

### Style and appearance in Vue Button component

To modify the Button appearance, you need to override the default CSS of Button component. Please find the list of CSS classes and its corresponding section in Button component. Also, you have an option to create your own custom theme for the controls using our [Theme Studio](#).

CSS Class | Purpose of Class

- |.e-btn|To customize the button.
- |.e-btn: hover|To customize the button on hover.
- |.e-btn: focus|To customize the button on focus.
- |.e-btn: active|To customize the button on active.

### Accessibility in Vue Button component

The Button component followed the accessibility guidelines and standards, including [ADA, Section 508](#), [WCAG 2.2](#) standards, and [WCAG roles](#) that are commonly used to evaluate accessibility.

The accessibility compliance for the Button component is outlined below.

| Accessibility Criteria | Compatibility |

| -- | -- |

| [WCAG 2.2](#) Support | `` |

| [Section 508](#) Support | `` |

| Screen Reader Support | `` |

| Right-To-Left Support | `` |

| Color Contrast | `` |

| Mobile Device Support | `` |

| Keyboard Navigation Support | `` |

| [Accessibility Checker](#) Validation | `` |

| [Axe-core](#) Accessibility Validation | `` |

`<style>`

`.post .post-content img {`

`display: inline-block;`

`margin: 0.5em 0;`

`}`

`</style>`

`<div>` - All features of the component meet the requirement.`</div>`

`<div>` - Some features of the component do not meet the requirement.`</div>`

`<div>` - The component does not meet the requirement.`</div>`

### WAI-ARIA attributes

The Button component followed the [WAI-ARIA](#) patterns to meet the accessibility. The following ARIA attributes are used in the Button component:

| Attributes | Purpose |

| --- | --- |

| `aria-label` | Provides an accessible name for the icon only button. |

### Keyboard interaction

The Button component followed the [keyboard interaction](#) guideline, making it easy for people who use assistive technologies (AT) and those who completely rely on keyboard navigation. The following keyboard shortcuts are supported by the Button component.

| **Press** | **To do this** |

| --- | --- |

| **Space** | When the button has focus, pressing the space key changes the state of the button. |

### Ensuring accessibility

The Button component's accessibility levels are ensured through an [accessibility-checker](#) and [axe-core](#) software tools during automated testing.

The accessibility compliance of the Button component is shown in the following sample. Open the [sample](#) in a new window to evaluate the accessibility of the Button component with accessibility tools.

```
{% previewsample "page.domainurl/code-snippet/button/default-cs1" %}
```

See also

- [Accessibility in Syncfusion Vue components](#)

### How To

#### Add link to a button in Vue Button component

The appearance of the Button can be changed like a link by `e-link` class using [cssClass](#) property and link navigation can be handled in Button click.

In the following example, link is added in Button click by using `window.open()` method.

#### APP.VUE

```
<template>
<ejs-button cssClass='e-link' v-on:click.native='btnClick'>Button</ejs-
button>
</template>
<script>
import Vue from 'vue';
import { ButtonPlugin } from '@syncfusion/ej2-vue-buttons';
import { enableRipple } from '@syncfusion/ej2-base';
enableRipple(true);
Vue.use(ButtonPlugin);
export default {
  methods : {
    btnClick: function(event) {
      window.open("https://www.google.com");
    }
  }
}
</script>
<style>
@import '../node_modules/@syncfusion/ej2-base/styles/material.css';
@import '../node_modules/@syncfusion/ej2-buttons/styles/material.css';
button {
  margin: 25px 5px 20px 20px;
```

```
}
</style>
```

```
{% previewsample "page.domainurl/code-snippet/button/default-cs3" %}
```

### Create a block button in Vue Button component

You can customize a Button into a Block Button that will span the entire width of its parent element. To create a Block Button, set the [cssClass](#) property to `e-block`.

#### APP.VUE

```
<template>
<div>
  <ejs-button cssClass='e-block'>Block Button</ejs-button>
  <ejs-button cssClass='e-block' isPrimary=true>Block Button</ejs-button>
  <ejs-button cssClass='e-block e-success'>Block Button</ejs-button>
</div>
</template>
<script>
import Vue from 'vue';
import { ButtonPlugin } from '@syncfusion/ej2-vue-buttons';
import { enableRipple } from '@syncfusion/ej2-base';
enableRipple(true);
Vue.use(ButtonPlugin);
export default {}
</script>
<style>
@import '../node_modules/@syncfusion/ej2-base/styles/material.css';
@import '../node_modules/@syncfusion/ej2-buttons/styles/material.css';
button {
  margin: 25px 0;
}
</style>
```

```
{% previewsample "page.domainurl/code-snippet/button/default-cs4" %}
```

### Customize button appearance in Vue Button component

You can customize the appearance of the Button by using the Cascading Style Sheets (CSS). Define the CSS according to your requirement, and assign the class name to the [cssClass](#) property. In the following code snippet the background color, text color, height, width, and sharp corner of the Button can be customized through the `e-custom` class for all states (hover, focus, and active).

#### APP.VUE

```
<template>
  <ejs-button cssClass='e-custom'>Custom</ejs-button>
</template>
<script>
import Vue from 'vue';
import { ButtonPlugin } from '@syncfusion/ej2-vue-buttons';
import { enableRipple } from '@syncfusion/ej2-base';
enableRipple(true);
Vue.use(ButtonPlugin);
export default {}
</script>
```

```

<style>
@import '../node_modules/@syncfusion/ej2-base/styles/material.css';
@import '../node_modules/@syncfusion/ej2-buttons/styles/material.css';
.e-custom {
  border-radius: 0;
  height: 30px;
  width: 80px;
}
.e-custom, .e-custom:hover, .e-custom:focus, .e-custom:active {
  background-color: #ff6e40;
  color: #fff;
}
button {
  margin: 25px 5px 20px 20px;
}
</style>

```

{% previewsample "page.domainurl/code-snippet/button/default-cs6" %}

### Customize input and anchor elements in Vue Button component

You can customize the appearance of the input and anchor elements using predefined styles through the class property. In the following code snippet, the input element is customized as a link Button by setting the `e-btn e-link` class, and the anchor element is customized as a primary Button by setting the `e-btn e-primary` class.

#### APP.VUE

```

<template>
<div>
  <div>
    <input type='button' value='Input Button' class='e-btn e-link' />
  </div>
  <br>
  <div>
    <a id='anchorbtn' class='e-btn e-primary' href='#'>Google</a>
  </div>
</div>
</template>
<script>
import Vue from 'vue';
import { ButtonPlugin } from '@syncfusion/ej2-vue-buttons';
import { enableRipple } from '@syncfusion/ej2-base';
enableRipple(true);
Vue.use(ButtonPlugin);
export default {}
</script>
<style>
  @import '../node_modules/@syncfusion/ej2-base/styles/material.css';
  @import '../node_modules/@syncfusion/ej2-buttons/styles/material.css';
</style>

```

{% previewsample "page.domainurl/code-snippet/button/default-cs7" %}

### Repeat button in Vue Button component

The Repeat button is a type of Button in that the click event is triggered at regular time interval from the pressed state till the released state.

The following example explains about how to achieve Repeat Button in mouse and touch events.

#### APP.VUE

```
<template>
  <div id='container'>
    <div class='btncontainer'>
      <ejs-button id='button' cssClass='e-small'>Button</ejs-button>
    </div>
    <div class='event' style='height:auto;'>
      <table title='Event Trace' style='width:100%'>
        <tbody>
          <tr>
            <th>Event Trace</th>
          </tr>
          <tr>
            <td>
              <div class='eventarea' style='height: 250px;overflow:
auto'>
                <span id='eventlog' style='word-break:
normal;'></span>
              </div>
            </td>
            <td>
              <div class='evtbtn' style='padding:20px 0 0 20px'>
                <ejs-button id='clear' cssClass='e-
small'>Clear</ejs-button>
              </div>
            </td>
          </tr>
        </tbody>
      </table>
    </div>
  </div>
</template>
<script>
import Vue from 'vue';
import { ButtonPlugin } from '@syncfusion/ej2-vue-buttons';
import { enableRipple, EventHandler } from '@syncfusion/ej2-base';
enableRipple(true);
Vue.use(ButtonPlugin);
export default {
  mounted () {
    var timeout;
    document.getElementById('clear').onclick = function () {
      document.getElementById('eventlog').innerHTML = '';
    }
    var button = document.getElementById('button');
    button.addEventListener('mousedown', mouseDownHandler);
    button.addEventListener('touchstart', mouseDownHandler);
    button.addEventListener('mouseup', mouseUpHandler);
```



```

button.addEventListener('touchend', mouseUpHandler);
button.addEventListener('click', clickEventHandler);
function mouseUpHandler() {
    clearInterval(timeout);
}
function mouseDownHandler(event) {
    event.preventDefault();
    timeout = setInterval(clickEventHandler, 200);
}
function clickEventHandler() {
    EventHandler.trigger(button, 'click');
    appendSpanElement('Button click event triggered.<hr>');
}
function appendSpanElement(text) {
    var span = document.createElement('span');
    span.innerHTML = text;
    var log = document.getElementById('eventlog');
    log.insertBefore(span, log.firstChild);
}
}
}
</script>
<style>
@import '../node_modules/@syncfusion/ej2-base/styles/material.css';
@import '../node_modules/@syncfusion/ej2-buttons/styles/material.css';
.btncontainer {
    float: left;
    width: 40%;
}
.event {
    float: right;
    width: 60%;
    border-left: 1px solid #D7D7D7;
}
#eventlog b {
    color: #388e3c;
}
hr {
    margin: 1px 10px 1px 0px;
    border-top: 1px solid #eee;
}
</style>

```

```
{% previewsample "page.domainurl/code-snippet/button/default-cs8" %}
```

Right to left in Vue Button component

Button component has RTL support. This can be achieved by setting [enableRtl](#) as

**true**.

The following example illustrates how to enable right-to-left support in Button component.

#### APP.VUE

```

<template>
    <ejs-button iconCss='e-icons e-setting-icon'
    enableRtl=true>Settings</ejs-button>

```

```

</template>
<script>
import Vue from 'vue';
import { ButtonPlugin } from '@syncfusion/ej2-vue-buttons';
import { enableRipple } from '@syncfusion/ej2-base';
enableRipple(true);
Vue.use(ButtonPlugin);
export default {}
</script>
<style>
@import '../node_modules/@syncfusion/ej2-base/styles/material.css';
@import '../node_modules/@syncfusion/ej2-buttons/styles/material.css';
.e-setting-icon::before {
  content: '\e434';
}
button {
  margin: 25px 5px 20px 20px;
}
</style>

```

{% previewsample "page.domainurl/code-snippet/button/default-cs9" %}

#### Set the disabled state in Vue Button component

Vue Button component can be enabled/disabled by giving [disabled](#) property. To disable Vue Button component, the `disabled` property can be set as `true`.

The following example demonstrates Button in `disabled` state.

#### APP.VUE

```

<template>
  <ejs-button disabled=true>Disabled</ejs-button>
</template>
<script>
import Vue from 'vue';
import { ButtonPlugin } from '@syncfusion/ej2-vue-buttons';
import { enableRipple } from '@syncfusion/ej2-base';
enableRipple(true);
Vue.use(ButtonPlugin);
export default {}
</script>
<style>
@import '../node_modules/@syncfusion/ej2-base/styles/material.css';
@import '../node_modules/@syncfusion/ej2-buttons/styles/material.css';
button {
  margin: 25px 5px 20px 20px;
}
</style>

```

{% previewsample "page.domainurl/code-snippet/button/default-cs10" %}

#### Tooltip for button in Vue Button component

Tooltip can be shown on Button hover and it can be achieved by setting `title` attribute.

The following snippets illustrates how to show tooltip on Button hover.

**APP.VUE**

```

<template>
  <ejs-button id='button' isPrimary=true>Button</ejs-button>
</template>
<script>
import Vue from 'vue';
import { ButtonPlugin } from '@syncfusion/ej2-vue-buttons';
import { enableRipple } from '@syncfusion/ej2-base';
enableRipple(true);
Vue.use(ButtonPlugin);
export default {
  mounted () {
    document.getElementById('button').setAttribute('title', 'Primary
Button');
  }
}
</script>
<style>
@import '../node_modules/@syncfusion/ej2-base/styles/material.css';
@import '../node_modules/@syncfusion/ej2-buttons/styles/material.css';
button {
  margin: 25px 5px 20px 20px;
}
</style>

```

{% previewsample "page.domainurl/code-snippet/button/default-cs11" %}

## Calendar

### Getting Started with the Vue Calendar Component in Vue 2

This article provides a step-by-step guide for setting up a Vue 2 project using [Vue-CLI](#) and integrating the Syncfusion Vue Calendar component using the [Composition API](#) / [Options API](#)

#### Prerequisites

[System requirements for Syncfusion Vue UI components](#)

#### Dependencies

The list of dependencies required to use the Calendar component in your application is given below:

```

`javascript
|-- @syncfusion/ej2-vue-calendars
|-- @syncfusion/ej2-base
|-- @syncfusion/ej2-data
|-- @syncfusion/ej2-vue-base
|-- @syncfusion/ej2-calendars
|-- @syncfusion/ej2-inputs
|-- @syncfusion/ej2-splitbuttons
|-- @syncfusion/ej2-lists

```

```
|-- @syncfusion/ej2-popups  
|-- @syncfusion/ej2-buttons  
,
```

### Setting up the Vue 2 project

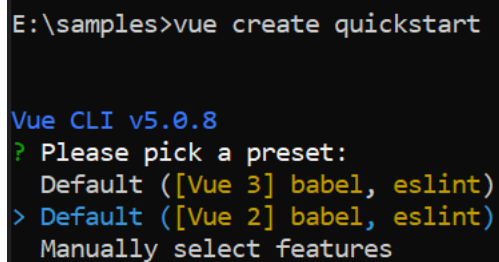
To generate a Vue 2 project using Vue-CLI, use the [vue create](#) command. Follow these steps to install Vue CLI and create a new project:

```
`bash  
npm install -g @vue/cli  
vue create quickstart  
cd quickstart  
npm run serve  
,
```

or

```
`bash  
yarn global add @vue/cli  
vue create quickstart  
cd quickstart  
yarn run serve  
,
```

When creating a new project, choose the option **Default ([Vue 2] babel, eslint)** from the menu.



```
E:\samples>vue create quickstart  
  
Vue CLI v5.0.8  
? Please pick a preset:  
  Default ([Vue 3] babel, eslint)  
> Default ([Vue 2] babel, eslint)  
  Manually select features
```

Once the **quickstart** project is set up with default settings, proceed to add Syncfusion components to the project.

### Adding Syncfusion packages

Syncfusion packages are available at [npmjs.com](https://www.syncfusion.com). To use Vue components, install the required npm package.

This article uses the [Vue Calendar component](#) as an example. Install the **@syncfusion/ej2-vue-calendars** package by running the following command:

```
`bash
```

```
npm install @syncfusion/ej2-vue-calendars --save
```

```
,
```

```
or
```

```
`bash
```

```
yarn add @syncfusion/ej2-vue-calendars
```

```
,
```

### Import Syncfusion CSS styles

You can import themes for the Syncfusion Vue component in various ways, such as using CSS or SASS styles from npm packages, CDN, [CRG](#) and [Theme Studio](#). Refer to [themes topic](#) to know more about built-in themes and different ways to refer to themes in a Vue project.

In this article, the **Material** theme is applied using CSS styles, which are available in installed packages. The necessary **Material** CSS styles for the Calendar component and its dependents were imported into the `<style>` section of **src/App.vue** file.

#### ~/SRC/APP.VUE

```
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-buttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-calendars/styles/material.css";
</style>
```

### Add Syncfusion Vue component

Follow the below steps to add the Vue Calendar component using **Composition API** or **Options API**:

1\ First, import and register the Calendar component in the **script** section of the **src/App.vue** file. If you are using the **Composition API**, you should add the **setup** attribute to the **script** tag to indicate that Vue will be using the **Composition API**.

#### COMPOSITION API (~/SRC/APP.VUE)

```
<script setup>
import { CalendarComponent as EjsCalendar } from '@syncfusion/ej2-vue-
calendars';
</script>
```

#### OPTIONS API (~/SRC/APP.VUE)

```
<script>
import { CalendarComponent } from '@syncfusion/ej2-vue-calendars';
export default {
  components: {
    'ejs-calendar': CalendarComponent
  }
}
</script>
```

2\ In the **template** section, define the Calendar component.

**(~/SRC/APP.VUE)**

```
<template>
<div id="app">
<ejs-calendar ></ejs-calendar>
</div>
</template>
```

Here is the summarized code for the above steps in the **src/App.vue** file:

**COMPOSITION API (~/SRC/APP.VUE)**

```
<template>
  <div id="app">
    <div class='wrapper'>
      <ejs-calendar ></ejs-calendar>
    </div>
  </div>
</template>
<script setup>
import { CalendarComponent as EjsCalendar } from '@syncfusion/ej2-vue-
calendars';
</script>
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-buttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-calendars/styles/material.css";
.wrapper {
  max-width: 250px;
  margin: 0 auto;
}
</style>
```

**OPTIONS API (~/SRC/APP.VUE)**

```
<template>
  <div id="app">
    <div class='wrapper'>
      <ejs-calendar ></ejs-calendar>
    </div>
  </div>
</template>
<script>
import { CalendarComponent } from '@syncfusion/ej2-vue-calendars';
export default {
  components: {
    'ejs-calendar': CalendarComponent
  }
}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-buttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-calendars/styles/material.css";
.wrapper {
  max-width: 250px;
}
```

```
margin: 0 auto;
}
</style>
```

### Running the Project

To run the project, use the following command:

```
`bash
```

```
npm run serve
```

```
,
```

```
or
```

```
`bash
```

```
yarn run serve
```

```
,
```

```
{% previewsample "page.domainurl/code-snippet/calendar/getting-started-cs1" %}
```

Setting the value, min and max dates

The following example demonstrates how to set the value, min and max dates on initializing the Calendar. Here the Calendar allows you to select a date within the range from 9th to 15th in the month of May 2017. To know more about range restriction in Calendar, please refer this [page](#).

### COMPOSITION API (~SRC/APP.VUE)

```
<template>
  <div id="app">
    <div class='wrapper'>
      <ejs-calendar :min="minDate" :max="maxDate" :value="dateVal" ></ejs-
calendar>
    </div>
  </div>
</template>
<script setup>
import { CalendarComponent as EjsCalendar } from '@syncfusion/ej2-vue-
calendars';
const minDate = new Date("05/09/2017");
const maxDate = new Date("05/15/2017");
const dateVal = new Date("05/11/2017");

</script>
<style>
  @import "../node_modules/@syncfusion/ej2-base/styles/material.css";
  @import "../node_modules/@syncfusion/ej2-buttons/styles/material.css";
  @import "../node_modules/@syncfusion/ej2-vue-
calendars/styles/material.css";
  .wrapper {
    max-width: 250px;
    margin: 0 auto;
  }
</style>
```

**OPTIONS API (~SRC/APP.VUE)**

```

<template>
  <div id="app">
    <div class='wrapper'>
      <ejs-calendar :min="minDate" :max="maxDate" :value="dateVal" ></ejs-
calendar>
    </div>
  </div>
</template>
<script>
import { CalendarComponent } from '@syncfusion/ej2-vue-calendars';
export default {
  components: {
    'ejs-calendar': CalendarComponent
  },
  data () {
    return {
      minDate : new Date("05/09/2017"),
      maxDate : new Date("05/15/2017"),
      dateVal : new Date("05/11/2017")
    }
  }
}
</script>
<style>
  @import "../node_modules/@syncfusion/ej2-base/styles/material.css";
  @import "../node_modules/@syncfusion/ej2-buttons/styles/material.css";
  @import "../node_modules/@syncfusion/ej2-vue-
calendars/styles/material.css";
  .wrapper {
    max-width: 250px;
    margin: 0 auto;
  }
</style>

```

{% previewsample "page.domainurl/code-snippet/calendar/min-max-cs7" %}

See Also

- [Select multiple dates in the Calendar](#)
- [Render Calendar with specific culture](#)
- [How to change the initial view of the Calendar](#)
- [Render Calendar with week numbers](#)
- [Show other month dates](#)

### Getting Started with the Vue Calendar Component in Vue 3

This article provides a step-by-step guide for setting up a [Vite](#) project with a JavaScript environment and integrating the Syncfusion Vue Calendar component using the [Composition API](#) / [Options API](#).

The **Composition API** is a new feature introduced in Vue.js 3 that provides an alternative way to organize and reuse component logic. It allows developers to write components as functions that use smaller, reusable functions called composition functions to manage their properties and behavior.



The **Options API** is the traditional way of writing Vue.js components, where the component logic is organized into a series of options that define the component's properties and behavior. These options include data, methods, computed properties, watchers, lifecycle hooks, and more.

#### Prerequisites

##### [System requirements for Syncfusion Vue UI components](#)

#### Set up the Vite project

A recommended approach for beginning with Vue is to scaffold a project using [Vite](#). To create a new Vite project, use one of the commands that are specific to either NPM or Yarn.

```
`bash
```

```
npm create vite@latest
```

```
`
```

or

```
`bash
```

```
yarn create vite
```

```
`
```

Using one of the above commands will lead you to set up additional configurations for the project as below:

1. Define the project name: We can specify the name of the project directly. Let's specify the name of the project as **my-project** for this article.

```
`bash
```

```
? Project name: » my-project
```

```
`
```

2. Select **Vue** as the framework. It will create a Vue 3 project.

```
`bash
```

```
? Select a framework: » - Use arrow-keys. Return to submit.
```

```
Vanilla
```

```
Vue
```

```
React
```

```
Preact
```

```
Lit
```

```
Svelte
```

```
Others
```

```
`
```

3. Choose **JavaScript** as the framework variant to build this Vite project using JavaScript and Vue.

```
`bash
```

? Select a variant: » - Use arrow-keys. Return to submit.

JavaScript

TypeScript

Customize with create-vue ↗

Nuxt ↗

,

4. Upon completing the aforementioned steps to create the `my-project`, run the following command to install its dependencies:

```
`bash
```

```
cd my-project
```

```
npm install
```

,

or

```
`bash
```

```
cd my-project
```

```
yarn install
```

,

Now that `my-project` is ready to run with default settings, let's add Syncfusion components to the project.

#### Add Syncfusion Vue packages

Syncfusion Vue component packages are available at [npmjs.com](https://www.npmjs.com). To use Syncfusion Vue components in the project, install the corresponding npm package.

This article uses the [Vue Calendar component](#) as an example. To use the Vue Calendar component in the project, the `@syncfusion/ej2-vue-calendars` package needs to be installed using the following command:

```
`bash
```

```
npm install @syncfusion/ej2-vue-calendars --save
```

,

or

```
`bash
```

```
yarn add @syncfusion/ej2-vue-calendars
```

,

#### Import Syncfusion CSS styles

You can import themes for the Syncfusion Vue component in various ways, such as using CSS or SASS styles from npm packages, CDN, [CRG](#) and [Theme Studio](#). Refer to [themes topic](#) to know more about built-in themes and different ways to refer to themes in a Vue project.

In this article, **Material** theme is applied using CSS styles, which are available in installed packages. The necessary **Material** CSS styles for the Calendar component and its dependents were imported into the `<style>` section of **src/App.vue** file.

#### ~/SRC/APP.VUE

```
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-buttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-calendars/styles/material.css";
</style>
```

The order of importing CSS styles should be in line with its dependency graph.

#### Add Syncfusion Vue component

Follow the below steps to add the Vue Calendar component using **Composition API** or **Options API**:

1.First, import and register the Calendar component in the **script** section of the **src/App.vue** file. If you are using the **Composition API**, you should add the **setup** attribute to the **script** tag to indicate that Vue will be using the **Composition API**.

#### COMPOSITION API (~/SRC/APP.VUE)

```
<script setup>
import { CalendarComponent as EjsCalendar } from '@syncfusion/ej2-vue-
calendars';
</script>
```

#### OPTIONS API (~/SRC/APP.VUE)

```
<script>
import { CalendarComponent } from "@syncfusion/ej2-vue-calendars";
//Component registration
export default {
  name: "App",
  components: {
    'ejs-calendar' : CalendarComponent,
  }
}
</script>
```

2.In the **template** section, define the Calendar component with the [dataSource](#) property and column definitions.

#### ~/SRC/APP.VUE

```
<template>
<div class="control_wrapper">
<ejs-calendar></ejs-calendar>
</div>
</template>
```

Here is the summarized code for the above steps in the **src/App.vue** file:

#### COMPOSITION API (~/SRC/APP.VUE)

```
<template>
<div class="control_wrapper">
<ejs-calendar></ejs-calendar>
</div>
</template>
<script setup>
import { CalendarComponent as EjsCalendar } from '@syncfusion/ej2-vue-
calendars';
</script>
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-buttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-calendars/styles/material.css";
.control_wrapper {
max-width: 250px;
margin: 0 auto;
}
</style>
```

### OPTIONS API (~SRC/APP.VUE)

```
<template>
<div class="control_wrapper">
<ejs-calendar></ejs-calendar>
</div>
</template>
<script>
import { CalendarComponent } from "@syncfusion/ej2-vue-calendars";
//Component registration
export default {
name: 'App',
components: {
"ejs-calendar": CalendarComponent
},
}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-buttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-calendars/styles/material.css";
.control_wrapper {
max-width: 250px;
margin: 0 auto;
}
</style>
```

### Run the project

To run the project, use the following command:

```
`bash
```

```
npm run dev
```

```
,
```

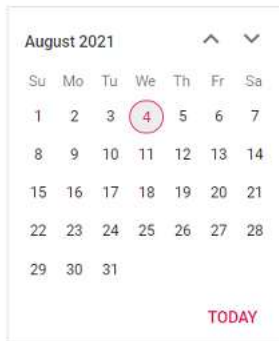
or

```
`bash
```

```
yarn run dev
```

```
,
```

The output will appear as follows:



### Setting the value, min and max dates

The following example demonstrates how to set the value, min and max dates on initializing the Calendar. Here the Calendar allows you to select a date within the range from 9th to 15th in the month of May 2017. To know more about range restriction in Calendar, please refer this [page](#).

#### **COMPOSITION API (~SRC/APP.VUE)**

```
<template>
<div id="app">
  <div class='wrapper'>
    <ejs-calendar :min="data[0].minDate" :max="data[0].maxDate"
      :value="data[0].dateVal" ></ejs-calendar>
  </div>
</div>
</template>
<script setup>
import { CalendarComponent as EjsCalendar } from "@syncfusion/ej2-vue-
calendars";
const data = [{minDate: new Date("05/04/2017"),
maxDate: new Date("05/16/2017"),
dateVal: new Date("05/10/2017")}];
</script>
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-buttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-calendars/styles/material.css";
.wrapper {
max-width: 250px;
margin: 0 auto;
}
</style>
```

#### **OPTIONS API (~SRC/APP.VUE)**

```
<template>
<div id="app">
<div class='wrapper'>
```

```

<ejs-calendar :min="minDate" :max="maxDate" :value="dateVal" ></ejs-
calendar>
</div>
</div>
</template>
<script>
import { CalendarComponent } from "@syncfusion/ej2-vue-calendars";
//Component registration
export default {
name: 'App',
components: {
"ejs-calendar": CalendarComponent
},
data () {
return {
minDate : new Date("05/09/2017"),
maxDate : new Date("05/15/2017"),
dateVal : new Date("05/11/2017")
}
}
}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-buttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-calendars/styles/material.css";
.wrapper {
max-width: 250px;
margin: 0 auto;
}
</style>

```

The output will appear as follows:



See Also

- [Select multiple dates in the Calendar](#)
- [Render Calendar with specific culture](#)
- [How to change the initial view of the Calendar](#)
- [Render Calendar with week numbers](#)
- [Show other month dates](#)

## Date range in Vue Calendar component

Calendar provides an option to select a date value within a specified range by defining the min and max properties. The min date should always be lesser than the max date. If the value of `min` or `max` properties are changed through code behind, then update the `value` property to be set within the specified range. Or else, if the value is out of specified date range and less than min date, value property will be updated with the `min` date or, if the value is higher than max date, value property will be updated with the `max` date.

The following example allows you to select a date within the range of 7th to 27th days in a month.

### APP.VUE

```
<template>
  <div id="app">
    <div class='wrap'>
      <ejs-calendar id='calendar' :value='dateVal' :min='minVal'
:max='maxVal' placeholder='Select a Date'></ejs-calendar>
    </div>
  </div>
</template>
<script>
import Vue from "vue";
import { CalendarPlugin } from "@syncfusion/ej2-vue-calendars";
Vue.use(CalendarPlugin);
export default {
  data () {
    var today = new Date();
    var currentYear = today.getFullYear();
    var currentMonth = today.getMonth();
    var currentDay = today.getDate();
    return {
      dateVal: new Date(new Date().setDate(14)),
      minVal: new Date(currentYear, currentMonth, 7),
      maxVal: new Date(currentYear, currentMonth, 27),
    }
  }
}
</script>
<style>
  @import "../node_modules/@syncfusion/ej2-base/styles/material.css";
  @import "../node_modules/@syncfusion/ej2-buttons/styles/material.css";
  @import "../node_modules/@syncfusion/ej2-vue-
calendars/styles/material.css";
  .wrap {
    margin: 35px auto;
    width: 240px;
  }
</style>
```

{% previewsample "page.domainurl/code-snippet/calendar/min-max-cs6" %}

## Globalization in Vue Calendar component

Globalization is the combination of internationalization and localization. You can adapt the component to various languages by parsing and formatting the date or number ([Internationalization](#)), and also add culture specific customization and translation to the text ([localization](#)).

By default, the Calendar date format, week, and month names are specific to American English culture. It uses the [Essential JavaScript 2 Internationalization](#) package to parse and format date object based on the culture using the official [UNICODE CLDR](#) JSON data. It provides the [loadCldr](#) method to load the culture-specific CLDR JSON data.

The Calendar component supports only the Gregorian type of calendar. All the Essential JS 2 component are specific to English culture ('en-US').

If you want to go with the different culture other than **English**, follow the below steps.

- Install the CLDR-Data package by using the below command (installs the CLDR JSON data).

To know more about CLDR data, refer to the [CLDR-Data](#) link.

```
npm install cldr-data --save
```

Once the package installed, you can find the culture specific JSON data under the location `/node_modules/cldr-data`.

- Now use the `loadCldr` method to load the culture specific CLDR JSON data from the installed location to `app.vue` file.
- Calendar displayed **Sunday** as the first day of week based on default culture ("en-US"). If you want to display the Calendar with loaded culture's first day of week, you need to import `weekdata.json` file from the `cldr-data/supplemental` as given in the code example.

```
`ts
```

```
//Load the loadCldr from ej2-base
```

```
import { loadCldr } from '@syncfusion/ej2-base';
```

```
import * as numberingSystems from 'cldr-data/supplemental/numberingSystems.json';
```

```
import * as gregorian from 'cldr-data/main/de/ca-gregorian.json';
```

```
import * as numbers from 'cldr-data/main/de/numbers.json';
```

```
import * as timeZoneNames from 'cldr-data/main/de/timeZoneNames.json';
```

```
import * as weekData from 'cldr-data/supplemental/weekdata.json'; // To load the culture based first day of week
```

```
loadCldr(numberingSystems, gregorian, numbers, timeZoneNames, weekData);
```



The **Localization** library allows you to localize default text content of the Calendar. The Calendar component has static text for **today** feature that can be changed to other cultures (Arabic, Deutsch, French, etc.) by defining the [locale](#) value and translation object.

Locale keywords | Text

today | Name of the button to choose Today date.

- Before changing the culture other than **English**, ensure that locale text for the concerned culture is loaded through **load** method of

[L10n](#) class.

```
`ts
//Load the L10n from ej2-base
import { L10n } from '@syncfusion/ej2-base';
//load the locale object to set the localized today value
L10n.load({
  'de': {
    'calendar': { today: 'heute' }
  }
});
`
```

- Set the culture by using the [locale](#) property.

The following example demonstrates the Calendar in **German** culture

#### **APP.VUE**

```
<template>
  <div id="app">
    <div class='wrap'>
      <ejs-calendar id='calendar' locale='de'></ejs-calendar>
    </div>
  </div>
</template>
<script>
import Vue from "vue";
import { loadCldr, L10n } from '@syncfusion/ej2-base';
import { CalendarPlugin } from "@syncfusion/ej2-vue-calendars";
// Here we have referred local json files for preview purpose
import * as numberingSystems from './numberingSystems.json';
import * as gregorian from './ca-gregorian.json';
import * as numbers from './numbers.json';
import * as timeZoneNames from './timeZoneNames.json';
Vue.use(CalendarPlugin);
loadCldr(numberingSystems, gregorian, numbers, timeZoneNames);
L10n.load({
```

```

    'de': {
      'calendar': { today: 'heute' }
    }
  });
export default {}
</script>
<style>
  @import "../node_modules/@syncfusion/ej2-base/styles/material.css";
  @import "../node_modules/@syncfusion/ej2-buttons/styles/material.css";
  @import "../node_modules/@syncfusion/ej2-vue-
  calendars/styles/material.css";
  .wrap {
    margin: 0px auto;
    width: 240px;
  }
</style>

```

{% previewsample "page.domainurl/code-snippet/calendar/internationalization-cs1" %}

### Right-to-Left

The Calendar supports right-to-left functionality for languages like Arabic, Hebrew, etc. To display text in the right-to-left direction, use

[enableRtl](#) property.

The following code example initializes the Calendar component in **Arabic** culture.

### APP.VUE

```

<template>
  <div id="app">
    <div class='wrap'>
      <ejs-calendar id='calendar' locale='ar' enableRtl='true'></ejs-
      calendar>
    </div>
  </div>
</template>
<script>
import Vue from "vue";
import { loadCldr, L10n } from '@syncfusion/ej2-base';
import { CalendarPlugin } from "@syncfusion/ej2-vue-calendars";
// Here we have referred local json files for preview purpose
import * as numberingSystems from './numberingSystems.json';
import * as gregorian from './ca-gregorian.json';
import * as numbers from './numbers.json';
import * as timeZoneNames from './timeZoneNames.json';
Vue.use(CalendarPlugin);
loadCldr(numberingSystems, gregorian, numbers, timeZoneNames);
L10n.load({
  'ar': {
    'calendar': { today: "اليوم" }
  }
});
export default {}
</script>
<style>
  @import "../node_modules/@syncfusion/ej2-base/styles/material.css";

```

```

@import "../node_modules/@syncfusion/ej2-buttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-
calendars/styles/material.css";
.wrap {
    margin: 0px auto;
    width: 240px;
}
</style>

```

{% previewsample "page.domainurl/code-snippet/calendar/rtl-cs1" %}

### Customization in Vue Calendar component

Each day cell of the Calendar can be customized by using the [renderDayCell](#)

event.

The following section demonstrates how to disable or highlight specific dates in the Calendar.

#### Disable weekends

You can disable weekends of every month in the Calendar by using the [renderDayCell](#) event. The [renderDayCell](#) event offers the following arguments on each day cell creation to help you disable the dates.

| **View** | **Description** |

| --- | --- |

| **date** | Defines the current date of the Calendar. |

| **isDisabled** | Specifies whether the current date is to be disabled or not. |

| **isOutOfRange** | Defines whether the current date is out of range or not. |

The following example demonstrates how to disable weekends of every month.

#### APP.VUE

```

<template>
  <div id="app">
    <div class='wrap'>
      <ejs-calendar id='calendar' :renderDayCell="disableDate" ></ejs-
calendar>
    </div>
  </div>
</template>
<script>
import Vue from "vue";
import { CalendarPlugin } from "@syncfusion/ej2-vue-calendars";
Vue.use(CalendarPlugin);
export default {
  methods: {
    disableDate: function(args) {
      if (args.date.getDay() === 0 || args.date.getDay() === 6) {
        args.isDisabled = true;
      }
    }
  }
}

```

```

</script>
<style>
  @import "../node_modules/@syncfusion/ej2-base/styles/material.css";
  @import "../node_modules/@syncfusion/ej2-buttons/styles/material.css";
  @import "../node_modules/@syncfusion/ej2-vue-
  calendars/styles/material.css";
  .wrap {
    margin: 35px auto;
    width: 240px;
  }
</style>

```

{% previewsample "page.domainurl/code-snippet/calendar/min-max-cs4" %}

### Day cell format

You can also highlight specific dates by adding custom CSS or element to the day cell by using the [renderDayCell](#) event. You can customize the appearance of the Calendar by overriding the existing styles. The following list of CSS class names are used to customize the Calendar component.

Class Name	Description
---   ---	
e-calendar	Applied to the Calendar.
e-header	Applied to the header.
e-title	Applied to the title.
e-icon-container	Applied to the previous and next icon container.
e-prev	Applied to the previous icon.
e-next	Applied to the next icon.
e-weekend	Applied to weekends.
e-other-month	Applied to days of other months.
e-day	Applied to each day cell.
e-selected	Applied to the selected dates.
e-disabled	Applied to the disabled dates.

The following example highlights the World Health Day (every 7th April) and World Forest Day (every 21st March) by using the custom icon and ToolTip.

### APP.VUE

```

<template>
  <div id="app">
    <div class='wrap'>
      <ejs-calendar id='calendar' :renderDayCell='load' ></ejs-
calendar>
    </div>
  </div>
</template>
<script>
import Vue from "vue";

```

```

import { CalendarPlugin } from "@syncfusion/ej2-vue-calendars";
import { addClass } from '@syncfusion/ej2-base';
Vue.use(CalendarPlugin);
export default {
  methods: {
    load: function(args) {
      let span;
      //Defines the custom HTML element to be added.
      span = document.createElement('span');
      //Uses "e-icons" class name to load Essential JS 2 built-in icons.
      span.setAttribute('class', 'e-icons highlight-day');
      if (+args.date.getDate() === 7 && +args.date.getMonth() === 3) {
        //Appends the span element to day cell.
        args.element.appendChild(span);
        //Sets the custom tooltip to the special dates.
        args.element.setAttribute('title', 'World health day!');
        //Uses "special" class name to highlight special dates that you can
        refer in "styles.css".
        args.element.className = 'special';
      }
      if (+args.date.getDate() === 21 && +args.date.getMonth() === 2) {
        args.element.appendChild(span);
        args.element.className = 'special';
        //Sets the custom tooltip to the special dates.
        args.element.setAttribute('title', 'World forest day');
      }
    }
  }
}
</script>
<style>
  @import "../node_modules/@syncfusion/ej2-base/styles/material.css";
  @import "../node_modules/@syncfusion/ej2-buttons/styles/material.css";
  @import "../node_modules/@syncfusion/ej2-vue-
calendars/styles/material.css";
  .wrap {
    margin: 35px auto;
    width: 240px;
  }
  /* Custom - styles */
  span.special.e-day.e-ripple-style { /* csslint allow: adjoining-classes*/
    font-weight: bold;
    color: red;
  }
  span.e-icons.highlight-day { /* csslint allow: adjoining-classes*/
    color: red;
    margin-top: -13px;
    display: block;
    margin-left: 4px;
  }
  span.e-icons.highlight-day, /* csslint allow: adjoining-classes*/
  span.e-icons.highlight-day:before { /* csslint allow: adjoining-classes*/
    color: rgb(0, 0, 255);
  }
  span.e-icons.highlight-day:before { /* csslint allow: adjoining-classes*/
    content: "\e190";
    vertical-align: middle;
    margin-right: 3px;
  }

```

```

    font-size: 4px;
    position: relative;
    top: 1px;
    font-weight: normal;
}
span.special.e-day.e-ripple-style { /* csslint allow: adjoining-classes*/
    font-weight: bold;
    color: red;
}
.e-selected span.e-icons.highlight-day:before { /* csslint allow: adjoining-
classes*/
    color: #fff;
}
@font-face {
    font-family: 'e-icons';
    src: url(data:application/x-font-ttf;charset=utf-
8;base64,AAEAAAAKAIAAAwAgTlMvMjciQ6oAAAEoAAAAVmNtYXBH1Ec8AAABsAAAAHJnbHlmKcX
fOQAAAKAAAAg4aGVhZBLt+DYAADQAAAAANmhoZWEHogNsAAAArAAAACRobXR4LvgAAAAAYAAAAA
wbG9jYQYkCgIAAAIkAAAAGmlheHABGQEOAAABCAAAACBuYW1lR4040wAACngAAAJtcG9zdEFgIbw
AAAZoAAAArAABAAADUv9qAFoEAAAA//UD8wABAAAAAAAAAAAAAAAAADAABAAAAAQAAIbm7l8
PPPUACwPoAAAAANfuWa8AAAAA1+5ZrwAAAAAD8wPzAAAACAACAAAAAAAAAAAAEAAAAQAIAAwAAAAA
AAgAAAAoACgAAAP8AAAAAAAAAAQpQAZAABQAAANoCvAAAAIwCegK8AAAB4AAxAQIAAAIABQMAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAUGZFZABA4QLhkANS/2oAWgPzAJYAAAAABAAAAAAAAABAAAAAPoAAA
D6AAAA+gAAAPoAAAD6AAAA+gAAAPoAAAD6AAAA+gAAAPoAAAD6AAAAAAAgAAAAAMAAAAUAAAAQA
AABQABABeAAAAADgAIAAIABuEC4QnhD+ES4RvhkP//AADhAuEJ4QvhEuEa4ZD//wAAAAAAAAAAAA
AAAAABAA4ADgAOABYAFgAYAAAAAQACAAYABAADAAGABWAKAAKABQALAAAAAAAAAAB4AQABaAQYB5gJ
kAnoCjgKwA8oEHAaaaaIAAAAAA+oDlQAEAAoAAAEFESERCQEVCQE1AgcBZv0mAXQB5P4c/g4Cw/D
+lwFpAcP+s24BTf6qbgAAAAEAAAAAA+oD6gALAAATCQEXCQEHQEnCQF4AYgBiGP+eAGIY/54/nh
jAYj+eAPr/ngBiGP+eP54YwGI/nhjAYgBiAAAAwAAAAAD6gOkAAMABWALAAA3IRUhESEVIREhFSE
VA9b8KgPW/CoDlVwq6I0B64wB640AAAEAAAAAA+oD4QCaaAABMx8aHQEPDjEPah8bIT8bNS8SPxs
CAAb0aGhgMDAsLCwoKCgkJCQgHBWYGBgUEBAMCAgECAwUFBggICQoLCwwMDg0GAgEBAGIDBAMIBiI
dHh0cHBoZFhUSEAcFBgQDAwEB/CoBAQMDBAUGBw8SFRYYGhsbHB0cHwsJBQQEAwIBAQMEDg0NDAs
LCQkJBwYGBAMCAQEBAgIDBAQFBQYGBwgICAKJCgoKCwsLDawMGRoD4gMEBwQFBQYGBwgICAKKCgs
LDawNDQ4ODxAQEBEWFxYWFhYVFRQUEXIRERAOFxMLCggIBgYFBgQMDawNDg4QDxERERIJCQkKCQk
JFRQJCQoJCQgJEhERERAPDw4NDQsMBWgFBgYICQkKDAwODw8RERMTEuUUFhUWFxYWFxEQEBApDg4
NDQwMCwsKCgkICAgHBgYFBQQEBQAAAAAAwAAAAAD8wPzAEEAZQDFAAABMx8FFREzHwYdAg8GIS8
GPQI/BjM1KwEvBT0CPwUzNzMfBR0CDwUrAi8FPQI/BTMnDw8fFz8XLxcPBgI+BQQDAwMCAT8EBAM
DAwIBAQIDAwMEBP7cBAQDAwMCAQECAwMDBAQ/PwQEAWMDAgEBAGMDAwQE0AUEAWMDAgEBAGMDAwQ
FfaUEAWMDAgEBAGMDAwQFvRsbGRcWFRMREA4LCQgFAwEBaWUHCgsOEBETFRYXGRocHR4eHyAgISI
iISAgHx4eHRsbGRcWFRMREA4LCQgFAwEBaWUHCgsOEBETFRYXGRsbHR4eHyAgISIiISAgHx4eAqY
BAGIDBAQE/rMBAQEDAwQEBGgEBAQDAgIBAQEBAgIDBAQEaAQEBAMDAQEB0AECaWMDBAVoBAQDAwM
CAeUBAgIEAwQEaAUEAWMDAgEBAGMDAwQFAwQEAWQCAgElERMVfhcZGhwdHh4fICAhIiIhICAfHh4
dGxsZFxYVExEQDgsJCAUDAQEDBQcKCw4QERMVfhcZGxsDhH4fICAhIiIhICAfHh4dHBoZFxYVExE
QDgsKBwUDAQEDBQcKCw4AAAIAAAAAAAA9MD6QALAE8AAAEAOAQcuAsc+ATceAQEHBgcNjgYPAQYWHwE
GFBCHDgEfAR4BPWEWHwEeATsBMjY/ATY3FxY2PwE2Ji8BNjQnNz4BLwEuAQ8Bji8BLgErASIGaps
BY0tKYwICY0pLY/7WEy4nfAkRBWQEAWdqAwNqBwMEZAURCXwnLhMBDgnICg4BEy4mfQkRBGQFAwh
pAwNpCAMFZAQSCH0mLhMBDgrICQ4B9UpjAgJjSkpjAgJjAZWEFB4yBAYIrggSBLIYMhhSBhIIrgg
FAzIfe4QJDawJhBQeMgQGCK4IEgZSGDIYUgYSCK4IBQMyHxOEQCwMAAEAAAAAAwED6gAFAAAJAic
JAQEAef+FhoBzf4za+v+Ff4VHwHMAc0AAAAAAQAAAAADAQPqAAUAAAEEXCQEAQL1Hf4zAc0a/hY
D6x7+M/40HwHrAAEAAAAAA/MD8wALAAATCQEXCQE3CQEnCQENAY7+cmQBjwGPZP5yAY5k/nH+cQO
P/nH+cWQBjv5yZAGPAY9k/nEBjwAAAwAAAAAD8wPzAEEAgQEBAALDw4rAS8dPQE/DgUVDw4BPw4
7AR8dBRUFHTsBPx09AS8dKwEPHQL1DQ00Dg4PDw8QEBAQERERERUUFbQTExITEREREBAPDw00DAw
LCwkJCACGBgQEAgIBAgIEAwUFBgYHBwkICQoCygECAGQDBQUGBgHCQgJCv3QDQ00Dg4PDw8QEBA
QERERERUUFbQTExITEREREBAPDw00DAwLCwkJCACGBgQEAgL8fgIDBQUHCAkKCwNDg8PERESExQ
UFRYWFhgXGBkZGRoaGRkZGBcYFhYWFRQUEXIREQ8PDg0MCwoJCACFBQMCAGMFBQcICQoLDA00Dw8
RERITFBQVfHYWGBcYGRkZGhoZGRkYFwgWfHYVFBQTEhERDw8ODQwLCgkIBwUFAwLFCgkICQcHBgY
FBQMEAgIBAgIEBAYGBwgJCQsLDawODQ8PEBARERETeHMTFBQUFREREREQEBAQDw8PDg4ODQ31ERE

```

```

RERAQEBAPDw8ODg4NDQIwCgkICQcHBgYFBQMEAgIBAgIEBAYGBwgJCQsLDAwODQ8PEBARERETehM
TFBQUFRoZGRkYFwgWfHvYVFBQTEhERDw8ODQwLCgkIBwUFAwICAwUFBwgJCgsMDQ4PDxEREhMUFBU
WfHvYFwgZGRkaGhkZGRgXGBYWFHvUUFBMSEREPdW4NDAsKCQgHBQUDAgIDBQUHCAkKCwwNDg8PERE
SExQUFRYWFhgXGBkZGQAAQAAAAAD6gPqAEMAABMhHw8RDw8hLw8RPw6aAswNDgwMDAsKCcgIBwU
FAwIBAQIDBQUHCAgKCgsMDAwODf00DQ4MDAwLCgoICAcFBQMCAQECAwUFBwgICgoLDAwMDgPrAQI
DBQUHCAgKCgsLDA0NDv00Dg0NDAsLCgoICAcFBQMCAQECAwUFBwgICgoLCwwNDQ4CzA4NDQwLCwo
KCAgHBQUDAgAAABIA3gABAAAAAAAAAAAAAABAAAAAAAAABAA0AAQABAAAAAAAACAAcADgABAAAAAA
DAA0AFQABAAAAAAAAEAA0AIgABAAAAAAFAAsALwABAAAAAAGAA0AOgABAAAAAAKACwARwABAAA
AAAAABIAcWADAAEECQAAAAIAhQADAAEECQABABOAhwADAAEECQACAA4AoQADAAEECQADABOArwA
DAAEECQAEABOAYQADAAEECQAFABYA4wADAAEECQAGABOA+QADAAEECQAKAFgBEwADAAEECQALACQ
BayB1LW1jb25zLW1ldHJvUmVndWxhcmtUtaWNvbnMtbnV0cm91LW1ldHJvVmVyc2lvbiA
xLjB1LW1jb25zLW1ldHJvRm9udCBnZW51cmF0ZWQgdXNpbmcgU3luY2Z1c2lvbiBNZXRybyBTdHV
kaW93d3cuc3luY2Z1c2lvbi5jb20AIABlAC0AaQBjAG8AbgBzAC0AbQB1AHQAcbBvAFIAZQBnAHU
AbABhAHIAZQAtAGkAYwBvAG4AcwAtAG0AZQB0AHIAbwB1AC0AaQBjAG8AbgBzAC0AbQB1AHQAcbB
vAFYAZQByAHMAaQBvAG4AIAAxAAC4AMABlAC0AaQBjAG8AbgBzAC0AbQB1AHQAcbBvAEYAbwBuAHQ
AIAbnAGUAbgB1AHIAyQB0AGUAZAAGAHUAcwBpAG4AZwAgAFMAeQBvAGMAZgB1AHMAaQBvAG4AIAAB
NAGUAdABYAG8AIABTAHQAAdQBkAGkAbwB3AHcAdwAuAHMAeQBvAGMAZgB1AHMAaQBvAG4ALgBjAG8
AbQAAAAACAAAAAoooooAAAAAAAAAAAAAAAAAAAAAAAAAwBAGEDAQQBBQEGAQcBCAEJAQo
BCwEMAQ0AB2hvbWUtMDELQ2xvc2UtaWNvbnMhbnVudS0wMQRlc2VyB0JUX2luZm8PU2V0dGluZ19
BbmRyb2lkDWNoZXZyb24tcmlnaHQMY2hldnJvb1lsZWZ0CE1UX0NsZWFiYDE1UX0p1bmtdtYWlscwR
zdG9wAAA=) format('trueType');
    font-weight: normal;
    font-style: normal;
}
</style>

```

{% previewsample "page.domainurl/code-snippet/calendar/min-max-cs5" %}

### Highlight Weekends

You can highlight the weekends of every month in a Calendar by using the [renderDayCell](#) event. The following example demonstrates how to highlights the weekends of every month.

#### APP.VUE

```

<template>
  <div id="app">
    <div class='wrap'>
      <ejs-calendar id='calendar' :renderDayCell="highlightWeekend"
    ></ejs-calendar>
    </div>
  </div>
</template>
<script>
import Vue from "vue";
import { CalendarPlugin } from "@syncfusion/ej2-vue-calendars";
Vue.use(CalendarPlugin);
export default {
  methods: {
    highlightWeekend: function(args) {
      if (args.date.getDay() === 0 || args.date.getDay() === 6) {
        // To highlight the week end of every month
        args.element.classList.add('e-highlightweekend');
      }
    }
  }
}

```

```

</script>
<style>
  @import "../node_modules/@syncfusion/ej2-base/styles/material.css";
  @import "../node_modules/@syncfusion/ej2-buttons/styles/material.css";
  @import "../node_modules/@syncfusion/ej2-vue-
  calendars/styles/material.css";
  .wrap {
    margin: 35px auto;
    width: 240px;
  }
  #date_label {
    display: block;
    width: 248px;
    color: rgba(0, 0, 0, 0.58);
    margin-left: 5px;
  }
  body.highcontrast #date_label {
    color: white;
  }
  .e-highlightweekend {
    background-color: #cfe9f3;
  }
</style>

```

{% previewsample "page.domainurl/code-snippet/calendar/highlight-weekend-cs1" %}

See Also

- [Add the external button in the Calendar popup](#)
- [How to skip a month in Calendar](#)
- [How to change the first day of week](#)
- [How to customize the calendar day header](#)

## Multi select in Vue Calendar component

Calendar provides an option to select **single** or **multiple dates** by using `isMultiSelection` and `values` properties. By default, `isMultiSelection` property will be in disabled state.

| API | Type | Description |

|-----|-----|-----|

| `isMultiSelection` | **Boolean** | Enables the multi-selection option in the Calendar control |

| `values` | **Date[]** | Gets or sets the date range values in multi-selection option |

The following example demonstrates the functionality of `isMultiSelection` property and `values` properties in the Calendar control.

### APP.VUE

```

<template>
  <div id="app">
    <div class='wrap'>
      <ejs-calendar id='calendar' ref="CalendarInstance"
      :isMultiSelection="isMultiSelection" :values="values" ></ejs-calendar>

```



```

        </div>
    </div>
</template>
<script>
import Vue from "vue";
import { CalendarPlugin } from "@syncfusion/ej2-vue-calendars";
Vue.use(CalendarPlugin);
export default {
    data () {
        return {
            isMultiSelection: true,
            values: [new Date('1/1/2020'), new Date('1/15/2020'), new
Date('1/3/2020'), new Date('1/25/2020')]
        }
    }
}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-buttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-calendars/styles/material.css";
.wrap {
    margin: 0px auto;
    max-width: 250px;
}
</style>

```

{% previewsample "page.domainurl/code-snippet/calendar/multi-selection-cs2" %}

### Calendar views in Vue Calendar component

The Calendar has the following predefined views that provide a flexible way to navigate back and forth when selecting dates.

#### | View | Description |

| --- | --- |

| month (default) | Displays the days in a month. |

| year | Displays the months in a year. |

| decade | Displays the years in a decade. |

When view is defined to the [start](#) property of the Calendar, it allows you to set the initial view on rendering.

The following example demonstrates how to set the **year** as the start view of the Calendar.

#### APP.VUE

```

<template>
    <div id="app">
        <div class='wrap'>
            <ejs-calendar id='calendar' :value='calVal' start='Year'></ejs-
calendar>
        </div>
    </div>
</template>

```

```

<script>
import Vue from "vue";
import { CalendarPlugin } from "@syncfusion/ej2-vue-calendars";
Vue.use(CalendarPlugin);
export default {
  data () {
    return {
      calVal: new Date()
    }
  }
}
</script>
<style>
  @import "../node_modules/@syncfusion/ej2-base/styles/material.css";
  @import "../node_modules/@syncfusion/ej2-buttons/styles/material.css";
  @import "../node_modules/@syncfusion/ej2-vue-
calendars/styles/material.css";
  .wrap {
    margin: 35px auto;
    width: 240px;
  }
</style>

```

{% previewsample "page.domainurl/code-snippet/calendar/min-max-cs2" %}

#### View restriction

By defining the [start](#) and [depth](#) property with the different view, drill-down and drill-up views navigation can be limited to the user. Calendar views will be drill-down up to the view which is set in **depth** property and drill-up up to the view which is set in **start** property.

The following example displays the Calendar in **decade** view, and allows you to select a date in **month** view.

Depth view should always be smaller than the start view. If the views are the same, then the Calendar view remains unchanged

#### APP.VUE

```

<template>
  <div id="app">
    <div class='wrap'>
      <ejs-calendar id='calendar' start='Decade' depth='Year'></ejs-
calendar>
    </div>
  </div>
</template>
<script>
import Vue from "vue";
import { CalendarPlugin } from "@syncfusion/ej2-vue-calendars";
Vue.use(CalendarPlugin);
export default {
  data () {
    return {
    }
  }
}

```

```

</script>
<style>
  @import "../node_modules/@syncfusion/ej2-base/styles/material.css";
  @import "../node_modules/@syncfusion/ej2-buttons/styles/material.css";
  @import "../node_modules/@syncfusion/ej2-vue-
  calendars/styles/material.css";
  .wrap {
    margin: 35px auto;
    width: 240px;
  }
</style>

```

{% previewsample "page.domainurl/code-snippet/calendar/min-max-cs3" %}

### Accessibility in Vue Calendar component

The Calendar component followed the accessibility guidelines and standards, including [ADA, Section 508](#), [WCAG 2.2](#) standards, and [WCAG roles](#) that are commonly used to evaluate accessibility.

The accessibility compliance for the Calendar component is outlined below.

| Accessibility Criteria | Compatibility |

| -- | -- |

| [WCAG 2.2 Support](#) |  |

| [Section 508 Support](#) |  |

| [Screen Reader Support](#) |  |

| [Right-To-Left Support](#) |  |

| [Color Contrast](#) |  |

| [Mobile Device Support](#) |  |

| [Keyboard Navigation Support](#) |  |

| [Accessibility Checker Validation](#) |  |

| [Axe-core Accessibility Validation](#) |  |

<style>

.post .post-content img {

display: inline-block;

margin: 0.5em 0;

```

}
</style>

<div> - All
features of the component meet the requirement.</div>

<div> - Some features of the component do not meet the requirement.</div>

<div> - The
component does not meet the requirement.</div>

```

### WAI-ARIA attributes

The web accessibility makes web content and web applications more accessible for disabled people. It especially helps in dynamic content change and development of advanced user interface controls with AJAX, HTML, JavaScript, and related technologies.

Calendar provides built-in compliance with [WAI-ARIA](#) specifications. WAI-ARIA support is achieved through attributes like `aria-label`, `aria-selected`, `aria-disabled`, and `aria-activedescendant` applied for navigation buttons, and disable and active day cells.

It helps disabled persons by providing information about the widget for assistive technology in the screen readers. Calendar component contains grid role and grid cell for each day cell.

- **Aria-label:** This attribute provides text labels for an object for the previous and next month's elements. It helps the screen reader object to read.
- **Aria-selected:** Indicates the currently selected date of the Calendar component.
- **Aria-disabled:** Indicates the disabled state of the Calendar component.
- **Aria-activedescendant:** Helps in managing the current active child of the Calendar component.
- **Role:** Gives information to assistive technologies about how to handle each element in a widget.
- **Grid-cell:** Defines the individual cell that can be focused and selected.

### Keyboard interaction

You can use the following keys to interact with the Calendar. This component implements keyboard navigation support by following the [WAI-ARIA practices](#).

It supports the following list of shortcut keys:

| Press | To do this |

| --- | --- |

| Upper Arrow | Focuses the same day of the previous week. |

| Down Arrow | Focuses the same day of the next week. |

| Left Arrow | Focuses the day before. |

| Right Arrow | Focuses the next day. |

| Home | Focuses the first day of the month. |

| End | Focuses the last day of the month. |

| Page Up | Focuses the same date of the previous month. |

| Page Down | Focuses the same date of the next month. |

| Enter | Selects the currently focused date. |

| Shift + Page Up | Focuses the same date for the previous year. |

| Shift + Page Down | Focuses the same date for the next year. |

| Control + Upper Arrow | Moves to the inner level of view like month to year and year to decade. |

| Control + Down Arrow | Moves out from the depth level view like decade to year and year to month. |

| Control + Home | Focuses the first date of the current year. |

| Control + End | Focuses the last date of the current year. |

To focus the Calendar component, use **alt+t** keys.

#### APP.VUE

```
<template>
  <div id="app">
    <div class='wrap'>
      <ejs-calendar id='calendar' ref="CalendarInstance"></ejs-
calendar>
    </div>
  </div>
</template>
<script>
import Vue from "vue";
import { CalendarPlugin } from "@syncfusion/ej2-vue-calendars";
Vue.use(CalendarPlugin);
export default {
  mounted () {
    let calendarObj = this.$refs.CalendarInstance;
    document.onkeyup = function (e) {
      if (e.altKey && e.keyCode === 84) {
        // press alt+t to focus the control.
        calendarObj.$el.querySelectorAll('.e-content table')[0].focus();
      }
    }
  }
}
</script>
<style>
  @import "../node_modules/@syncfusion/ej2-base/styles/material.css";
  @import "../node_modules/@syncfusion/ej2-buttons/styles/material.css";
  @import "../node_modules/@syncfusion/ej2-vue-
calendars/styles/material.css";
  .wrap {
    margin: 35px auto;
    width: 240px;
  }
</style>
```

{% previewsample "page.domainurl/code-snippet/calendar/min-max-cs1" %}

### Ensuring accessibility

The Calendar component's accessibility levels are ensured through an [accessibility-checker](#) and [axe-core](#) software tools during automated testing.

The accessibility compliance of the Calendar component is shown in the following sample. Open the [sample](#) in a new window to evaluate the accessibility of the Calendar component with accessibility tools.

```
{% previewsample "https://ej2.syncfusion.com/accessibility/calendar.html" %}
```

See also

- [Accessibility in Syncfusion Vue components](#)

### Islamic calendar in Vue Calendar component

In addition to the Gregorian calendar, the Calendar control supports displaying the Islamic calendar (Hijri calendar). **Islamic calendar** or **Hijri calendar** is a **lunar calendar** consisting of 12 months in a year of 354 or 355 days. To know more about Islamic calendar, please refer this [wikipedia](#).

Also, it consists of all Gregorian calendar functionalities as like min and max date, week number, start day of the week, multi selection enable RTL, start and depth view, localization, highlight and customize the specific dates.

By default, calendar mode is in **Gregorian**. You can enable the Islamic mode by setting the **calendarMode** as **Islamic**. Also, need to import and injecting the **Islamic** module into the **provide** section from **ej2-vue-calendars** as shown below.

```
import { Islamic, CalendarPlugin } from '@syncfusion/ej2-vue-calendars';\
```

```
Vue.use(CalendarPlugin);
```

The following example demonstrates how to display the Islamic Calendar (Hijri Calendar).

#### APP.VUE

```
<template>
  <div id="app">
    <div class='wrap'>
      <ejs-calendar :renderDayCell="load" :calendarMode="type"
:cssClass="cssClass"></ejs-calendar>
    </div>
  </div>
</template>
<script>
import Vue from "vue";
import { CalendarPlugin, Islamic } from "@syncfusion/ej2-vue-calendars";
import { addClass } from "@syncfusion/ej2-base";
Vue.use(CalendarPlugin);
export default {
  data () {
    return {
      cssClass: "e-customStyle",
      type : "Islamic",
    }
  },
  methods: {
    load: function(args) {
```

```

    /*Date need to be disabled*/
    if (args.date.getDate() === 12 || args.date.getDate() === 17 ||
args.date.getDate() === 22) {
        args.isDisabled = true;
    }
    /*Date need to be customized*/
    if (args.date.getDate() === 13) {
        let span = document.createElement('span');
        args.element.children[0].className += 'e-day sf-icon-cup highlight';
        addClass([args.element], ['special', 'e-day', 'dinner']);
        args.element.setAttribute('data-val', ' Dinner !');
        args.element.appendChild(span);
    }
    if (args.date.getDate() === 23) {
        let span = document.createElement('span');
        args.element.children[0].className += 'e-day sf-icon-start highlight';
        span.setAttribute('class', 'sx !');
        //use the imported method to add the multiple classes to the given
element
        addClass([args.element], ['special', 'e-day', 'holiday']);
        args.element.setAttribute('data-val', ' Holiday !');
        args.element.appendChild(span);
    }
}
},
provide: {
    calendar: [Islamic]
}
});
</script>
<style>
    @import "../node_modules/@syncfusion/ej2-base/styles/material.css";
    @import "../node_modules/@syncfusion/ej2-buttons/styles/material.css";
    @import "../node_modules/@syncfusion/ej2-vue-
calendars/styles/material.css";
    .wrap {
        margin: 0 auto;
        width: 250px;
    }
    .e-calendar .e-content td.e-selected span.e-day.sf-icon-
cup.highlight:before,
    .e-calendar .e-content td.e-selected span.e-day.sf-icon-
start.highlight:before {
        color: #ffff !important;
    }
    .sf-icon-cup:before {
        content: '\e724';
        color: #0b0bd9a1;
        position: relative;
        top: 1px;
        left: -2px;
        font-size: 11px;
        font-family: 'e-sb-icons' !important;
    }
    .sf-icon-start:before {
        content: '\e708';
        color: #0b0bd9a1;

```

```

    position: relative;
    top: 1px;
    left: -1px;
    font-size: 12px;
    font-family: 'e-sb-icons' !important;
  }
  .e-customStyle span.e-icons.highlight {
    margin-top: -13px;
    display: block;
    margin-left: 4px;
  }
  .e-customStyle .e-other-month span.e-icons.highlight:before {
    content: "";
  }
  .e-customStyle .e-selected span.e-icons.highlight:before {
    color: #fff;
  }
  .e-customStyle span.e-icons.highlight,
  .e-customStyle .e-calendar .e-content span.special,
  .e-customStyle .e-calendar .e-content span.daycell {
    font-weight: bold;
    font-size: 14px;
  }
  .e-customStyle .e-calendar .e-content span.special {
    border-radius: 50%;
  }
  /* custom generated icons styles */
  @font-face {
    font-family: 'e-sb-icons';
    src: url(data:application/x-font-ttf;charset=utf-
8;base64,AAEAAAIAIAAwAgTlMvMj0gSUMAAAEoAAAVmNtYXCfQJ/XAAACSAAAAK5nbHlMkTE
7TgAAA2AAACmAAgVhZBMHxX8AAADQAAAAANmhoZWEHlQOTAAAArAAAAACRobXR4w2z/4QAAAYAAAAD
IbG9jYdGqyAAAAAL4AAAAZmlheHABSGEWAABCAAAACBuYW1lTGtTDAALoAAAAAJCg9zdGE/72E
AAC8sAAACyGABAAADUv9qAFoEAP/0//YD8gABAAAAAAAAAAAAAAAAAMgABAAAAAQAA3FAQWF8
PPPUACwPoAAAAANGBRFGAAAAA2AFWEP/0//8D8gPrAAAAACAACAAAAAAAAAAAAEAAAAyAQoADgAAAAA
AAgAAAAoACgAAAP8AAAAAAAAAAQPoAZAABQAAANoCvAAAAIwCegK8AAAB4AAxAQIAAAIABQMAAAA
AAAAAAAAAAAAAAAAAAAAAAAAUGZFZABA5wDnMwNS/2oAWgPrAJYAAAABAAAAAAAAABAAAAAPo//Q
D6AAAA+gAAAPoAAAD6AAAA+gAAAPoAAAD6AAAA+gAAAPoAAAD6AAAA+gAAAPoAAAD6AAAA+j//gP
oAAAD6AAAA+gAAAPoAAAD6AAAA+gAAAPoAAAD6AAAA+gAAAPoAAAD6P/8A+gAAAPoAAAD6AAAA+g
AAAPoAAAD6AAAA+gAAAPoAAAD6AAAA+j/9APoAAAD6AAAA+gAAAPo//8D6AAAA+gAAAPoAAAD6AA
AA+gAAAPoAAAD6AAAA+gAAAPoAAAAAAAACAAAAAwAAABQAAwABAAAAFAAAEAJoAAAAKAAGAAgAC5xv
nHuc15zP//wAA5wDnHecg5yf//wAAAAAAAAAAAAAEACgBAAEIATAAAAAEAAgADAAQABQAGAAACAA
JAAoACwAMAA0ADgAPABAAEQASABMAFAAVABYAFwAZABgAGgAbABwAHQAeAB8AIAAhACIAIwAkACU
AJgAnACgAKQAqACsALAAAtAC4ALwAwADEAAAAAAAAAAVgCIANABBAE+AUwBYgGMAagBtgHOAmACsAL
UA24D2gRWBjAE0gVCBZgFuAXsBkwGegc0B5wIEgh2CTQJrgqcC1AL6gw2DJgODg6cDu4PUg/AEGQ
REBF2EfITNBnmFG4UwAAAAAP/9AAAA+oDeQASACQALwAAEwYSFxyY2NxcTBRCOAScmAajcOATcRHgE
XISc3NTYnAQYiJwEOATcTFjI3ATc2NSEWBhPrxoDeTV0+/mxgQKlhorsQBw1lDB4Rabdq+QEN/ts
HFAf+2wUEMv8HEQcBAAQC/dUBAonK/tYdD15cRwGwokpMSwsXAQi3GDGR/rgSJRBRZNoNCP7bBgY
BJQIMMF8ABwcBAACBAoAAAAADAAAAAPqA20ABAAJABKAAAKBESERJRUJATUjER4BMjEYnJURNcy
nIQ4BAfyYBtvyVA2v+Sv5LPwEjGwNrGyMjG/yVGyMBdwFF/gICAHE1/rsBRyP9jxsjIxsCcRsjaQE
jAAAAAwAAAAAD6gNtABQAGQApAAABJyYOARYfARUHNtc+AS4BDwERCQE1FQkBNSMRHgEzITI2NRE
0JichDgEDrNILGBAEC/b8lekLBBAYC8UBRQG+/kL+Uz8BIxsDaxsjI xv8lRsjaTaYCAQWQGQiyKzW
oCBkWBAiOAXf+wAFLbR/+tQFBKf2PGyMjGwJxGyMBASMBQAAAAAD7QNLAAQABwAKAA0AFgAANYE
lByc3BRMBJQkCISUzNzYmByIGJwYDsP5tRUWZAYED/FABff5/AdwB2PxQAv64LAYRCwcIAaLaMjI
E0gHu/hLSARz+uwFbGVgZEAECgAAAAFAAAAAAPqA64ABQAIaAsAEQAZAAALISUXPwEFLQENARE
lASCHASUFESERNyc1JQN6/QUBJAEFXQGL/u0BE/3I/s4DX/7EXl7+ugGg/gAD5wIC/hp+XgEFQOH
F/vrRAeku/ttDQAEqrI/9ZAjLAQE01AAAAAEAAAAAA+oDKwACAAA3IQECA+j+DMICaQAAAAACAAA

```



Copyright © 2001 -2024 Syncfusion Inc.

BIiYvAQEeAQcGFh8BHgEHISY2PwE+AScmNjc+ATceASUOARcWBicHDgEfASEXHgEyNj8BITc2Ji8  
BBiY3NiYnLgEnDgECVQEPMD0wDwEBKioOAhk/EwM8Ew79Aw4SPQUSPhkCDiogZ0REaP5bORADFSg  
HCVYSFg4BBAETVnFWEWEBBA4WE1UHBiWVAXA5KX9SUn6AAxshIRSDAuQ0WwTDkgQCnmUdHGY2AwO  
SwQRdNCI1AQE1CEV7BqN4AQZMLsgfAjderDcCHyiVTAQDdqsFekUsLwEBLwAEAAAAA0cA+oAdwA  
ZAB4APgAAJTIZNx4BFw4BBY4BJz4BNwEHDgEHY4BLwE1HQEHNScVHgEXMRUjDgEHHgEyNjcuASc  
jNTc+ATc1NCYnIQ4BAfMCAQ1jdAEBe2loewICc2QBbgIbv4QBhb8cAQLI/TI/BNqoCWaSAwSu4q4  
FBjJlCQKn2QQSDfzZDRLXAQqWGBgxAgIxGBgwBAI4B32fAgOffQacWwJdH3mq6hCZBkU+REVFRD5  
FBpgBEomqeg4RAQERAAAAAIAAAAAA8YD6gBYAH8AAAEeARUOAQ8BMxYXNzY3PgEeAgYHDgEPARU  
WBzEXFhceARQOAicuAS8BBgcXFhcWDgEiLgE3PgE3NTEmJzQ3NScmJy4CPgEyFx4BHwE2NzEnJic  
mPgEyJw4BBxUhFR4BFy4BJzUjAREjDgEHIS4BJyMRExceATc+AScuASciAzoHBwcSBgEBBAIKHRS  
ECw0IAwQGFy4XAQECCh8iBgkICw0EDxoPAQIEAQMHAgILDw4IAQcRBwcCAQUiJQYKAQGLDAQQHBA  
BAwQBawcAQwPIztJav56EUYEKFYUoAFUAULcAQGgAVxJAFEDGDsgQUshCLQ6DAPHAgOGfiwWAgE  
DBxQBZQCFDQ4LAWUIBQECBAMGEg4CCA4NCAEEIEicSAQIBCYQkBBQwJBAoGFisXAgIIAwMBAhINAgc  
PDQoEESUSAQMBBScmBgwJIA1UOGMBQqCHM5cmAf5k/oQEHBQUHAQBfAEjAhMQBQpjQjtJAgAABQA  
AAAAD6gPnAAcAdGAmACwASwAAJSE3FxYyPwETDgEHLgEvAR4BHwEHBhYzITI2LwE3PgE3NCYjIQ4  
BJQ4BIiYnJQ4BDwEjISIGFR4BFz4BPwI+ATc+ATcyNjQmIw4BAnT+iCoFQ5NCB+MQZ5aXzxBAAnx  
nBUIJEhIB5xIRCUIEZ3wBEg381Q4RADwQM0E0EAGkBGm3BAX+zg0SAm1NPl8TAgi9agYDhWcoERE  
OdJVERQIWFgIBeJK9BAS9kh99xjkCbRafHxBtAjnHfA0SARHfERSbEZIDOhYBEg0sWgQCQSYFARc  
+BAU3BBIAegQ+AAAAAIAAAAAA7ED6gAPAKgAACUWNj8BBw4BBY4BJzU3BhYDFBYzIxUjFR4BFxY  
2PwEVDgEHLgEnFSM1IxY2Ny4BJw4BBx4BNyMVIxUeARcWNj8BBw4BBY4BJxUhNQ4BBY4BJzU3BhY  
XFjY3NSM1Iz4BNS4BJw4BBxQWFyMVIzUOAQcuASc1NwYWFxY2NzUjNSMyNjcuAScOAQceATMjFSM  
1IzI2NS4BJw4BBxQWMyMVIzUjMjY1LgEnDgEB7wZtKAIBA09GSU0DAgFFXhcRD4UBR0IFZSYEA0p  
DP04BQhUSFgEDJQIDJQMBfXEuLRA/NgZsJwEBak1GNUMRA3YRRjZITgMBAUVQBWYmLBorFwI1AwI  
lAxcRDkMQRDTfSwMBAUNMBWgmhRURFwEDJQMCJQMBfHETWRcRfWIlAwIlAxcREVgaERcCJQMDJcg  
DGkMEB0FpAwNpQQUDBlMCrimJm1ESQAMDG0IBz5kAwJSOJ6yARwiJkACAUAmIh0BsoAXMAIDGkM  
CBUFpAwE+LtzIMEECA2lBBwEGUwYDFz18mwEkIyZfAgNFJyIjAZueN1MCA2Q+BwEGTwYDGUBQmyU  
jJkQDA0UnIyObmyUjJkQDA0UnIyObmyUjJkQDA0UAAAAACQAAAAAD6gPhAAsAFwAjAC8APgBVAfK  
AcQB7AAAlFRQWMjY9ATQmIgYHFRQWMjY9ATQmIgYlFRQWMjY9ATQmIgYFFRQWMjY9ATQmIgYnHgE  
yNj8BFQ4BBY4BJzUlHgEXDgEHLgEnPgE3MxUUFjSBmjY9AScRIxEnFSMOAQcDFgQgJdCrJiQnIzU  
0JisBIgY3HgEyNjcuATUGAlAJDQkJDQnxCQ0JCQ0JAdAJDQkJDQn9cAkNCQkNCVlI6f/pSAUC5s7  
N5gIB96jJAwLmzs3mAgPHpwYJB10HCR8/Hwii/wwBCQeATAX0BLQgM/v6jBAkHXQcJFAEYJBcBBSg  
p5V4GCQkGXgcJCQdeBgkJB14HCQkYXQcJCQddBwkJB10HCQkHXQcJCWovMjIvBKMvWMDWi+j8At  
SLs9aAwNaLy1SC6EGCQkGoYj+5wEZEfKJYVn+8WNNz2MBDlpiCFkGCQhwEhgyEiEjJTQAAwAAAAA  
DrAPqAAGAKgBgAAABHgEUBgcjNjclFgYHBjY3PgE3HgEXDgEHLgEnPgE3FAYHBjY3PgE3NjcwAw4  
BFx4BFw4BBxYSFz4BNzM+ATcuAScNjcuAScuATc2JiIGBw4BFx4BFyYjIgcUATc2Jg4BAy0fKio  
fQx4T/v4BFRkNkgwZGgJQWakMoZWVogsJa2AUGg0rCwxZAhwgNysyJg4IFwlxjQIazZFTjzVdNkg  
BAUg2BwcEan9oFydrCAcTEwQzJg8HFgoqLCEgGCNPCAcsEwGkASTBKwFGU8gaLxUNAgcVLhk0Jw0  
VNgMDNhUPKwzLhUNAgcXMxsBAQEBCyZfQB09Hw5DLO/+zwUBb2ECSjc4SgEmKCpBD1KlRgcFBQM  
mXz8dPB4EA1ChRacFAQQAAAAADAAAAAPqA9UACQAVAcSAAEzHgEUBgcjNjcnDgEHLgEnPgE3HgE  
FGgEXPgE3Mz4BNy4BJyM2Ny4BJw4BA0UUJDAwJE0jFiUNuaqquA4OuKqqufzvHuulX6M8az5SAQF  
SPggIBQXyt7fyAmIBPVs8AWJ0tB1MBARMHRxMBARMfv6w/1MHAZyJAmhPTmgCNThPaQICaQAAAAA  
/9AAAA+oD1gAcAdoAAAEGBw4BBw4BBwYHDgEHBhYXFjI3PgE3PgEnLgEjAQYCFx4BNzY3Njc+ATc  
+ATc2NxcWJy4BJy4BJyIGA3lJR0B5NC1zRiIkDysMDjUTZPl/h6gVBgEVDykU/cKpsBMGJS4oKR0  
boF8l04xQWWImEQQDFwk6oF0/gwKMAyMhcs+aCQRDAQECxYgCTk+ReKBHkYWDAGBC1j+0aIxNAI  
BDQoNHVU1U4MpLgMDARoOHQo/QwIeAAIAAAAAALxA+oAIQBBAFAAYwCEAKIAXAEJAAALFxQOAgc  
jLgM3NTY7ARYUHgIXMzI+AjUxJjsBmiUWBw4BBw4BFYXHG7ATIUBymIjicuATQ2Nz4BNzMyNxc  
WFzMWBiMnIiY1MTQ2NxYHBisBjicxJjYeATsBNjc2MgM2FhUHBgcVFBYXFCInLgE3NTQ/ATQjJwY  
PAQYmPwE2PwEWBw4BBw4BBwYjIi8BIjYzFxyZmjc+ATc+ATc2MgEhMgYWFx4BHwEeAQcGBxYGBY4  
BNyYmJjY/ATY3Nj8BNDMHFA4BDwEOARceBxcUAGcOAwcARceAjYiFj4BNzYmJjY4DJj4BNzY+BJc  
2Ji8BLgI0IyEiAsABxNVwlcUBcIAQECAQMEFV9bCVZMFQQBawIC/vYBAW0YfAgJERgoOhgPawM  
PGDsoGBYNcBUyCwEDFwIDECQDAQMkEAsHUQICCh0EGgQCBAQMDAQdBQECFQIIBwUBAwYGAQYEAQY  
GASACAgEDBQIDAgSXAqMJKgsNFQ8KDBEODwMBaw8NEQwIEBUNCSOIAQP+zwFZBgEGAwMSCQCdHQQ  
LIgJSdHNOAiULAwWHGwDAwICBQoGBw4VBw0DBihZMBYQCAUBCQoKI0woBgCDBglBYSACIWBBCQC  
DCAUoTCMKCGQCBAGLfjBZKAYDDQcVDggFCv6nCiwBAwgJDAEBDAkIAwIBAQMFCawBDQgFAQMYAgE  
EBgQBBQMIawYCBQEDBQIDAYCAwYECQICAQEFACAgEHAQMHAQYCAwMDAQQBAWwBBgdZRB0QDjk  
yAwIzOQ4QGkVYBgEBAQECawICAgFIAQEHFgQFBgMBAQIFAQEBawQFBBUGAQHYDUQWED0bFBNFGTQ  
kAygDayQCJjgXRBVOHhAWIioFBQdFKik+FEYYIEI1Fg8YUEIkKGkTDw8JCAMFDAUHEAUBAQUQBwU

MBQMICQ4QE2ooJENPFw4XNUQeGEUVPikqRQ8AAAAAAwAAAAAD6gPBABIAJABcAAA1HgEXIqYjDgE  
HBg8BNS4BJx4BBRUuAS8BJicyNjM+ATc2Nw4BExUOAQceAQcGBw4DFBYXFhceAR8BHgEyNjc+ATc  
2Nz4DNCYnJi8BNCMmNjcuASclLgEiBgMsFSCMAwcDGk0vFRcxAQgHOGX+izpmKCgUDAMHAXpNLz1  
HCQoZaYICBgpmIBsMExMMDAkLyFXNJYPLjcuEEJ2MEIrdBMTDAwJHC8EA1QLCAKHbgEvSS79BQk  
FAwGPBQUBBAYWKBEDDTgHawwHCGQFAwcQBQUFEYgClg01pE1Fe0MJDQULEhkgGw0cDQwQBgoYHR4  
aAw0JDxQEDBIYIrKJGw0DAUV9R0urIwohLCwAAAADAAAAAAPqA+oAFAAGACwAAAErFBYXFjM2PwE  
2LgEGDwERLgEiBgEOAQcuASc+ATceAQUWABc2ADcmACcGAAHXDAoFBBANgCgDFRkIZgERGxEB1AX  
3urn3BQX3ubr3/FsGARrU1QEaBQX+5tXU/uYDZP6TCxADAgEM0wsYEAMLiAEQDRIS/oa69wUF97q  
59wUF97nV/uYFBQEaldQBGGYG/uYAAAX//wAAA+4DjQAFABEAIQAvADsAAAEVMzUjNQUOAQcuASc  
+ATceAScGBx4BFzY3NiYnLgEjIgYFDgEXFjM+ATcnJgciBhMeARc+ATcuAScOAQG52Z0BSQS4iom  
5Aw05iYm5KwUFQFoVDwohCCgdRB8MFv0TKAghCg8VWkAKExgiRRUF5a2t5gQE5q2t5QKs/jzCtYq  
4BAS4iom4BAS49wQFL4NMAQkaaTYmKwZLNmkaCkyDLwoNASz+nq3lBATlra7lBATlAAAAAGAAAA  
D6gOzAdCAQwAAQ8BBhQeAT8BFwEOARY2NwEXIw4BFBYXITY3NiYvATcfARYfARYyPwE2NCcuAgY  
PAQMjJyYjJhceARc+ATcuAScOAQGLA8IQHysRmUv+BxMBIi4SAVp9pRggIBgBLhgOEgMS1UtbBgU  
EBxEoDsUQEAcVExUHNp0DAXIWFLYBPi4uPgEBPi4wPgNYA7sPKyEBDPNL/gYRLyECEQFegAEgMB8  
BAQ8RMhTYS14GBAMDDg6/DyoSCQkDCAeWAQADEwEgLj4CAj4uLT8BAT8AAAGAAAAAA8gD6wAGAA0  
AFwAbACUALgBLAGYAACUhPgE1ESEBFBYXIREhJSEyNj0BNCYjIQMzESMFFRQWFyE1ISIGJR4BMjY  
0JiIGJQ4BBw4BBx4BFACWNjc2NzYmJz4BNzYmJyYHDgEFHqEXFAYXFh8BFjYXFjY3JjY3NiYnLgE  
nJgYCMgFIDhT+lv4fFA4BSP6WAeEBdA4UFA7+jGdUVP5aFA4BdP6MDhQBdAE0UTQ0UTQBLsNEIhg  
mBwwQCRavGGziJyQGAQ4EAgcOCQ0TKf0DBg8BCw4LEyAhQyMKUAEMBhUDPgoPjTsaGQIBew8B9/4  
JDxMBAhopFA5hDhT9GALoImEOFQKoFGwfKCg+KCh0ChwMBxYsSchocDgsCBRMKCTsTDhkOCA8FBAE  
CDBQMGGwWLREJBAYGAQEBAKykRAHJAQIKwIBEAJAaaaaaOsA+sABQAJAA8AEwAYABwALAA8AHI  
AAAEExMxEjESMRiXejOwERiXelFSM1ITMVITUjFSM1JR4BDgEPASM3PgE3NjczFiUWFx4BHWerAS4  
CNjc2MyMOARcWHwEjIgYHFR4BFxEUFhchPgE3ET4BNzUuASsBNTY3NiYnJgciIwYHBg8BJyYnJi8  
BJgKyfbs/fX4BP7wCsJ3+88/+xz6cAkIDCGZBWggJAQcWCxoZBRz+2RkaCxYHAQYCXkUGCQIPHHM  
PDgg0OwGwGyMBASMBIxscCsrsjARojAQEjGrA7DgoXCj03BgU5KQwKAwQKCyk5CzcB9v5LAbX+SwG  
1/ksBtbx9fX19fX3SBBcgNx8DBiNKfi8EAQEELxZJIwYgOCIVAiorNB8wJwEjG30bIwH+SxsjAQE  
jGwG1ASMbfRsJAScvJzYJPgEISxchCQwfFksIAQEAAAGAAAAAA+oD6gAGAA4AFgAdACQAKgAyADO  
AACU+ATcjHgEFPgE3LgEnIwEeARcRIw4BJz4BNyMeAQmZLgEnDgEFMyYnDgEHMz4BNy4BJwUeARc  
zEQ4BA4YrNAXnBkv+wlygP0JUBp/+f0CkXqWfUHMxRgbeBTI33ghFMCoyAvznCloyS/OfCFJCPaJ  
c/oI+UAWsXqPHOYpNaI/eBkk7KbBy/rg9SgYB1W+tDxuMZkuHARFmjBs7h0uadhmpAHKwKtTJB00  
srW8B1QZKAAQAP//A8YD6gADAC4ANwBOAAABMzUjJQYWHwEVAwYWFzMWnJcTFxYXMjY3Ex4BNzM  
+AScDNTMyNjQmJyEnJiMiBjceATI2LgEiBiUUFzYWFxYGDwEXPgE3NiYvASYjJw4BAw64uP0hCwg  
TanoGDxQDEyAIYQMKDwcNBWEHIRMDExAGdP0UICAU/nNqCg8MFXgBMEgxAi9IMAFwCQdqEwcPbA8  
fCYMHBBy6BgMEBgoMAokZRREjCki//qITIQCHEBMBGQMFAQIE/ucTEAcHIRMBsMwJhsBSAkMLiQ  
vL0gwMI8LBQGHkAfXgBAiAriaCel7BwIBAw0ADgAAAAAD6wPqAAIABgAWABkALgBkAGgAbAB/AJs  
AqQCvALQAzwAAJTC1Nwc3BicOAQczPgE/AQc2Jy4BLwE/ATUnBg8BMx4BFxU3Nj8BNDUnLgEnJic  
lBg8BFQYVfHIXMyC+AT8BIYjYjDgEHDgEjBgQvARQ/ATY3PgE3PgE3Nh8BNz4BNzUnJiQHDgEnBhU  
3MT8BNSUOAQcVHgEdAR4BHwE3Nic0LwEBDgEHFT4BNzYEHwE1Nic1LgEnLgE1LgEnJiMnARc+Aj8  
BJy4BLwEeASUGDwI3BSYvASMFBzc2FhcyFhcWHwEWHwE1LgEvASYzIy4BIwYdfQMBaQYDTiarmSt  
vv0oDEwQBARYOBKYDgWUOAwwMWJwIGMBgDEwkbDiEX/QImFxoGBPzFAz+cuyoDaxwmM2AZBwYCPf7  
6CBkmLhshQptMA3lPKSUGAwgMAgZx/vN8QGx+AgMBAgOzHD4NAGQEVSdAwQBIGp9WmSMHQebgXo  
BDnUGAQQcNRYcIkfBPQkGFwIpAwsoHwkDAyh+TgYmb/4RBQYLAyMBZw4QHQT+5AYfCM1qAgcGCAo  
XISEDJHgdBAECBho6HVO+AwECAGgDhFKkUwFXTgMTIRgxPBADMgwK0yoqBhNRRQMJRlMKAgESchI  
KFw53BQMEAYYmx/7wEwlvrlUGCGEXCwIETysDJgEEBQQHcy4nAYYDAQ0EBxUnFQYDPi4BAQCEBQc  
QAgcGQiMhBAMOHw8iAjUgAwwdIWRVBgEDL6pvAwQRAQIqPAMNIBSGCCMUGCUBUYWEQAgH+3QkJEX4  
RAwZLciEDBpukAgMEAw8DBQQKQKBFGLCACkChUYFAMDD6UaCAQHBgEAAAABAAAAAAPPa+oAGgA  
AASsBBg8BJQcFBycHHwE3JzcTNwM3Njc1Nic3A3wKAw4O4f3/VAGsrIZFlk5CEKzbV4PkDgUEARM  
D2AUO8ZBa7rgSSFSiSiY4/j5eAhr0DgoECARUAAABAAAAAD6gPJABsANgBMAKQArQAAJQ4BBY4  
BJzY/ARceAT4BLwEmLwE3NjMeASUXIw4BIw4BHqEXNXy2NxcOAQcuASc+ATcyFhMeARcWHwE  
HDgEHFScuAicmNjcwFycOAQcOARceAR8BIwYHIYcuASMOAQceARc+ATc0LwE3Nj8BFQ4BBwYWFzM  
yNjc+AS8BNTY3Fh8BBw4BBx4BFz4BNy4BJyIHIYcuATc0JicjJyYvAS4BIwY3HgEyNjQmIgYDrAJ  
HNTVHAgERAwQbJAMKGRMBRCRQSGwQEGb41R/2BAQExQgUNEAMRDAMDSdGBakc1NUcCAkc1IzhyGB0  
QCyRIAgwwIAEXRUkMDCIUcGwqCRcQFB4TE1cvHQELJwECGvc2UGoDA2pQUGsCAQEHKSgCAQYFARA  
NBawRagYHBAE0IxYfBgUUFgECa1BQawICa1AxKQIDHSgCDQsCkwYJERwvFwgGASM2IyM2I+e2RwE  
BRzYkHQGGJCsDCQESGQoXFiYEAgwBRwWDEAwCEXoPAQEBCDRMDNkcBAUc2NUcBIQFVIYABAQECDDd  
VIAEBCxskFSWMKQwQYwMdJi+SMYyWEgwbEgMtNgJqUfBrAgJrUAKKAQSHAEFI1wuDRMCDw0+fSg

```

FATBIPDQKBRK9tILBrAgJrUFbQkAhcGMXE5CxIDbQcMFyo1AXYbIyM2IyMAAAAAABwAAAAAD2wPqAAM
ABgAKAAwAEAAUAC4AAAERIRElBzUnFSElISMnISc3ASMBNycBBhQfASmiBhURFBYzITI2NzU3NjQ
nASYiAxz9NANNQj/9NALrH+7+hES/Aeln/rwzLP7MEhIYChokJBoCzBsJAW4Sev4GEzMBXP7lARt
WQoQCPDw+RL/+ /QFFNCz+yxMyExgkGv5LGiQkGtZvEzITAFsSAAAAABIA3gABAAAAAAAAAAAAEAAA
AAAAAAAAABAAoAAQABAAAAAAAAACAaCwABAAAAAAAAADAAoAEgABAAAAAAAAEAAoAHAABAAAAAAAAFAAs
AJgABAAAAAAAAAGAAoAMQABAAAAAAAAAKACwAOwABAAAAAAAAALABIAZwADAAEECQAAAAIAeQADAAEECQA
BABQAewADAAEECQACAA4AjwADAAEECQADABQAnQADAAEECQAEABQAsQADAAEECQAFABYAxQADAAE
ECQAGABQA2wADAAEECQAKAFgA7wADAAEECQALACQBRYbLlXNiLWljb25zUmVndWxhcmUtc2ItaWVn
vbNlLXNiLWljb25zVmVyc2lubiAxLjBlLXNiLWljb25zRm9udCBnZW5lcmF0ZWQgdXNpbmcgU3I
uY2Zlc2lubiBNZXRYbyBTdHVkaW93d3cuc3luY2Zlc2lubi5jb20AIAblAC0AcwBiAC0AaQBjAG8
AbgBzAFIAZQBnAHUAbABhAHIAZQAtAHMAYGAtAGkAYwBvAG4AcwBlAC0AcwBiAC0AaQBjAG8AbgB
zAFYAZQByAHMAaQBvAG4AIAAaAC4AMABlAC0AcwBiAC0AcwBiAG8AbgBzAEYAbwBuAHQAIAbnAGU
AbgBlAHIAZQB0AGUAZAAgAHUAcwBpAG4AZwAgAFMAeQBuaGMAZgBlAHMAaQBvAG4AIAbnAGUAdAB
yAG8AIABTahQAdQBkAGkAbwB3AHcAdwAuAHMAeQBuaGMAZgBlAHMAaQBvAG4ALgBjAG8AbQAAAAA
CAAAAAAAAAAAoAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAADIBAgEDAQQBBQEGAQcBCAEJAQoBCwEMAQ0
BDgEPARABEQESARMBFAEVARYBFwEYARKbGgEbArWBHQEeAR8BIAEHASIBIwEkASUBJgEnASgBKQE
qASsBLAEtAS4BLwEwATEBMgEzAAltYwlsLXNlbnQLaW5ib3gtMDIttd2YLaW5ib3gtMDEtd2YFaW5
ib3gPb3BlbiItZXNzYwdlLXdmDGFycm93aGVhZC0wMRBhcnJvd2hlYWQtdG9wLXdmCXJhdGluZy1
3ZgtYXRXpbmctLS0wMxPhcnJvd2hlYWQtdG93bi0wMQlmb2xkZXItMDMIdXNlcnMtd2YIY2xvY2s
tMDIGdXBSb2FkCG9uZWRYaXZlEWNsb3VklWRvd25sb2FkLXdmD3dvcmtb2ZmbGluZS13ZglzZW
FyY2gttd2YPbm90ZS1tZW1vLTAxLXdmDGltcG9ydGFudC13ZglzZW50LW1haWwIaW5ib3gttd2YLBWF
pbC0tLXNlbnQJdXNlcilYWNrDHVzZXItcHJvZmlsZS05hbGFybS1jbG9jay13Zg5zZWfyY2gttdG1
tZS13ZgtiZWxsLW5ldy13ZhdidXZlcmFnZS1jb2NrdGFpbC0wNC13ZhRiZXZlcmFnZS1jb2NrdGF
pbC0wMgdib3dsLXdmB2Nha2UtMDIHY2FrZS13Zgtjb2ZmZWUtLS0wMQ9jb2ZmZWUtY3VwLS0tMDI
LY29mZmVlLWJlYW4Nd2luZS1nbGFfcy0wMQRiZS0wMgFfYXJtLTAxLXdmLQlhbGFybS0tMDIHYXR
obGV0ZQdnaWZ0LTAxB2dpZnQtd2YJYmFsbC0tLTAzB2FyY2hlcnkKYmFza2V0YmFsbAlhZXJvcGx
hbmUNYmljeWNsZS0wMy13ZglhdG0tMDIttd2YAAAAA format('trueType');
font-weight: normal;
font-style: normal;
}
[class^="sf-icon-"],
[class*=" sf-icon-"] {
    speak: none;
    font-size: 15px;
    font-style: normal;
    font-weight: normal;
    font-variant: normal;
    text-transform: none;
    line-height: 1;
    -webkit-font-smoothing: antialiased;
    -moz-osx-font-smoothing: grayscale;
}
/* end of custom generated icons styles */
</style>

```

```
{% previewsample "page.domainurl/code-snippet/calendar/islamic-calendar-cs1" %}
```

## Two way binding in Vue Calendar component

Two-way binding can be achieved by using the `v-model` directive in Vue. In the following sample, when you change the value in one Calendar component, `v-model` will automatically update the value in the other Calendar.

The following example demonstrates how to set the **two-way-binding** in the Calendar.

## APP.VUE

```

<template>
<div id="app">
  <div id="wrapper1">
    <ejs-calendar id="calendar" v-model="date"></ejs-calendar>
  </div>
  <div id="wrapper2">
    <ejs-calendar id="calendar1" v-model="date"></ejs-calendar>
  </div>
</div>
</template>
<script>
import Vue from "vue";
import { CalendarPlugin } from "@syncfusion/ej2-vue-calendars";
Vue.use(CalendarPlugin);
export default {
  data: function() {
    return {
      date: new Date()
    };
  }
}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-buttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-
calendars/styles/material.css";
#wrapper1{
  max-width: 300px;
  float: left;
  margin-left: 100px;
}
#wrapper2{
  max-width: 300px;
  float: right;
  margin-right: 100px;
}
</style>

```

{% previewsample "page.domainurl/code-snippet/calendar/two-way-cs1" %}

### Style appearance in Vue Calendar component

The following content provides the exact CSS structure that can be used to modify the control's appearance based on the user preference.

#### Customizing the background color for the Calendar

Use the following CSS to customize the background color and border for the Calendar.

,

*/ To specify background color and border /*

```

.e-calendar {
background-color: peachpuff;
border: 3px solid red;

```

```
}  
`
```

### Customizing the Calendar date elements on hovering

Use the following CSS to customize the date elements on hovering in the Calendar.

```
`
```

*/ To specify background color, color, and border /*

```
.e-calendar .e-content td:hover span.e-day, .e-calendar .e-content td:focus span.e-day, .e-bigger.e-small  
.e-calendar .e-content td:hover span.e-day, .e-bigger.e-small .e-calendar .e-content td:focus span.e-day  
{  
background-color: red;  
border: 2px solid;  
color: #212529;  
}
```

```
`
```

### Customizing the border of date cell grid

Use the following CSS to add the border to the date cell grid.

```
`
```

*/ To specify border /*

```
.e-calendar .e-content span.e-day, .e-bigger.e-small .e-calendar .e-content span.e-day {  
border: 1px solid;  
}
```

```
`
```

### Customizing the Calendar title

Use the following CSS to customize the Calendar title.

```
`
```

*/ To specify color and font size /*

```
.e-calendar .e-header .e-title, .e-bigger.e-small .e-calendar .e-header .e-title {  
color: black;  
font-size: 20px;  
}
```

```
`
```

### Customizing the previous and next icon

Use the following CSS to customize the previous and next icon.

```
`
```

*/ To specify color and border /*

```
.e-calendar .e-header span, .e-bigger.e-small .e-calendar .e-header span {  
border: 1px solid;  
color: chocolate;  
}  
`
```

#### Customizing the footer button

Use the following CSS to customize the footer button.

```
`  
  
/ To specify background color, color, and border-color /  
  
.e-calendar .e-btn.e-today.e-flat.e-primary, .e-calendar .e-css.e-btn.e-today.e-flat.e-primary {  
background-color: red;  
border-color: black;  
color: black;  
}  
`
```

#### Customizing the selected date cell grid

Use the following CSS to customize the selected date cell grid in Calendar.

```
`  
  
/ To specify background color and color /  
  
.e-calendar .e-content td.e-focused-date.e-today span.e-day {  
background-color: maroon;  
color: #fff;  
}  
`
```

#### Customizing the content header in Calendar

Use the following CSS to customize the content header in Calendar.

```
`  
  
/ To specify background /  
  
.e-calendar .e-content thead, .e-bigger.e-small .e-calendar .e-content thead {  
background: aquamarine;  
}  
`
```



## How To

## Set clear button in calendar in Vue Calendar component

To configure **clear** button in Calendar UI, do the following:

1. To the **created** event of the Calendar, add the required elements to make the clear button visible. In the following example, div with Essential JS 2 button component is used.
2. When the **e-footer** class is used, the div tag acts as the footer.
3. Using these button, selected date can be cleared.
4. Bind the required event handler to the button tags to update the value.

**APP.VUE**

```
<template>
  <div id="app">
    <div class='wrap'>
      <ejs-calendar id='calendar' :created='onCreate'
ref="CalendarInstance"></ejs-calendar>
    </div>
  </div>
</template>
<script>
import Vue from "vue";
import { CalendarPlugin } from "@syncfusion/ej2-vue-calendars";
Vue.use(CalendarPlugin);
export default {
  methods: {
    onCreate: function(args) {
      let calendarObj = this.$refs.CalendarInstance;
      let clearBtn = document.createElement('button');
      let footerElement = document.getElementsByClassName('e-
footer-container')[0];
      clearBtn.className = 'e-btn e-clear e-flat';
      clearBtn.textContent = 'Clear';
      footerElement.prepend(clearBtn);
      calendarObj.$el.appendChild(footerElement);
      document.querySelector('.e-footer-container .e-
clear').addEventListener('click', function () {
        calendarObj.value = null;
        calendarObj.dataBind();
      });
    }
  }
}
</script>
<style>
  @import "../node_modules/@syncfusion/ej2-base/styles/material.css";
  @import "../node_modules/@syncfusion/ej2-buttons/styles/material.css";
  @import "../node_modules/@syncfusion/ej2-vue-
calendars/styles/material.css";
  .wrap {
    margin: 35px auto;
    width: 240px;
  }
  .e-clear {
```



```
margin-right: 81px;
}
</style>
```

{% previewsample "page.domainurl/code-snippet/calendar/min-max-cs9" %}

### Customize the calendar day header in Vue Calendar component

You can change the format of the day that to be displayed in header using [dayHeaderFormat](#) property. By default, the format is **Short**.

You can find the possible formats on below.

Name	Description
Short	Sets the short format of day name (like Su ) in day header.
Narrow	Sets the single character of day name (like S ) in day header.
Abbreviated	Sets the min format of day name (like Sun ) in day header.
Wide	Sets the long format of day name (like Sunday ) in day header.

### APP.VUE

```
<template>
  <div id="container">
    <div id='calendar'>
      <ejs-calendar dayHeaderFormat="Short" ref="calendarObj"></ejs-
calendar>
    </div>
    <div id="format">
      <label id="custom-input-label">Header Format Types</label>
      <ejs-dropdownlist id="select" :fields = "fields" :index =
"currentIndex" :dataSource = "items" :popupHeight= "popupHeight" :change=
"formatHandler">
      </ejs-dropdownlist>
    </div>
  </div>
</template>
<script>
import Vue from "vue";
import { CalendarPlugin } from "@syncfusion/ej2-vue-calendars";
import { DropDownListPlugin } from "@syncfusion/ej2-vue-dropdowns";
Vue.use(CalendarPlugin);
Vue.use(DropDownListPlugin);
export default {
  data () {
    return {
      items: [
        { sizeVal: 'Short' , sizeTxt: 'Short' },
        { sizeVal: 'Narrow' , sizeTxt: 'Narrow' },
        { sizeVal: 'Abbreviated' , sizeTxt: 'Abbreviated' },
        { sizeVal: 'Wide' , sizeTxt: 'Wide' },
      ],
      fields: { text: 'sizeTxt', value: 'sizeVal' },
      popupHeight: 200,
    }
  }
}
```

```

        currentIndex: 0,
        formatHandler: (args) => {
            this.$refs.calendarObj.dayHeaderFormat = args.value;
        }
    }
}
</script>
<style>
    @import "../node_modules/@syncfusion/ej2-base/styles/material.css";
    @import "../node_modules/@syncfusion/ej2-buttons/styles/material.css";
    @import "../node_modules/@syncfusion/ej2-vue-
calendars/styles/material.css";
    @import "../node_modules/@syncfusion/ej2-vue-
dropdowns/styles/material.css";
    #container{
        width: 100%;
    }
    #calendar {
        width: 60%;
        margin: 30px auto;
        display: inline-flex;
    }
    #format {
        width: 20%;
        margin-left: 8%;
        display: inline-block;
    }
    #select {
        height: 30px;
        width: 150px;
        margin-top: 10px;
    }
    #custom-input-label{
        font-size: 15px;
        font-weight: bold
    }
</style>

```

{% previewsample "page.domainurl/code-snippet/calendar/header-format-cs1" %}

### Show dates of other months in Vue Calendar component

The following example demonstrates how to show dates of other months.

Using the styles below, you can bring the dates of other months to visibility from its hidden state.

,

```

<style>
.e-calendar .e-content tr.e-month-hide,
.e-calendar .e-content td.e-other-month>span.e-day {
display: block;
}

```

```
.e-calendar .e-content td.e-month-hide,
.e-calendar .e-content td.e-other-month {
pointer-events: auto;
touch-action: auto;
}
</style>
,
```

### APP.VUE

```
<template>
  <div id="app">
    <div class='wrap'>
      <ejs-calendar id='calendar'></ejs-calendar>
    </div>
  </div>
</template>
<script>
import Vue from "vue";
import { CalendarPlugin } from "@syncfusion/ej2-vue-calendars";
Vue.use(CalendarPlugin);
export default {
  return {
  }
}
</script>
<style>
  @import "../node_modules/@syncfusion/ej2-base/styles/material.css";
  @import "../node_modules/@syncfusion/ej2-buttons/styles/material.css";
  @import "../node_modules/@syncfusion/ej2-vue-
calendars/styles/material.css";
  .wrap {
    margin: 0px auto;
    max-width: 250px;
  }
  .e-calendar .e-content tr.e-month-hide,
  .e-calendar .e-content td.e-other-month>span.e-day {
    display: block;
  }
  .e-calendar .e-content td.e-month-hide,
  .e-calendar .e-content td.e-other-month {
    pointer-events: auto;
    touch-action: auto;
  }
</style>
```

{% previewsample "page.domainurl/code-snippet/calendar/min-max-cs10" %}

Select a sequence of dates in calendar in Vue Calendar component

The following example demonstrates how to select the week dates of chosen date in the Calendar using [values](#) property, when [isMultiSelection](#) property is enabled. Methods of Moment.js is used in this sample for calculating the start and end of week from the selected date.

**APP.VUE**

```

<template>
  <div id="app">
    <div class="wrap" style="height:357px; display: inline-block;">
      <ejs-calendar id='calendar' ref="CalendarInstance"
        :isMultiSelection="isMultiSelection" :change="onChange"></ejs-calendar>
    </div>
    <div id="btncontainer">
      <ejs-button id="week" class="e-btn" v-
        on:click.native="onweekChange"> Week </ejs-button>
      <ejs-button id="workweek" class="e-btn" v-
        on:click.native="onworkweekChange"> Work Week </ejs-button>
    </div>
  </div>
</template>
<script>
import Vue from "vue";
import { CalendarPlugin } from "@syncfusion/ej2-vue-calendars";
import { ButtonPlugin } from "@syncfusion/ej2-vue-buttons";
import moment from "moment";
Vue.use(CalendarPlugin);
Vue.use(ButtonPlugin);
export default {
  data() {
    return {
      isMultiSelection: true
    }
  },
  methods: {
    onChange: function(args) {
      let startOfWeek = moment(args.value).startOf('week');
      let endOfWeek = moment(args.value).endOf('week');
      if
        (this.$refs.CalendarInstance.$el.classList.contains('workweek')) {
          this.getWeekArray(startOfWeek.day(1), endOfWeek.day(5));
        } else if
        (this.$refs.CalendarInstance.$el.classList.contains("week")) {
          this.getWeekArray(startOfWeek, endOfWeek);
        }
    },
    /*selected current week dates when click the button*/
    onweekChange: function() {
      if
        (this.$refs.CalendarInstance.$el.classList.contains('workweek')) {
          this.$refs.CalendarInstance.$el.classList.remove('workweek')
        }
        this.$refs.CalendarInstance.$el.classList.add('week');
    },
    onworkweekChange: function() {
      if (this.$refs.CalendarInstance.$el.classList.contains('week'))
      {
        this.$refs.CalendarInstance.$el.classList.remove('week')
      }
      this.$refs.CalendarInstance.$el.classList.add('workweek');
    },
    getWeekArray: function(startOfWeek, endOfWeek) {

```

```

        let days = [];
        let day = startOfWeek;
        while (day <= endOfWeek) {
            days.push(day.toDate());
            day = day.clone().add(1, 'd');
        }
        this.$refs.CalendarInstance.values = days;
    }
}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-buttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-calendars/styles/material.css";
.wrap {
    margin: 0px auto;
    max-width: 250px;
}
#app{
    max-width: 550px;
}
#btncontainer{
    float:right;
    margin-left:30px;
    margin-top: 75px;
    margin-right: 10px;
}
button#workweek {
    margin-left: 15px;
}
</style>

```

{% previewsample "page.domainurl/code-snippet/calendar/multi-selection-cs1" %}

### Skip a month in calendar in Vue Calendar component

The following example demonstrates how to skip a month in the Calendar while clicking the previous and next icons. In the example below, the [navigated](#) event is used to skip a month with [navigateTo](#) method.

#### APP.VUE

```

<template>
  <div id="app">
    <div class='wrap'>
      <ejs-calendar id='calendar' :navigated='onNavigate'
      ref='Calendar'></ejs-calendar>
    </div>
  </div>
</template>
<script>
import Vue from "vue";
import { CalendarPlugin } from "@syncfusion/ej2-vue-calendars";
Vue.use(CalendarPlugin);
export default {
  methods: {

```

```

        onNavigate: function(args){
            let date;
            if (args.event.currentTarget.classList.contains('e-next')) {
                //Increments the month while clicking the next icon.
                date = new Date(args.date.setMonth(args.date.getMonth() + 1));
            }
            if (args.event.currentTarget.classList.contains('e-prev')) {
                //Decrements the month while clicking the previous icon.
                date = new Date(args.date.setMonth(args.date.getMonth() - 1));
            }
            if (args.view == 'month') {+
                this.$refs.Calendar.navigateTo('month', date);
            }
        }
    }
}
</script>
<style>
    @import "../node_modules/@syncfusion/ej2-base/styles/material.css";
    @import "../node_modules/@syncfusion/ej2-buttons/styles/material.css";
    @import "../node_modules/@syncfusion/ej2-vue-
    calendars/styles/material.css";
    .wrap {
        margin: 0px auto;
        max-width: 250px;
    }
</style>

```

{% previewsample "page.domainurl/code-snippet/calendar/min-max-cs11" %}

Render the calendar with week numbers in Vue Calendar component

You can enable `weekNumbers` in the Calendar by using the [weekNumber](#) property.

#### APP.VUE

```

<template>
    <div id="app">
        <div class='wrap'>
            <ejs-calendar id='calendar' :weekNumber='weeknumber'></ejs-
            calendar>
        </div>
    </div>
</template>
<script>
import Vue from "vue";
import { CalendarPlugin } from "@syncfusion/ej2-vue-calendars";
Vue.use(CalendarPlugin);
export default {
    data () {
        return {
            weeknumber : true
        }
    }
}
</script>
<style>

```

```

@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-buttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-
calendars/styles/material.css";
.wrap {
    margin: 0px auto;
    max-width: 250px;
}
</style>

```

{% previewsample "page.domainurl/code-snippet/calendar/min-max-cs8" %}

### Change the first day of week in Vue Calendar component

The Calendar provides an option to change the first day of the week by using the [firstDayOfWeek](#) property. Generally, the day of the week starts from 0 (Sunday) and ends with 6 (Saturday).

By default, the first day of the week is culture specific.

The following example shows the Calendar with **Tuesday** as the first day of the week.

### APP.VUE

```

<template>
  <div id="app">
    <div class='wrap'>
      <ejs-calendar id='calendar' :firstDayOfWeek='firstDay'></ejs-
calendar>
    </div>
  </div>
</template>
<script>
import Vue from "vue";
import { CalendarPlugin } from "@syncfusion/ej2-vue-calendars";
Vue.use(CalendarPlugin);
export default {
  data () {
    return {
      firstDay : 2
    }
  }
}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-buttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-
calendars/styles/material.css";
.wrap {
    margin: 0px auto;
    max-width: 250px;
}
</style>

```

{% previewsample "page.domainurl/code-snippet/calendar/getting-started-cs2" %}

## Card

### Getting Started with the Vue Card Component in Vue 2

This article provides a step-by-step guide for setting up a Vue 2 project using [Vue-CLI](#) and integrating the Syncfusion Vue Card component using the [Composition API](#) / [Options API](#).

#### Prerequisites

[System requirements for Syncfusion Vue UI components](#)

#### Dependencies

The Card is pure CSS component so no other package dependencies are needed to render the Card.

```
`js
|-- @syncfusion/ej2-layouts
`
```

#### Setting up the Vue 2 project

To generate a Vue 2 project using Vue-CLI, use the [vue create](#) command. Follow these steps to install Vue CLI and create a new project:

```
`bash
npm install -g @vue/cli
vue create quickstart
cd quickstart
npm run serve
`

or

`bash
yarn global add @vue/cli
vue create quickstart
cd quickstart
yarn run serve
`
```

When creating a new project, choose the option **Default ([Vue 2] babel, eslint)** from the menu.

```
E:\samples>vue create quickstart

Vue CLI v5.0.8
? Please pick a preset:
  Default ([Vue 3] babel, eslint)
> Default ([Vue 2] babel, eslint)
  Manually select features
```



Once the `quickstart` project is set up with default settings, proceed to add Syncfusion components to the project.

### Adding Syncfusion packages

Syncfusion packages are available at [npmjs.com](https://www.npmjs.com). To use Vue components, install the required npm package.

This article uses the [Vue card component](#) as an example. Install the `@syncfusion/ej2-vue-layouts` package by running the following command:

```
`bash
```

```
npm install @syncfusion/ej2-vue-layouts --save
```

```
`
```

or

```
`bash
```

```
yarn add @syncfusion/ej2-vue-layouts
```

```
`
```

### Import Syncfusion CSS styles

You can import themes for the Syncfusion Vue component in various ways, such as using CSS or SASS styles from npm packages, CDN, [CRG](#) and [Theme Studio](#). Refer to [themes topic](#) to know more about built-in themes and different ways to refer to themes in a Vue project.

In this article, the `Material` theme is applied using CSS styles, which are available in installed packages. The necessary `Material` CSS styles for the Card component and its dependents were imported into the `<style>` section of `src/App.vue` file.

#### ~/SRC/APP.VUE

```
<style>
@import '../node_modules/@syncfusion/ej2-base/styles/material.css';
@import '../node_modules/@syncfusion/ej2-vue-layouts/styles/material.css';
</style>
```

### Creating Vue Sample

Add the HTML div element with `e-card` class into the `<template>` section of the `App.vue` file in `src` directory.

#### ~/SRC/APP.VUE

```
<template>
<div id="app">
<div class = "e-card">
Sample Card
</div>
</div>
</template>
```

### Adding a header and content

You can create Card with a header in a specific structure. For adding header you need to create a `div` element with `e-card-header` class added.

- You can include heading inside the Card header by adding a `div` element with `e-card-header-caption` class, and also content will be added by adding element with `e-card-content`. For detailed information, refer to the [Header and Content](#).

### ~/SRC/APP.VUE

```
<div class = "e-card">                                --> Root Element
<div class="e-card-header">                            --> Root Header Element
<div class="e-card-header-caption">                    --> Root Heading Element
<div class="e-card-header-title"></div>                --> Heading Title Element
</div>
<div class="e-card-content"></div>                    --> Card content Element
</div>
</div>
```

Here is the summarized code for the above steps in the `src/App.vue` file:

### APP.VUE

```
<template>
  <div id="app">
    <div id='container' style="margin:50px auto 0; width:100%;">
      <br>
      <div tabindex="0" class="e-card" id="basic">
        <div class="e-card-header">
          <div class="e-card-header-caption">
            <div class="e-card-title">Advanced UWP</div>
          </div>
        </div>
        <div class="e-card-content">
          Communicating with Windows 10 and Other Apps, the second in
          a five-part series written by Succinctly series
          author Matteo Pagani. To download the complete white paper,
          and other papers in the series, visit
          the White Paper section of Syncfusion's Technology Resource
          Portal.
        </div>
      </div>
    </div>
  </div>
</template>
<style>
  @import "../node_modules/@syncfusion/ej2-base/styles/material.css";
  @import "../node_modules/@syncfusion/ej2-vue-layouts/styles/material.css";
</style>
```

### Running the Project

To run the project, use the following command:

```
`bash
```

```
npm run serve
```

```
,
```

```
or
```

```
`bash
```

```
yarn run serve
```

```
,
```

```
{% previewsample "page.domainurl/code-snippet/card/getting-started-cs1" %}
```

See Also

- [How to add a header and content](#)

### Getting Started with the Vue Card Component in Vue 3

This article provides a step-by-step guide for setting up a [Vite](#) project with a JavaScript environment and integrating the Syncfusion Vue Card component using the [Composition API](#) / [Options API](#).

The **Composition API** is a new feature introduced in Vue.js 3 that provides an alternative way to organize and reuse component logic. It allows developers to write components as functions that use smaller, reusable functions called composition functions to manage their properties and behavior.

The **Options API** is the traditional way of writing Vue.js components, where the component logic is organized into a series of options that define the component's properties and behavior. These options include data, methods, computed properties, watchers, lifecycle hooks, and more.

#### Prerequisites

##### [System requirements for Syncfusion Vue UI components](#)

##### Set up the Vite project

A recommended approach for beginning with Vue is to scaffold a project using [Vite](#). To create a new Vite project, use one of the commands that are specific to either NPM or Yarn.

```
`bash
```

```
npm create vite@latest
```

```
,
```

```
or
```

```
`bash
```

```
yarn create vite
```

```
,
```

Using one of the above commands will lead you to set up additional configurations for the project as below:

1. Define the project name: We can specify the name of the project directly. Let's specify the name of the project as **my-project** for this article.

```
`bash
```

? Project name: » my-project

,

2. Select **Vue** as the framework. It will create a Vue 3 project.

`bash

? Select a framework: » - Use arrow-keys. Return to submit.

Vanilla

Vue

React

Preact

Lit

Svelte

Others

,

3. Choose **JavaScript** as the framework variant to build this Vite project using JavaScript and Vue.

`bash

? Select a variant: » - Use arrow-keys. Return to submit.

JavaScript

TypeScript

Customize with create-vue ↗

Nuxt ↗

,

4. Upon completing the aforementioned steps to create the **my-project**, run the following command to install its dependencies:

`bash

cd my-project

npm install

,

or

`bash

cd my-project

yarn install

,

Now that `my-project` is ready to run with default settings, let's add Syncfusion components to the project.

### Add Syncfusion Vue packages

Syncfusion Vue component packages are available at [npmjs.com](https://www.npmjs.com). To use Syncfusion Vue components in the project, install the corresponding npm package.

This article uses the [Vue Card component](#) as an example. To use the Vue Card component in the project, the `@syncfusion/ej2-vue-layouts` package needs to be installed using the following command:

```
`bash
```

```
npm install @syncfusion/ej2-vue-layouts --save
```

```
,
```

or

```
`bash
```

```
yarn add @syncfusion/ej2-vue-layouts
```

```
,
```

### Import Syncfusion CSS styles

You can import themes for the Syncfusion Vue component in various ways, such as using CSS or SASS styles from npm packages, CDN, [CRG](#) and [Theme Studio](#). Refer to [themes topic](#) to know more about built-in themes and different ways to refer to themes in a Vue project.

In this article, `Material` theme is applied using CSS styles, which are available in installed packages. The necessary `Material` CSS styles for the Card component and its dependents were imported into the `<style>` section of `src/App.vue` file.

### ~/SRC/APP.VUE

```
<style>
@import '../node_modules/@syncfusion/ej2-base/styles/material.css';
@import '../node_modules/@syncfusion/ej2-vue-layouts/styles/material.css';
</style>
```

The order of importing CSS styles should be in line with its dependency graph.

### Adding a header and content

You can create Card with a header in a specific structure. For adding header you need to create a `div` element with `e-card-header` class added.

- You can include heading inside the Card header by adding a `div` element with `e-card-header-caption` class, and also content will be added

by adding element with `e-card-content`. For detailed information, refer to the [Header and Content](#).

### ~/SRC/APP.VUE

```
<div class = "e-card">                                --> Root Element
<div class="e-card-header">                            --> Root Header Element
<div class="e-card-header-caption">                    --> Root Heading Element
```

```

<div class="e-card-header-title"></div>    --> Heading Title Element
</div>
<div class="e-card-content"></div>        --> Card content Element
</div>

```

### Adding Syncfusion Vue Card component in the application

You have completed all the necessary configurations needed for rendering the Syncfusion Vue component. Add the HTML div element with **e-card** class into the `<template>` section of the `App.vue` file in `src` directory.

#### COMPOSITION API (~SRC/APP.VUE)

```

<template>
<div id='container' style="margin:50px auto 0; width:100%;">
<br>
<div tabindex="0" class="e-card" id="basic">
<div class="e-card-header">
<div class="e-card-header-caption">
<div class="e-card-title">Advanced UWP</div>
</div>
</div>
<div class="e-card-content">
Communicating with Windows 10 and Other Apps, the second in a five-part
series written by Succinctly series
author Matteo Pagani. To download the complete white paper, and other papers
in the series, visit
the White Paper section of Syncfusion's Technology Resource Portal.
</div>
</div>
</div>
</template>
<script setup>
const data = [];
</script>
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-layouts/styles/material.css";
</style>

```

#### OPTIONS API (~SRC/APP.VUE)

```

<template>
<div id='container' style="margin:50px auto 0; width:100%;">
<br>
<div tabindex="0" class="e-card" id="basic">
<div class="e-card-header">
<div class="e-card-header-caption">
<div class="e-card-title">Advanced UWP</div>
</div>
</div>
<div class="e-card-content">
Communicating with Windows 10 and Other Apps, the second in a five-part
series written by Succinctly series

```

```
author Matteo Pagani. To download the complete white paper, and other papers
in the series, visit
the White Paper section of Syncfusion's Technology Resource Portal.
</div>
</div>
</div>
</template>
<script>
export default {
  name: 'App',
  components: {
  },
  data() {
    return {
    };
  }
}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-layouts/styles/material.css";
</style>
```

### Run the project

To run the project, use the following command:

```
`bash
```

```
npm run dev
```

```
,
```

or

```
`bash
```

```
yarn run dev
```

```
,
```

The output will appear as follows:

#### Advanced UWP

Communicating with Windows 10 and Other Apps, the second in a five-part series written by Succinctly series author Matteo Pagani. To download the complete white paper, and other papers in the series, visit the White Paper section of Syncfusion's Technology Resource Portal.

### Header content in Vue Card component

#### Header

The Card can be created with header title, sub title and images. For adding header you need to create `div` element with the class `e-card-header` added.

Card provides below elements and corresponding class definitions to include header.

## Elements | Description

**e-card-header-caption** | To group the title and subtitle within the header which acts as wrapper.

**e-card-header-title** | Main title text within the header.

**e-card-sub-title** | A sub-title within the header.

**e-card-header-image** | To include heading image within the header.

**e-card-corner** | To add rounded corner for the image.

### *Title and Subtitle*

For adding header to the Card, you need to create wrapper **div** element with **e-card-header-caption** class.

- Place the **div** element with **e-card-header-title** class inside the header caption for adding main title.
- Place the **div** element with **e-card-sub-title** class inside the header caption element for adding sub-title.

### *Image*

Card header has an option for adding images in the header. It is aligned with either before or after the header based on the HTML element positioned in the header structure.

- The header image can be added by creating a **div** element with **e-card-header-image** class which can be placed before or after the header caption wrapper element.

## **APP.VUE**

```
<template>
  <div id="app">
    <div style="margin: 50px;">
      <div tabindex="0" class="e-card">
        <div class="e-card-header">
          <div class="e-card-header-image football e-card-
corner"></div>
          <div class="e-card-header-caption">
            <div class="e-card-header-title"> Laura Callahan</div>
            <div class="e-card-sub-title">Sales Coordinator and
Representative</div>
          </div>
        </div>
      </div>
    </div>
    <div style="margin-left: 50px;margin-top:30px">
      <div tabindex="0" class="e-card">
        <div class="e-card-header">
          <div class="e-card-header-caption">
            <div class="e-card-header-title"> Laura Callahan</div>
            <div class="e-card-sub-title">Sales Coordinator and
Representative</div>
          </div>
          <div class="e-card-header-image football"></div>
        </div>
      </div>
    </div>
  </div>
</template>
```



```

        </div>
      </div>
    </div>
  </template>
  <script>
import Vue from 'vue';
export default {
  name: 'app',
}
</script>
<style>
  @import '../node_modules/@syncfusion/ej2-base/styles/material.css';
  @import '../node_modules/@syncfusion/ej2-vue-
layouts/styles/material.css';
.e-card .e-card-header .e-card-header-image.football {
  background-image: url('../football.png');
}
.e-card {
  width: 300px
}
</style>

```

{% previewsample "page.domainurl/code-snippet/card/header-content/image-cs1" %}

## Content

Content in Card holds texts, images, links and all possible HTML elements. Its adaptable within the Card root element.

- Create a `div` element with the class `e-card-content`.
- Place content `div` element in the Card root element or within any Card inner elements.

## APP.VUE

```

<template>
  <div id="app">
    <div tabindex="0" class="e-card">
      <div class="e-card-header">
        <div class="e-card-header-image football"></div>
        <div class="e-card-header-caption">
          <div class="e-card-header-title"> Laura Callahan</div>
          <div class="e-card-sub-title">Sales Coordinator and
Representative</div>
        </div>
      </div>
      <div class="e-card-content">
        Laura received a BA in psychology from the University of
Washington. She has also completed a course in business French. She reads
and writes French.
      </div>
    </div>
  </div>
</template>
<script>
import Vue from 'vue';

```

```
export default {
  name: 'app',
}
</script>
<style>
  @import '../node_modules/@syncfusion/ej2-base/styles/material.css';
  @import '../node_modules/@syncfusion/ej2-vue-
layouts/styles/material.css';
  .e-card .e-card-header .e-card-header-image.football {
    background-image: url('../football.png');
  }
  .e-card {
    width: 300px
  }
</style>
```

{% previewsample "page.domainurl/code-snippet/card/header-content/content-cs1" %}

## Card image in Vue Card component

### Images

The Card supports to include images within the elements, you can add image as direct element anywhere inside card root by adding the

**e-card-image** class to **div** element. Using the class defined, you can write CSS styles to load images to that element.

By default, card images occupies full width of its parent element.

,

```
<div class = "e-card">
  <div class="e-card-image">
  </div>
</div>
```

,

### Title

Card image is supported to include a title or caption for the image. By default, Title is placed over the image on left-bottom position with

overlay.

,

```
<div class = "e-card">
  <div class="e-card-image">
    <div class="e-card-title"></div>
  </div>
</div>
```

,

**APP.VUE**

```

<template>
  <div id="app">
    <div class="e-card">
      <div class="e-card-image">
        <div class="e-card-title">JavaScript </div>
      </div>
      <div class="e-card-content"> JavaScript Succinctly was written
to give readers an accurate, concise examination of JavaScript objects and
their supporting nuances, such as complex values, primitive values, scope,
inheritance, the head object, and more. </div>
    </div>
  </div>
</template>
<script>
import Vue from 'vue';
export default {
  name: 'app',
}
</script>
<style>
  @import '../node_modules/@syncfusion/ej2-base/styles/material.css';
  @import '../node_modules/@syncfusion/ej2-vue-
layouts/styles/material.css';
  .e-card-image {
    background: url('../sample.jpg');
    height: 160px;;
  }
  .e-card {
    width: 300px;
    margin: auto;
  }
</style>

```

{% previewsample "page.domainurl/code-snippet/card/images-divider/title-cs1" %}

**Divider**

Divider used to separate the elements inside the card. You can add divider inside the card elements to separate it.

- Place the `div` element with `e-card-separator` class inside the card element for adding a divider.

**APP.VUE**

```

<template>
  <div id="app">
    <div tabindex="0" class="e-card" id="basic">
      <div class="e-card-title">Explore Cities</div>
      <div class="e-card-separator"></div>
      <div class="e-card-content">
        Sydney is a city on the east coast of Australia. Sydney is
the capital city of New South Wales. About four million people
live in Sydney which makes it the biggest city in Oceania.
      </div>
    </div>
  </div>

```

```

        <div class="e-card-separator"></div>
        <div class="e-card-content">
            New York City has been described as the cultural, financial,
            and media capital of the world, and exerts a significant impact
            upon commerce and etc.,
        </div>
        <div class="e-card-separator"></div>
        <div class="e-card-content">
            Malaysia is one of the Southeast Asian countries, on a
            peninsula of the Asian continent, to a certain extent; it can be recognized
            as part of the Asian continent.
        </div>
    </div>
</template>
<script>
import Vue from 'vue';
export default {
    name: 'app',
}
</script>
<style>
    @import '../node_modules/@syncfusion/ej2-base/styles/material.css';
    @import '../node_modules/@syncfusion/ej2-vue-
    layouts/styles/material.css';
    #container {
        visibility: hidden;
    }
    #loader {
        color: #008cff;
        height: 40px;
        left: 45%;
        position: absolute;
        top: 45%;
        width: 30%;
    }
    .e-card {
        width: 300px
    }
</style>

```

{% previewsample "page.domainurl/code-snippet/card/images-divider/divider-cs1" %}

See Also

- [How to customize the card image title position](#)

## Action buttons in Vue Card component

You can include Action buttons within the Card and customize them. Action button is a `div` element with `e-card-actions` class followed by button tag or anchor tag within the card root element.

- For adding action buttons you can create button or anchor tag with `e-card-btn` class within the card action element.

```

<div class = "e-card">
<div class="e-card-actions">
<button class="e-card-btn"></button>
<a href="#"></a>
</div>
</div>

```

### Vertical

By default, action buttons positioned in horizontal alignment , and also it can be aligned to show in vertical alignment by adding `e-card-vertical` class.

```

<div class = "e-card">
<div class="e-card-actions e-card-vertical">
<button class="e-card-btn">More</button>
<a href="#">Share</a>
</div>
</div>

```

### APP.VUE

```

<template>
  <div id="app">
    <div style="margin: 50px;">
      <div class="e-card" style="max-width:400px">
        <div class="e-card-header-title">Eiffel Tower</div>
        <div class="e-card-content">
          The Eiffel Tower is acknowledged as the universal symbol of
          Paris and France.
        </div>
        <div class="e-card-actions">
          <button class="e-card-btn">
            
          </button>
          <button class="e-card-btn">
            
          </button>
          <button class="e-card-btn">
            
          </button>
        </div>
      </div>
    </div>
  </div>
</template>

```

```

        </div>
      </div>
      <div style="margin-left: 50px;">
        <div class="e-card" style="max-width:400px">
          <div class="e-card-header-title">Eiffel Tower</div>
          <div class="e-card-content">
            The Eiffel Tower is acknowledged as the universal symbol
of Paris and France.
          </div>
          <div class="e-card-actions e-card-vertical">
            <button class="e-card-btn">LIKE</button>
            <button class="e-card-btn">SHARE</button>
          </div>
        </div>
      </div>
    </div>
  </template>
  <script>
import Vue from 'vue';
export default {
  name: 'app',
}
</script>
<style>
  @import '../node_modules/@syncfusion/ej2-base/styles/material.css';
  @import '../node_modules/@syncfusion/ej2-vue-
layouts/styles/material.css';
#container {
  visibility: hidden;
}
#loader {
  color: #008cff;
  height: 40px;
  left: 45%;
  position: absolute;
  top: 45%;
  width: 30%;
}
.e-card {
  width: 300px
}
</style>

```

{% previewsample "page.domainurl/code-snippet/card/cardactionbtn-cs1" %}

See Also

- [How to integrate other component inside the card](#)

### Horizontal in Vue Card component

By default, all the card elements are aligned vertically one after the other as in the DOM. You can achieve the element to align horizontally as well by adding the class `e-card-horizontal` in the root card element.

## Stacked cards

- An horizontally aligned card can push a specific column to align vertical using `e-card-stacked` class. This will align the stacked section vertically aligned differentiating from horizontal layout.

## Class | Description

`e-card-horizontal` | To align card elements horizontally.

`e-card-stacked` | To align elements vertically within the horizontal layout.

```

<div tabindex="0" class="e-card e-card-horizontal">
 --> Aligned in horizontal
<div class="e-card-stacked">    --> Aligned in horizontal
// Inside the element all are aligned vertical directions
</div>
</div>

```

**APP.VUE**

```

<template>
  <div id="app">
    <div style="margin: 50px;display: flex;flex-direction: row;justify-
content: center;">
      <div tabindex="0" class="e-card e-card-horizontal"
style="width:400px">
        
        <div class="e-card-stacked">
          <div class="e-card-header">
            <div class="e-card-header-caption">
              <div class="e-card-header-title">Philips
Trimmer</div>
            </div>
          </div>
          <div class="e-card-content">
            Powered by the innovative DuraPower Technology which
            optimizes power consumption, Philips trimmers are designed to last longer
            than 4 ordinary trimmers.
          </div>
        </div>
      </div>
    </div>
  </div>
</template>
<script>
import Vue from 'vue';
export default {
  name: 'app',
}
</script>

```

```

<style>
  @import '../node_modules/@syncfusion/ej2-base/styles/material.css';
  @import '../node_modules/@syncfusion/ej2-vue-
layouts/styles/material.css';
  #container {
    visibility: hidden;
  }
  #loader {
    color: #008cff;
    height: 40px;
    left: 45%;
    position: absolute;
    top: 45%;
    width: 30%;
  }
  .e-card {
    width: 300px
  }
</style>

```

{% previewsample "page.domainurl/code-snippet/card/horizontal-cs1" %}

### Style in Vue Card component

The following content provides the exact CSS structure that can be used to modify the control's appearance based on user preference.

#### Customizing the card

Use the following CSS to customize the card properties.

,

```

.e-card {
background-color: aqua;
padding-left: 20px;
margin-bottom: 20px;
}

```

,

#### Customizing the Header element

Use the following CSS to customize the Header element properties.

,

```

.e-card .e-card-header {
font-family: cursive;
font-style: italic;
}

```

,



### Customizing the card content

Use the following CSS to customize the card content properties.

,

```
.e-card .e-card-content {  
font-size: 20px;  
color: gray;  
line-height: initial;  
font-weight: normal;  
}
```

,

### Divider used to separate the elements inside the card

Use the following CSS to customize the Divider used to separate the elements inside the card properties.

,

```
.e-card .e-card-separator {  
padding-bottom: 30px;  
}
```

,

### Including image within card element

Use the following CSS to Include image within card element.

,

```
.e-card .e-card-image {  
background-image: url(images.png);  
background-color: yellow;  
height: 160px;  
}
```

,

### Including a title or caption for the image

Use the following CSS to Include a title or caption for the image.

,

```
.e-card .e-card-image .e-card-title {  
font-family: cursive;  
font-style: italic;  
}
```

,

### To include heading image within the header

Use the following CSS to Include heading image within the header.

```
,  
  
.e-card .e-card-header .e-card-header-image {  
height: 48px;  
width: 48px;  
}  
,
```

### Customizing the Header main title

Use the following CSS to Customize the Header main title.

```
,  
  
.e-card .e-card-header .e-card-header-caption .e-card-header-title {  
font-size: large;  
color: aquamarine;  
}  
,
```

### Customizing the Header subtitle

Use the following CSS to Customize the Header subtitle.

```
,  
  
.e-card .e-card-header .e-card-header-caption .e-card-sub-title {  
font-size: 20px;  
font-variant: all-petite-caps;  
}  
,
```

### Including action buttons or anchor tags

Use the following CSS to Include action buttons or anchor tags.

```
,  
  
.e-card .e-card-actions .e-card-btn {  
padding-left: 20px;  
background-color: wheat;  
}  
,
```

### To align card elements horizontally

Use the following CSS to align card elements horizontally.

```

.e-card .e-card-horizontal {
margin: auto;
width: inherit;
}

```

To align elements vertically within the horizontal layout

Use the following CSS to align elements vertically within the horizontal layout.

```

.e-card .e-card-horizontal .e-card-stacked {
justify-content: flex-start;
margin: initial;
}

```

## How To

Customize the card image title position in Vue Card component

Card Image titles are placed as always Bottom-Left Corner only, You can manually customize to placing titles anywhere over the image by adding styles.

### APP.VUE

```

<template>
  <div id="app">
    <div>
      <div class="e-card">
        <div class="e-card-image">
          <div class="e-card-title">Node.js</div>
        </div>
        <div class="e-card-content">
          Node.js is a wildly popular platform for writing web
          applications that has revolutionized web development in many ways, enjoying
          support across the open source community as well as
          industry.
        </div>
      </div>
    </div>
    <div style="Margin: 5px 0;width:300px">
      <ejs-dropdownlist id='title_position' :dataSource='dropData'
placeholder="Select Position" :change="changed"></ejs-dropdownlist>
    </div>
  </template>
<script>
import Vue from 'vue';
import { DropDownListPlugin } from "@syncfusion/ej2-vue-dropdowns";
Vue.use(DropDownListPlugin);
export default {

```

```

    name: 'app',
    data: function() {
      return {
        dropData: ['BottomLeft', 'TopLeft', 'TopRight', 'BottomRight'];
      };
    }, methods:{
    changed: function(e) {
      var cardEle = document.querySelector('.e-card');
      var titleEle = cardEle.querySelector('.e-card-image .e-card-title');
      titleEle.className = '';
      titleEle.classList.add('e-card-title');
      titleEle.classList.add(e.value.toLowerCase());
    }
    }
  }
</script>
<style>
  @import '../..../node_modules/@syncfusion/ej2-base/styles/material.css';
  @import '../..../node_modules/@syncfusion/ej2-vue-
layouts/styles/material.css';
  #container {
    visibility: hidden;
  }
  #loader {
    color: #008cff;
    height: 40px;
    left: 45%;
    position: absolute;
    top: 45%;
    width: 30%;
  }
  .e-card-image {
    background: url('../sample.jpg');
    height: 164px;
  }
  .e-card {
    width: 200px;
    margin: auto;
  }
  .topleft {
    top: 0;
    bottom: auto !important;
  }
  .topright {
    top: 0;
    text-align: right;
    bottom: auto !important;
  }
  .bottomright {
    bottom: 0;
    right: 0;
    text-align: right;
  }
</style>

```

{% previewsample "page.domainurl/code-snippet/card/how-to/card\_image-cs1" %}

### Integrate other component inside the card in Vue Card component

You can integrate any component inside the card element. Here ListView component is placed inside the card for showcasing the To-Do list.

#### APP.VUE

```
<template>
  <div id="app">
    <div style="margin: 50px;">
      <!--element which is going to render the Card-->
      <div tabindex="0" class="e-card" id="basic">
        <div class="e-card-title">To-Do List</div>
        <div class="e-card-separator"></div>
        <div class="e-card-content">
          <div id='element'></div>
        </div>
      </div>
    </div>
  </div>
</template>
<script>
import Vue from 'vue';
import { ListView } from '@syncfusion/ej2-lists';
export default {
  name: 'app',
  data() {
  }, mounted() {
    //Initialize ListView component
    var listViewInstance: ListView = new ListView({
      dataSource: [
        { todoList: 'Pay Bills' },
        { todoList: 'Call Chris' },
        { todoList: 'Meet Andrew' },
        { todoList: 'Visit Manager' },
        { todoList: 'Customer Meeting' },
      ],
      //map the appropriate columns to fields property
      fields: { text: 'todoList' },
    }, "#element");
  }
}
</script>
<style>
  @import '../..../node_modules/@syncfusion/ej2-base/styles/material.css';
  @import '../..../node_modules/@syncfusion/ej2-vue-
layouts/styles/material.css';
  #container {
    visibility: hidden;
  }
  #loader {
    color: #008cff;
    height: 40px;
    left: 45%;
    position: absolute;
    top: 45%;
    width: 30%;
  }
</style>
```

```
}  
</style>
```

```
{% previewsample "page.domainurl/code-snippet/card/how-to/card_component-cs1" %}
```

## Carousel

### Getting Started with the Vue Carousel Component in Vue 2

This article provides a step-by-step guide for setting up a Vue 2 project using [Vue-CLI](#) and integrating the Syncfusion [Vue Carousel](#) component.

#### Prerequisites

[System requirements for Syncfusion Vue UI components](#)

#### Dependencies

The following list of dependencies are required to use the Carousel component in your application.

```
`js  
|-- @syncfusion/ej2-vue-navigations  
|-- @syncfusion/ej2-vue-base  
|-- @syncfusion/ej2-navigations  
|-- @syncfusion/ej2-base  
|-- @syncfusion/ej2-buttons  
`
```

#### Setting up the Vue 2 project

To generate a Vue 2 project using Vue-CLI, use the [vue create](#) command. Follow these steps to install Vue CLI and create a new project:

```
`bash  
npm install -g @vue/cli  
vue create quickstart  
cd quickstart  
npm run serve  
`
```

or

```
`bash  
yarn global add @vue/cli  
vue create quickstart  
cd quickstart  
yarn run serve  
`
```

When creating a new project, choose the option `Default ([Vue 2] babel, eslint)` from the menu.

```
E:\samples>vue create quickstart

Vue CLI v5.0.8
? Please pick a preset:
  Default ([Vue 3] babel, eslint)
> Default ([Vue 2] babel, eslint)
  Manually select features
```

Once the `quickstart` project is set up with default settings, proceed to add Syncfusion components to the project.

#### Add Syncfusion Vue packages

Syncfusion packages are available at [npmjs.com](https://www.npmjs.com). To use Vue components, install the required npm package.

This article uses the [Vue Carousel component](#) as an example. Install the `@syncfusion/ej2-vue-navigations` package by running the following command:

```
`bash
```

```
npm install @syncfusion/ej2-vue-navigations --save
```

```
,
```

or

```
`bash
```

```
yarn add @syncfusion/ej2-vue-navigations
```

```
,
```

#### Add Syncfusion Vue component

Follow the below steps to add the Vue Carousel component:

1\ First, import and register the Carousel component in the `script` section of the `src/App.vue` file.

##### ~/SRC/APP.VUE

```
<script>
import { CarouselComponent, CarouselItemDirective, CarouselItemsDirective }
  from "@syncfusion/ej2-vue-navigations";
export default {
  components: {
    'ejs-carousel': CarouselComponent,
    'e-carousel-item': CarouselItemDirective,
    'e-carousel-items': CarouselItemsDirective
  }
}
</script>
```

2\ In the `template` section, define the Carousel component.

~/SRC/APP.VUE

```

<template>
<div class="control-container">
<ejs-carousel>
<e-carousel-items>
<e-carousel-item template="<figure class='img-container'><img
src='https://ej2.syncfusion.com/products/images/carousel/cardinal.png'
alt='cardinal' style='height:100%;width:100%;' /><figcaption class='img-
caption'>Cardinal</figcaption></figure>"></e-carousel-item>
<e-carousel-item template="<figure class='img-container'><img
src='https://ej2.syncfusion.com/products/images/carousel/hunei.png'
alt='kingfisher' style='height:100%;width:100%;' /><figcaption class='img-
caption'>Kingfisher</figcaption></figure>"></e-carousel-item>
<e-carousel-item template="<figure class='img-container'><img
src='https://ej2.syncfusion.com/products/images/carousel/costa-rica.png'
alt='keel-billed-toucan' style='height:100%;width:100%;' /><figcaption
class='img-caption'>Keel-billed-toucan</figcaption></figure>"></e-carousel-
item>
<e-carousel-item template="<figure class='img-container'><img
src='https://ej2.syncfusion.com/products/images/carousel/kaohsiung.png'
alt='yellow-warbler' style='height:100%;width:100%;' /><figcaption
class='img-caption'>Yellow-warbler</figcaption></figure>"></e-carousel-item>
<e-carousel-item template="<figure class='img-container'><img
src='https://ej2.syncfusion.com/products/images/carousel/bee-eater.png'
alt='bee-eater' style='height:100%;width:100%;' /><figcaption class='img-
caption'>Bee-eater</figcaption></figure>"></e-carousel-item>
</e-carousel-items>
</ejs-carousel>
</div>
</template>

```

Here is the summarized code for the above steps in the **src/App.vue** file:

~/SRC/APP.VUE

```

<template>
  <div class="control-container">
    <ejs-carousel>
      <e-carousel-items>
        <e-carousel-item template="Cardinal"></e-carousel-item>
        <template v-slot:Cardinal>
          <figure class="img-container">
            
            <figcaption class="img-caption">Cardinal</figcaption>
          </figure>
        </template>
        <e-carousel-item template="kingfisher"></e-carousel-item>
        <template v-slot:kingfisher>
          <figure class="img-container">
            

```



```

        <figcaption class="img-caption">kingfisher</figcaption>
      </figure>
    </template>
    <e-carousel-item template="keel-billed-toucan"></e-carousel-item>
    <template v-slot:keel-billed-toucan>
      <figure class="img-container">
        
        <figcaption class="img-caption">keel-billed-toucan</figcaption>
      </figure>
    </template>
    <e-carousel-item template="yellow-warbler"></e-carousel-item>
    <template v-slot:yellow-warbler>
      <figure class="img-container">
        
        <figcaption class="img-caption">yellow-warbler</figcaption>
      </figure>
    </template>
    <e-carousel-item template="bee-eater"></e-carousel-item>
    <template v-slot:bee-eater>
      <figure class="img-container">
        
        <figcaption class="img-caption">bee-eater</figcaption>
      </figure>
    </template>
  </e-carousel-items>
</ejs-carousel>
</div>
</template>
<script>
  import { CarouselComponent, CarouselItemDirective, CarouselItemsDirective
} from "@syncfusion/ej2-vue-navigations";
  export default {
    components: {
      'ejs-carousel': CarouselComponent,
      'e-carousel-item': CarouselItemDirective,
      'e-carousel-items': CarouselItemsDirective
    },
    data: function () {
      return {};
    },
  };
</script>
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-buttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-navigations/styles/material.css";
.control-container {
  height: 360px;

```

```
margin: 0 auto;
width: 600px;
}
.img-container {
  height: 100%;
  margin: 0;
}
.img-caption {
  color: #fff;
  font-size: 1rem;
  position: absolute;
  bottom: 3rem;
  width: 100%;
  text-align: center;
}
</style>
```

### Run the project

To run the project, use the following command:

```
`bash
```

```
npm run serve
```

```
,
```

or

```
`bash
```

```
yarn run serve
```

```
,
```

```
{% previewsample "page.domainurl/code-snippet/carousel/getting-started-cs1" %}
```

**Note:** You can also explore our [Vue Carousel example](#) that shows you how to render the Carousel in Vue.

### Getting Started with Syncfusion Vue Carousel Component in Vue 3

This article provides a step-by-step guide for setting up a [Vite](#) project with a JavaScript environment and integrating the Syncfusion Vue Carousel component using the [Composition API](#) / [Options API](#).

The **Composition API** is a new feature introduced in Vue.js 3 that provides an alternative way to organize and reuse component logic. It allows developers to write components as functions that use smaller, reusable functions called composition functions to manage their properties and behavior.

The **Options API** is the traditional way of writing Vue.js components, where the component logic is organized into a series of options that define the component's properties and behavior. These options include data, methods, computed properties, watchers, lifecycle hooks, and more.

#### Prerequisites

##### [System requirements for Syncfusion Vue UI components](#)

#### Setup the Vite project

A recommended approach for beginning with Vue is to scaffold a project using [Vite](#). To create a new Vite project, use one of the commands that are specific to either NPM or Yarn.

```
`bash
npm create vite@latest
`
```

or

```
`bash
yarn create vite
`
```

Using one of the above commands will lead you to set up additional configurations for the project as below:

1. Define the project name: We can specify the name of the project directly. Let's specify the name of the project as `my-project` for this article.

```
`bash
? Project name: » my-project
`
```

2. Select `Vue` as the framework. It will create a Vue 3 project.

```
`bash
? Select a framework: » - Use arrow-keys. Return to submit.
```

Vanilla

Vue

React

Preact

Lit

Svelte

Others

`

3. Choose `JavaScript` as framework variant to build this Vite project using JavaScript and Vue.

```
`bash
? Select a variant: » - Use arrow-keys. Return to submit.
```

JavaScript

TypeScript

Customize with create-vue ↗

Nuxt ↗

`

4. Upon completing the aforementioned steps to create the `my-project`, run the following command to install its dependencies:

```
`bash
cd my-project
npm install
`
or
`bash
cd my-project
yarn install
`
```

Now that `my-project` is ready to run with default settings, let's add Syncfusion Vue Carousel component to the project.

#### Adding Syncfusion Vue packages

Syncfusion Vue component packages are available at [npmjs.com](https://www.npmjs.com). To use Syncfusion Vue components in the project, install the corresponding npm package.

This article uses the [Vue Carousel component](#) as an example. To use the `Vue Carousel` component in the project, the `@syncfusion/ej2-vue-navigations` package needs to be installed using the following command.

```
`bash
npm install @syncfusion/ej2-vue-navigations --save
`
or
`bash
yarn add @syncfusion/ej2-vue-navigations
`
```

#### Import Syncfusion CSS styles

You can import themes for the Syncfusion Vue component in various ways, such as using CSS or SASS styles from npm packages, CDN, [CRG](#) and [Theme Studio](#). Refer to [themes topic](#) to know more about built-in themes and different ways to refer to themes in a Vue project.

In this article, `Material` theme is applied using CSS styles, which are available in installed packages. The necessary `Material` CSS styles for the Carousel component and its dependents were imported into the `<style>` section of the `src/App.vue` file.

```
`html
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
```

```
@import "../node_modules/@syncfusion/ej2-buttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-navigations/styles/material.css";
</style>
`
```

### Adding Syncfusion Vue Carousel component in the application

Follow the below steps to add the Vue Carousel component using **Composition API** or **Options API**:

1.First, import and register the Carousel component and its child directives in the **script** section of the **src/App.vue** file. If you are using the **Composition API**, you should add the **setup** attribute to the **script** tag to indicate that Vue will be using the **Composition API**.

#### **COMPOSITION API (~SRC/APP.VUE)**

```
<script setup>
import {
  CarouselComponent as EjsCarousel, CarouselItemsDirective as ECarouselItems,
  CarouselItemDirective as ECarouselItem
} from '@syncfusion/ej2-vue-navigations';
</script>
```

#### **OPTIONS API (~SRC/APP.VUE)**

```
<script>
import {
  CarouselComponent,
  CarouselItemDirective,
  CarouselItemsDirective,
} from '@syncfusion/ej2-vue-navigations';
</script>
```

2.Add the component definition in template section.

```
`html
<template>
<div class="control-container">
<ejs-carousel>
<e-carousel-items>
<e-carousel-item
  template="<figure class='img-container'><img
src='https://ej2.syncfusion.com/products/images/carousel/cardinal.png' alt='cardinal'
style='height:100%;width:100%;' /><figcaption class='img-caption'>Cardinal</figcaption></figure>"></e-
carousel-item>
<e-carousel-item
  template="<figure class='img-container'><img
src='https://ej2.syncfusion.com/products/images/carousel/hunei.png' alt='kingfisher'
```

```

style='height:100%;width:100%;' /><figcaption class='img-
caption'>Kingfisher</figcaption></figure>"></e-carousel-item>

<e-carousel-item

template="<figure class='img-container'><img
src='https://ej2.syncfusion.com/products/images/carousel/costa-rica.png' alt='keel-billed-toucan'
style='height:100%;width:100%;' /><figcaption class='img-caption'>Keel-billed-
toucan</figcaption></figure>"></e-carousel-item>

<e-carousel-item

template="<figure class='img-container'><img
src='https://ej2.syncfusion.com/products/images/carousel/kaohsiung.png' alt='yellow-warbler'
style='height:100%;width:100%;' /><figcaption class='img-caption'>Yellow-
warbler</figcaption></figure>"></e-carousel-item>

<e-carousel-item

template="<figure class='img-container'><img
src='https://ej2.syncfusion.com/products/images/carousel/bee-eater.png' alt='bee-eater'
style='height:100%;width:100%;' /><figcaption class='img-caption'>Bee-
eater</figcaption></figure>"></e-carousel-item>

</e-carousel-items>

</ejs-carousel>

</div>

</template>

```

Here is the summarized code for the above steps in the **src/App.vue** file:

#### **COMPOSITION API (~SRC/APP.VUE)**

```

<template>
<div class="control-container">
<ejs-carousel>
<e-carousel-items>
<e-carousel-item
template="<figure class='img-container'><img
src='https://ej2.syncfusion.com/products/images/carousel/cardinal.png'
alt='cardinal' style='height:100%;width:100%;' /><figcaption class='img-
caption'>Cardinal</figcaption></figure>"></e-carousel-item>
<e-carousel-item
template="<figure class='img-container'><img
src='https://ej2.syncfusion.com/products/images/carousel/hunei.png'
alt='kingfisher' style='height:100%;width:100%;' /><figcaption class='img-
caption'>Kingfisher</figcaption></figure>"></e-carousel-item>
<e-carousel-item
template="<figure class='img-container'><img
src='https://ej2.syncfusion.com/products/images/carousel/costa-rica.png'
alt='keel-billed-toucan' style='height:100%;width:100%;' /><figcaption
class='img-caption'>Keel-billed-toucan</figcaption></figure>"></e-carousel-
item>
<e-carousel-item

```

```

template=<"<figure class='img-container'><img
src='https://ej2.syncfusion.com/products/images/carousel/kaohsiung.png'
alt='yellow-warbler' style='height:100%;width:100%;' /><figcaption
class='img-caption'>Yellow-warbler</figcaption></figure>"></e-carousel-item>
<e-carousel-item
template=<"<figure class='img-container'><img
src='https://ej2.syncfusion.com/products/images/carousel/bee-eater.png'
alt='bee-eater' style='height:100%;width:100%;' /><figcaption class='img-
caption'>Bee-eater</figcaption></figure>"></e-carousel-item>
</e-carousel-items>
</ejs-carousel>
</div>
</template>
<script setup>
import {
CarouselComponent as EjsCarousel, CarouselItemsDirective as ECarouselItems,
CarouselItemDirective as ECarouselItem
} from '@syncfusion/ej2-vue-navigations';
</script>
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-buttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-
navigations/styles/material.css";
.control-container {
height: 360px;
margin: 0 auto;
width: 600px;
}
.img-container {
height: 100%;
margin: 0;
}
.img-caption {
color: #fff;
font-size: 1rem;
position: absolute;
bottom: 3rem;
width: 100%;
text-align: center;
}
</style>

```

### **OPTIONS API (~SRC/APP.VUE)**

```

<template>
<div class="control-container">
<ejs-carousel>
<e-carousel-items>
<e-carousel-item
template=<"<figure class='img-container'><img
src='https://ej2.syncfusion.com/products/images/carousel/cardinal.png'
alt='cardinal' style='height:100%;width:100%;' /><figcaption class='img-
caption'>Cardinal</figcaption></figure>"></e-carousel-item>
<e-carousel-item

```

```

template=<figure class='img-container'><img
src='https://ej2.syncfusion.com/products/images/carousel/hunei.png'
alt='kingfisher' style='height:100%;width:100%;' /><figcaption class='img-
caption'>Kingfisher</figcaption></figure>"></e-carousel-item>
<e-carousel-item
template=<figure class='img-container'><img
src='https://ej2.syncfusion.com/products/images/carousel/costa-rica.png'
alt='keel-billed-toucan' style='height:100%;width:100%;' /><figcaption
class='img-caption'>Keel-billed-toucan</figcaption></figure>"></e-carousel-
item>
<e-carousel-item
template=<figure class='img-container'><img
src='https://ej2.syncfusion.com/products/images/carousel/kaohsiung.png'
alt='yellow-warbler' style='height:100%;width:100%;' /><figcaption
class='img-caption'>Yellow-warbler</figcaption></figure>"></e-carousel-item>
<e-carousel-item
template=<figure class='img-container'><img
src='https://ej2.syncfusion.com/products/images/carousel/bee-eater.png'
alt='bee-eater' style='height:100%;width:100%;' /><figcaption class='img-
caption'>Bee-eater</figcaption></figure>"></e-carousel-item>
</e-carousel-items>
</ejs-carousel>
</div>
</template>
<script>
import {
CarouselComponent,
CarouselItemDirective,
CarouselItemsDirective,
} from "@syncfusion/ej2-vue-navigations";
export default {
name: "App",
components: {
"ejs-carousel": CarouselComponent,
"e-carousel-items": CarouselItemsDirective,
"e-carousel-item": CarouselItemDirective
}}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-buttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-
navigations/styles/material.css";
.control-container {
height: 360px;
margin: 0 auto;
width: 600px;
}
.img-container {
height: 100%;
margin: 0;
}
.img-caption {
color: #fff;
font-size: 1rem;
position: absolute;
bottom: 3rem;

```



```
width: 100%;  
text-align: center;  
}  
</style>
```

### Run the project

To run the project, use the following command:

```
`bash
```

```
npm run dev
```

```
,
```

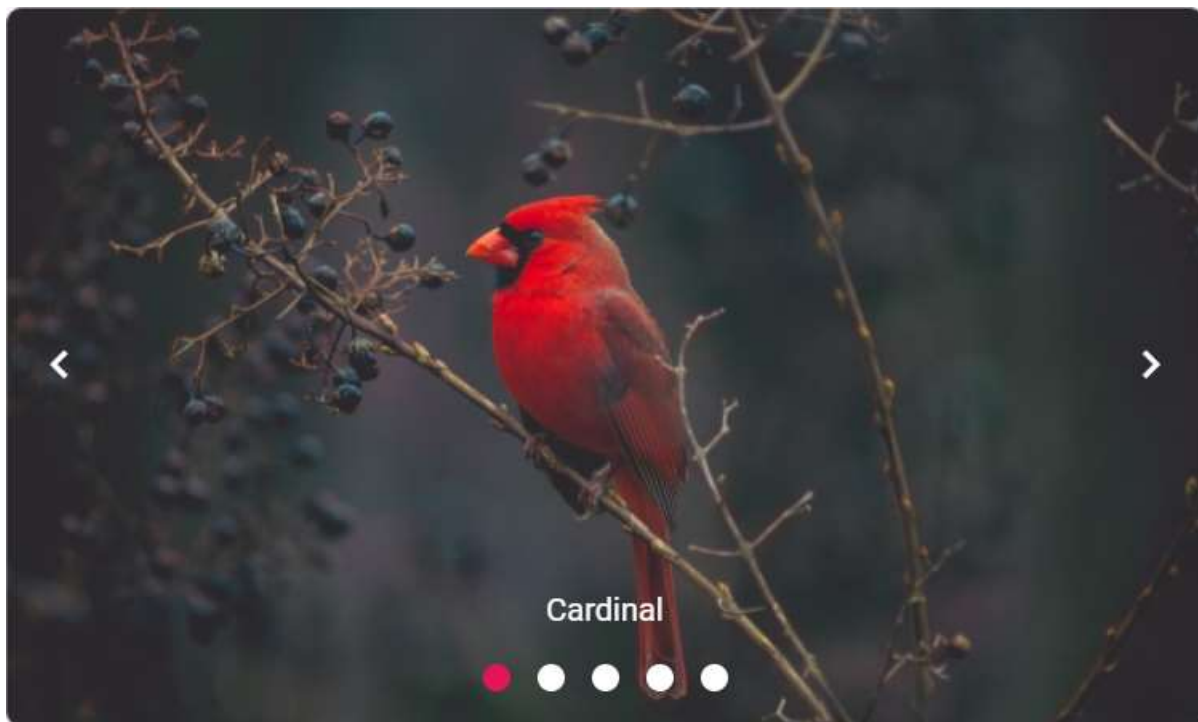
or

```
`bash
```

```
yarn run dev
```

```
,
```

The output will appear as follows:



Refer the sample [Vue 3 using Composition API Carousel getting started](#)

For migrating from Vue 2 to Vue 3, refer to the [migration](#) documentation.

See also

- [Getting Started with Vue UI Components using Composition API and TypeScript](#)
- [Getting Started with Vue UI Components using Options API and TypeScript](#)

## Populating items in Vue Carousel component

In the Carousel, slides can be rendered in two ways as follows,

- Populating items using carousel item
- Populating items using data source

### Populating items using carousel item

When rendering the Carousel component using items binding, you can assign templates for each item separately or assign a common template to each item. You can also customize the slide transition interval for each item separately. The following example code depicts the functionality as item property binding.

#### APP.VUE

```
<template>
  <div class="control-container">
    <ejs-carousel>
      <e-carousel-items>
        <e-carousel-item template="Cardinal"></e-carousel-item>
        <template v-slot:Cardinal>
          <figure class="img-container">
            
            <figcaption class="img-caption">Cardinal</figcaption>
          </figure>
        </template>
        <e-carousel-item template="kingfisher"></e-carousel-item>
        <template v-slot:kingfisher>
          <figure class="img-container">
            
            <figcaption class="img-caption">kingfisher</figcaption>
          </figure>
        </template>
        <e-carousel-item template="keel-billed-toucan"></e-carousel-item>
        <template v-slot:keel-billed-toucan>
          <figure class="img-container">
            
            <figcaption class="img-caption">keel-billed-toucan</figcaption>
          </figure>
        </template>
        <e-carousel-item template="yellow-warbler"></e-carousel-item>
        <template v-slot:yellow-warbler>
          <figure class="img-container">
            

```

```

        <figcaption class="img-caption">yellow-warbler</figcaption>
      </figure>
    </template>
    <e-carousel-item template="bee-eater"></e-carousel-item>
    <template v-slot:bee-eater>
      <figure class="img-container">
        
        <figcaption class="img-caption">bee-eater</figcaption>
      </figure>
    </template>
  </e-carousel-items>
</ejs-carousel>
</div>
</template>
<script>
import Vue from "vue";
import { CarouselPlugin } from "@syncfusion/ej2-vue-navigations";
Vue.use(CarouselPlugin);
export default {};
</script>
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-buttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-navigations/styles/material.css";
.control-container {
  height: 360px;
  margin: 0 auto;
  width: 600px;
}
.img-container {
  height: 100%;
  margin: 0;
}
.img-caption {
  color: #fff;
  font-size: 1rem;
  position: absolute;
  bottom: 3rem;
  width: 100%;
  text-align: center;
}
</style>

```

{% previewsample "page.domainurl/code-snippet/carousel/default-cs17" %}

### Populating items using data source

When rendering the Carousel component using data binding, you can assign a common template only for all items using the [itemTemplate](#) property. You cannot set the interval for each item. The following example code depicts the functionality as data binding.

### APP.VUE

```
<template>
```

```

<div class="control-container">
  <ejs-carousel :dataSource="productItems"
:itemTemplate="productTemplate"></ejs-carousel>
</div>
</template>
<script>
import Vue from 'vue';
import { CarouselPlugin } from "@syncfusion/ej2-vue-navigations";
Vue.use(CarouselPlugin);
var productVue = Vue.component("product", {
  template: `<figure class="img-container">
  <figcaption class="img-caption">{{data.Name}}</figcaption></figure>`,
  data() {
    return {
      data: {},
    };
  },
  computed: {},
  methods: {
    getImage: function (bird) {
      return "https://ej2.syncfusion.com/products/images/carousel/" + bird
+ ".png";
    },
    getAlt: function (altText) {
      return altText;
    }
  },
});
export default {
  data: function() {
    return {
      productItems: [
        { ID: 1, Name: "Cardinal", imageName: 'cardinal' },
        { ID: 2, Name: "Kingfisher", imageName: 'hunei' },
        { ID: 3, Name: "Keel-billed-toucan", imageName: 'costa-rica' },
        { ID: 4, Name: "Yellow-warbler", imageName: 'kaohsiung' },
        { ID: 5, Name: "Bee-eater", imageName: 'bee-eater' }
      ],
      productTemplate: function (e) {
        return {
          template: productVue
        };
      },
    };
  }
}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-buttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-navigations/styles/material.css";
.control-container {
  height: 360px;
  margin: 0 auto;
  width: 600px;
}

```

```

}
.img-container {
  height: 100%;
  margin: 0;
}
.img-caption {
  color: #fff;
  font-size: 1rem;
  position: absolute;
  bottom: 3rem;
  width: 100%;
  text-align: center;
}
</style>

```

{% previewsample "page.domainurl/code-snippet/carousel/default-cs18" %}

### Selection

The Carousel items will be populated from the first index of the Carousel items and can be customized using the following ways,

- Select an item using the property.
- Select an item using the method.

#### Select an item using the property

Using the [selectedIndex](#) property of the Carousel component, you can set the slide to be populated at the time of initial rendering else you can switch to the particular slide item.

### APP.VUE

```

<template>
  <div class="control-container">
    <ejs-carousel :selectedIndex="selectedIndex">
      <e-carousel-items>
        <e-carousel-item template="Cardinal"></e-carousel-item>
        <template v-slot:Cardinal>
          <figure class="img-container">
            
            <figcaption class="img-caption">Cardinal</figcaption>
          </figure>
        </template>
        <e-carousel-item template="kingfisher"></e-carousel-item>
        <template v-slot:kingfisher>
          <figure class="img-container">
            
            <figcaption class="img-caption">kingfisher</figcaption>
          </figure>
        </template>
        <e-carousel-item template="keel-billed-toucan"></e-carousel-item>
      </e-carousel-items>
    </ejs-carousel>
  </div>
</template>

```

```

    <template v-slot:keel-billed-toucan>
      <figure class="img-container">
        
        <figcaption class="img-caption">keel-billed-toucan</figcaption>
      </figure>
    </template>
    <e-carousel-item template="yellow-warbler"></e-carousel-item>
    <template v-slot:yellow-warbler>
      <figure class="img-container">
        
        <figcaption class="img-caption">yellow-warbler</figcaption>
      </figure>
    </template>
    <e-carousel-item template="bee-eater"></e-carousel-item>
    <template v-slot:bee-eater>
      <figure class="img-container">
        
        <figcaption class="img-caption">bee-eater</figcaption>
      </figure>
    </template>
  </e-carousel-items>
</ejs-carousel>
</div>
</template>
<script>
import Vue from "vue";
import { CarouselPlugin } from "@syncfusion/ej2-vue-navigations";
Vue.use(CarouselPlugin);
export default {
  data: function () {
    return {
      selectedIndex: 3,
    };
  },
};
</script>
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-buttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-navigations/styles/material.css";
.control-container {
  height: 360px;
  margin: 0 auto;
  width: 600px;
}
.img-container {
  height: 100%;
  margin: 0;

```

```

    }
    .img-caption {
      color: #fff;
      font-size: 1rem;
      position: absolute;
      bottom: 3rem;
      width: 100%;
      text-align: center;
    }
  </style>

```

{% previewsample "page.domainurl/code-snippet/carousel/default-cs19" %}

*Select an item using the method*

Using the [prev](#) or [next](#) public method of the Carousel component, you can switch the current populating slide to a previous or next slide.

### **APP.VUE**

```

<template>
  <div>
    <ejs-button cssClass="e-info" v-
on:click.native="prevBtnClick">Previous</ejs-button>
    <ejs-button cssClass="e-info" v-
on:click.native="nextBtnClick">Next</ejs-button>
    <div class="control-container">
      <ejs-carousel>
        <e-carousel-items>
          <e-carousel-item template="Cardinal"></e-carousel-item>
          <template v-slot:Cardinal>
            <figure class="img-container">
              
              <figcaption class="img-caption">Cardinal</figcaption>
            </figure>
          </template>
          <e-carousel-item template="kingfisher"></e-carousel-item>
          <template v-slot:kingfisher>
            <figure class="img-container">
              
              <figcaption class="img-caption">kingfisher</figcaption>
            </figure>
          </template>
          <e-carousel-item template="keel-billed-toucan"></e-carousel-item>
          <template v-slot:keel-billed-toucan>
            <figure class="img-container">
              
              <figcaption class="img-caption">keel-billed-toucan</figcaption>
            </figure>
          </template>
        </e-carousel-items>
      </ejs-carousel>
    </div>
  </div>
</template>

```

```

        </figure>
      </template>
      <e-carousel-item template="yellow-warbler"></e-carousel-item>
      <template v-slot:yellow-warbler>
        <figure class="img-container">
          
          <figcaption class="img-caption">yellow-warbler</figcaption>
        </figure>
      </template>
      <e-carousel-item template="bee-eater"></e-carousel-item>
      <template v-slot:bee-eater>
        <figure class="img-container">
          
          <figcaption class="img-caption">bee-eater</figcaption>
        </figure>
      </template>
    </e-carousel-items>
  </ejs-carousel>
</div>
</div>
</template>
<script>
import Vue from "vue";
import { CarouselPlugin } from "@syncfusion/ej2-vue-navigations";
import { ButtonPlugin } from "@syncfusion/ej2-vue-buttons";
Vue.use(ButtonPlugin);
Vue.use(CarouselPlugin);
export default {
  methods: {
    prevBtnClick: function (event) {
      var carouselObj = document.querySelector(".e-
carousel").ej2_instances[0];
      carouselObj.prev();
    },
    nextBtnClick: function (event) {
      var carouselObj = document.querySelector(".e-
carousel").ej2_instances[0];
      carouselObj.next();
    },
  },
};
</script>
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-buttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-navigations/styles/material.css";
.control-container {
  height: 360px;
  margin: 0 auto;
  width: 600px;
}

```



```

.img-container {
  height: 100%;
  margin: 0;
}
.img-caption {
  color: #fff;
  font-size: 1rem;
  position: absolute;
  bottom: 3rem;
  width: 100%;
  text-align: center;
}
</style>

```

{% previewsample "page.domainurl/code-snippet/carousel/default-cs20" %}

### Partial visible slides

The Carousel component supports to show one complete slide and a partial view of adjacent (previous and next) slides at the same time. You can enable or disable the partial slides using the [partialVisible](#) property.

### APP.VUE

```

<template>
  <div class="control-container">
    <ejs-carousel :partialVisible="true">
      <e-carousel-items>
        <e-carousel-item template="Cardinal"></e-carousel-item>
        <template v-slot:Cardinal>
          <figure class="img-container">
            
            <figcaption class="img-caption">Cardinal</figcaption>
          </figure>
        </template>
        <e-carousel-item template="kingfisher"></e-carousel-item>
        <template v-slot:kingfisher>
          <figure class="img-container">
            
            <figcaption class="img-caption">kingfisher</figcaption>
          </figure>
        </template>
        <e-carousel-item template="keel-billed-toucan"></e-carousel-item>
        <template v-slot:keel-billed-toucan>
          <figure class="img-container">
            
            <figcaption class="img-caption">keel-billed-toucan</figcaption>
          </figure>

```

```

    </template>
    <e-carousel-item template="yellow-warbler"></e-carousel-item>
    <template v-slot:yellow-warbler>
      <figure class="img-container">
        
        <figcaption class="img-caption">yellow-warbler</figcaption>
      </figure>
    </template>
    <e-carousel-item template="bee-eater"></e-carousel-item>
    <template v-slot:bee-eater>
      <figure class="img-container">
        
        <figcaption class="img-caption">bee-eater</figcaption>
      </figure>
    </template>
  </e-carousel-items>
</ejs-carousel>
</div>
</template>
<script>
import Vue from "vue";
import { CarouselPlugin } from "@syncfusion/ej2-vue-navigations";
Vue.use(CarouselPlugin);
export default {};
</script>
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-buttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-navigations/styles/material.css";
.control-container {
  margin: 0 auto 2em;
  max-width: 800px;
  height: 300px;
}
.img-container {
  margin: 0 10px;
  width: 100%;
  height: 100%;
}
.img-caption {
  bottom: 4em;
  color: #fff;
  font-size: 12pt;
  height: 2em;
  position: relative;
  padding: 0.3em 1em;
  text-align: center;
  width: 100%;
}
</style>

```

{% previewsample "page.domainurl/code-snippet/carousel/default-cs21" %}

Slide animation only applicable if the `partialVisible` is enabled.

The last slide will be displayed as a partial slide at the initial rendering when the `loop` and `partialVisible` properties are enabled.

The previous slide is not displayed at the initial rendering when the `loop` is disabled.

The following example code depicts the functionality of `partialVisible` and without `loop` functionalities.

#### APP.VUE

```
<template>
  <div class="control-container">
    <ejs-carousel :partialVisible="true" :loop="false">
      <e-carousel-items>
        <e-carousel-item template="Cardinal"></e-carousel-item>
        <template v-slot:Cardinal>
          <figure class="img-container">
            
            <figcaption class="img-caption">Cardinal</figcaption>
          </figure>
        </template>
        <e-carousel-item template="kingfisher"></e-carousel-item>
        <template v-slot:kingfisher>
          <figure class="img-container">
            
            <figcaption class="img-caption">kingfisher</figcaption>
          </figure>
        </template>
        <e-carousel-item template="keel-billed-toucan"></e-carousel-item>
        <template v-slot:keel-billed-toucan>
          <figure class="img-container">
            
            <figcaption class="img-caption">keel-billed-toucan</figcaption>
          </figure>
        </template>
        <e-carousel-item template="yellow-warbler"></e-carousel-item>
        <template v-slot:yellow-warbler>
          <figure class="img-container">
            
            <figcaption class="img-caption">yellow-warbler</figcaption>
          </figure>
        </template>
      </e-carousel-items>
    </ejs-carousel>
  </div>
</template>
```

```

    </template>
    <e-carousel-item template="bee-eater"></e-carousel-item>
    <template v-slot:bee-eater>
      <figure class="img-container">
        
        <figcaption class="img-caption">bee-eater</figcaption>
      </figure>
    </template>
  </e-carousel-items>
</ejs-carousel>
</div>
</template>
<script>
import Vue from "vue";
import { CarouselPlugin } from "@syncfusion/ej2-vue-navigations";
Vue.use(CarouselPlugin);
export default {};
</script>
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-buttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-navigations/styles/material.css";
.control-container {
  margin: 0 auto 2em;
  max-width: 800px;
  height: 300px;
}
.img-container {
  margin: 0 10px;
  width: 100%;
  height: 100%;
}
.img-caption {
  bottom: 4em;
  color: #fff;
  font-size: 12pt;
  height: 2em;
  position: relative;
  padding: 0.3em 1em;
  text-align: center;
  width: 100%;
}
</style>

```

{% previewsample "page.domainurl/code-snippet/carousel/default-cs22" %}

See Also

- [Customizing partial slides area size](#)

## Navigators and indicators in Vue Carousel component

The navigators and indicators are used to transition the slides manually.

## Navigators

### Show or hide previous and next button

In navigators, the previous and next slide transition buttons are used to perform slide transitions manually. You can show/hide the navigators using the [buttonsVisibility](#) property. The possible property values are as follows:

- **Hidden** – the navigator's buttons are not visible.
- **Visible** – the navigator's buttons are visible.
- **VisibleOnHover** – the navigator's buttons are visible only when hovering over the carousel.

The following example depicts the code to show/hide the navigators in the carousel.

### APP.VUE

```
<template>
  <div class="control-container">
    <ejs-carousel buttonsVisibility="Visible">
      <e-carousel-items>
        <e-carousel-item template="Cardinal"></e-carousel-item>
        <template v-slot:Cardinal>
          <figure class="img-container">
            
            <figcaption class="img-caption">Cardinal</figcaption>
          </figure>
        </template>
        <e-carousel-item template="kingfisher"></e-carousel-item>
        <template v-slot:kingfisher>
          <figure class="img-container">
            
            <figcaption class="img-caption">kingfisher</figcaption>
          </figure>
        </template>
        <e-carousel-item template="keel-billed-toucan"></e-carousel-item>
        <template v-slot:keel-billed-toucan>
          <figure class="img-container">
            
            <figcaption class="img-caption">keel-billed-toucan</figcaption>
          </figure>
        </template>
        <e-carousel-item template="yellow-warbler"></e-carousel-item>
        <template v-slot:yellow-warbler>
          <figure class="img-container">
            
          </figure>
        </template>
      </e-carousel-items>
    </ejs-carousel>
  </div>
</template>
```

```

        <figcaption class="img-caption">yellow-warbler</figcaption>
      </figure>
    </template>
    <e-carousel-item template="bee-eater"></e-carousel-item>
    <template v-slot:bee-eater>
      <figure class="img-container">
        
        <figcaption class="img-caption">bee-eater</figcaption>
      </figure>
    </template>
  </e-carousel-items>
</ejs-carousel>
</div>
</template>
<script>
import Vue from "vue";
import { CarouselPlugin } from "@syncfusion/ej2-vue-navigations";
Vue.use(CarouselPlugin);
export default {};
</script>
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-buttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-navigations/styles/material.css";
.control-container {
  height: 360px;
  margin: 0 auto;
  width: 600px;
}
.img-container {
  height: 100%;
  margin: 0;
}
.img-caption {
  color: #fff;
  font-size: 1rem;
  position: absolute;
  bottom: 3rem;
  width: 100%;
  text-align: center;
}
</style>

```

{% previewsample "page.domainurl/code-snippet/carousel/default-cs9" %}

*Show previous and next button on hover*

In the carousel, you can show the previous and next buttons only on mouse hover using the [buttonsVisibility](#) property. The following example depicts the code to show the navigators on mouse hover in the carousel.

#### APP.VUE

```
<template>
```

```

<div class="control-container">
  <ejs-carousel buttonsVisibility="VisibleOnHover">
    <e-carousel-items>
      <e-carousel-item template="Cardinal"></e-carousel-item>
      <template v-slot:Cardinal>
        <figure class="img-container">
          
          <figcaption class="img-caption">Cardinal</figcaption>
        </figure>
      </template>
      <e-carousel-item template="kingfisher"></e-carousel-item>
      <template v-slot:kingfisher>
        <figure class="img-container">
          
          <figcaption class="img-caption">kingfisher</figcaption>
        </figure>
      </template>
      <e-carousel-item template="keel-billed-toucan"></e-carousel-item>
      <template v-slot:keel-billed-toucan>
        <figure class="img-container">
          
          <figcaption class="img-caption">keel-billed-toucan</figcaption>
        </figure>
      </template>
      <e-carousel-item template="yellow-warbler"></e-carousel-item>
      <template v-slot:yellow-warbler>
        <figure class="img-container">
          
          <figcaption class="img-caption">yellow-warbler</figcaption>
        </figure>
      </template>
      <e-carousel-item template="bee-eater"></e-carousel-item>
      <template v-slot:bee-eater>
        <figure class="img-container">
          
          <figcaption class="img-caption">bee-eater</figcaption>
        </figure>
      </template>
    </e-carousel-items>
  </ejs-carousel>
</div>
</template>

```

```

<script>
  import Vue from "vue";
  import { CarouselPlugin } from "@syncfusion/ej2-vue-navigations";
  Vue.use(CarouselPlugin);
  export default {};
</script>
<style>
  @import "../node_modules/@syncfusion/ej2-base/styles/material.css";
  @import "../node_modules/@syncfusion/ej2-buttons/styles/material.css";
  @import "../node_modules/@syncfusion/ej2-navigations/styles/material.css";
  .control-container {
    height: 360px;
    margin: 0 auto;
    width: 600px;
  }
  .img-container {
    height: 100%;
    margin: 0;
  }
  .img-caption {
    color: #fff;
    font-size: 1rem;
    position: absolute;
    bottom: 3rem;
    width: 100%;
    text-align: center;
  }
</style>

```

{% previewsample "page.domainurl/code-snippet/carousel/default-cs10" %}

#### *Previous and next button template*

Template options are provided to customize the previous button using [previousButtonTemplate](#) and the next button using [nextButtonTemplate](#). The following example depicts the code for applying the template to previous and next buttons in the carousel.

#### **APP.VUE**

```

<template>
  <div class="control-container">
    <ejs-carousel :previousButtonTemplate="'previousTemplate'"
    :nextButtonTemplate="'nextTemplate'">
      <template v-slot:previousTemplate>
        <ejs-button cssClass="e-flat e-round" iconCss="e-icons e-chevron-
left-double"></ejs-button>
      </template>
      <template v-slot:nextTemplate>
        <ejs-button cssClass="e-flat e-round" iconCss="e-icons e-chevron-
right-double"></ejs-button>
      </template>
      <e-carousel-items>
        <e-carousel-item template="Cardinal"></e-carousel-item>
        <template v-slot:Cardinal>
          <figure class="img-container">

```



```

        
        <figcaption class="img-caption">Cardinal</figcaption>
    </figure>
</template>
<e-carousel-item template="kingfisher"></e-carousel-item>
<template v-slot:kingfisher>
    <figure class="img-container">
        
        <figcaption class="img-caption">kingfisher</figcaption>
    </figure>
</template>
<e-carousel-item template="keel-billed-toucan"></e-carousel-item>
<template v-slot:keel-billed-toucan>
    <figure class="img-container">
        
        <figcaption class="img-caption">keel-billed-toucan</figcaption>
    </figure>
</template>
<e-carousel-item template="yellow-warbler"></e-carousel-item>
<template v-slot:yellow-warbler>
    <figure class="img-container">
        
        <figcaption class="img-caption">yellow-warbler</figcaption>
    </figure>
</template>
<e-carousel-item template="bee-eater"></e-carousel-item>
<template v-slot:bee-eater>
    <figure class="img-container">
        
        <figcaption class="img-caption">bee-eater</figcaption>
    </figure>
</template>
</e-carousel-items>
</ejs-carousel>
</div>
</template>
<script>
import Vue from "vue";
import { CarouselPlugin } from "@syncfusion/ej2-vue-navigations";
import { ButtonPlugin } from "@syncfusion/ej2-vue-buttons";
Vue.use(ButtonPlugin);
Vue.use(CarouselPlugin);

```

```

export default {};
</script>
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-buttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-navigations/styles/material.css";
.control-container {
  height: 360px;
  margin: 0 auto;
  width: 600px;
}
.img-container {
  height: 100%;
  margin: 0;
}
.img-caption {
  color: #fff;
  font-size: 1rem;
  position: absolute;
  bottom: 3rem;
  width: 100%;
  text-align: center;
}
</style>

```

{% previewsample "page.domainurl/code-snippet/carousel/default-cs11" %}

## Indicators

### Show or hide indicators

In indicators, the total slides and current slide state have been depicted. You can show/hide the indicators using the [showIndicators](#) property. The following example depicts the code to show/hide the indicators in the carousel.

### APP.VUE

```

<template>
  <div class="control-container">
    <ejs-carousel :showIndicators="true">
      <e-carousel-items>
        <e-carousel-item template="Cardinal"></e-carousel-item>
        <template v-slot:Cardinal>
          <figure class="img-container">
            
            <figcaption class="img-caption">Cardinal</figcaption>
          </figure>
        </template>
        <e-carousel-item template="kingfisher"></e-carousel-item>
        <template v-slot:kingfisher>
          <figure class="img-container">
            

```

```

        <figcaption class="img-caption">kingfisher</figcaption>
      </figure>
    </template>
    <e-carousel-item template="keel-billed-toucan"></e-carousel-item>
    <template v-slot:keel-billed-toucan>
      <figure class="img-container">
        
        <figcaption class="img-caption">keel-billed-toucan</figcaption>
      </figure>
    </template>
    <e-carousel-item template="yellow-warbler"></e-carousel-item>
    <template v-slot:yellow-warbler>
      <figure class="img-container">
        
        <figcaption class="img-caption">yellow-warbler</figcaption>
      </figure>
    </template>
    <e-carousel-item template="bee-eater"></e-carousel-item>
    <template v-slot:bee-eater>
      <figure class="img-container">
        
        <figcaption class="img-caption">bee-eater</figcaption>
      </figure>
    </template>
  </e-carousel-items>
</ejs-carousel>
</div>
</template>
<script>
  import Vue from "vue";
  import { CarouselPlugin } from "@syncfusion/ej2-vue-navigations";
  Vue.use(CarouselPlugin);
  export default {};
</script>
<style>
  @import "../node_modules/@syncfusion/ej2-base/styles/material.css";
  @import "../node_modules/@syncfusion/ej2-buttons/styles/material.css";
  @import "../node_modules/@syncfusion/ej2-navigations/styles/material.css";
  .control-container {
    height: 360px;
    margin: 0 auto;
    width: 600px;
  }
  .img-container {
    height: 100%;
    margin: 0;
  }
  .img-caption {

```

```

    color: #fff;
    font-size: 1rem;
    position: absolute;
    bottom: 3rem;
    width: 100%;
    text-align: center;
  }
</style>

```

{% previewsample "page.domainurl/code-snippet/carousel/default-cs12" %}

### Indicators Template

Template option is provided to customize the indicators by using the [indicatorTemplate](#) property. The following example depicts the code for applying a template to indicators in the carousel.

### APP.VUE

```

<template>
  <div class="control-container">
    <ejs-carousel :indicatorsTemplate="'indicatorsTemplate'">
      <template v-slot:indicatorsTemplate>
        <div class="indicator" indicator-index="{data.index}">
        </div>
      </template>
    <e-carousel-items>
      <e-carousel-item template="Cardinal"></e-carousel-item>
      <template v-slot:Cardinal>
        <figure class="img-container">
          
          <figcaption class="img-caption">Cardinal</figcaption>
        </figure>
      </template>
      <e-carousel-item template="kingfisher"></e-carousel-item>
      <template v-slot:kingfisher>
        <figure class="img-container">
          
          <figcaption class="img-caption">kingfisher</figcaption>
        </figure>
      </template>
      <e-carousel-item template="keel-billed-toucan"></e-carousel-item>
      <template v-slot:keel-billed-toucan>
        <figure class="img-container">
          
          <figcaption class="img-caption">keel-billed-toucan</figcaption>
        </figure>
      </template>
      <e-carousel-item template="yellow-warbler"></e-carousel-item>
    </e-carousel-items>
  </div>
</template>

```

```

        <template v-slot:yellow-warbler>
            <figure class="img-container">
                
                <figcaption class="img-caption">yellow-warbler</figcaption>
            </figure>
        </template>
        <e-carousel-item template="bee-eater"></e-carousel-item>
        <template v-slot:bee-eater>
            <figure class="img-container">
                
                <figcaption class="img-caption">bee-eater</figcaption>
            </figure>
        </template>
    </e-carousel-items>
</ejs-carousel>
</div>
</template>
<script>
import Vue from 'vue';
import { CarouselPlugin } from "@syncfusion/ej2-vue-navigations";
Vue.use(CarouselPlugin);
export default {};
</script>
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-buttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-navigations/styles/material.css";
.control-container {
    height: 360px;
    margin: 0 auto;
    width: 600px;
}
.img-container {
    height: 100%;
    margin: 0;
}
.img-caption {
    color: #fff;
    font-size: 1rem;
    position: absolute;
    bottom: 3rem;
    width: 100%;
    text-align: center;
}
.e-carousel .e-carousel-indicators .e-indicator-bars .e-indicator-bar.e-
template .indicator {
    background-color: #ecec;
    border-radius: 0.25rem;
    cursor: pointer;
    height: 0.5rem;
    margin: 0.5rem;

```

```
width: 1.5rem;
}
.e-carousel .e-carousel-indicators .e-indicator-bars .e-indicator-bar.e-
active .indicator {
  background-color: #3c78ef;
}</style>
```

{% previewsample "page.domainurl/code-snippet/carousel/default-cs13" %}

#### *Showing preview of slide in indicator*

You can customize the indicators by showing the preview image of each slide using the [indicatorTemplate](#) property. The following example depicts the code for showing the preview image using a template for indicators in the carousel.

#### **APP.VUE**

```
<template>
  <div class="control-container">
    <ejs-carousel :indicatorsTemplate="indicatorsTemplate">
      <e-carousel-items>
        <e-carousel-item template="<div class='slide-content'>Slide
1</div>"></e-carousel-item>
        <e-carousel-item template="<div class='slide-content'>Slide
2</div>"></e-carousel-item>
        <e-carousel-item template="<div class='slide-content'>Slide
3</div>"></e-carousel-item>
        <e-carousel-item template="<div class='slide-content'>Slide
4</div>"></e-carousel-item>
        <e-carousel-item template="<div class='slide-content'>Slide
5</div>"></e-carousel-item>
      </e-carousel-items>
    </ejs-carousel>
  </div>
</template>
<script>
import Vue from "vue";
import { CarouselPlugin } from "@syncfusion/ej2-vue-navigations";
Vue.use(CarouselPlugin);
var indicatorsVue = Vue.component("indicators", {
  template: `<div class="indicator" indicator-index="{data.index}">
    <div class="preview-content">{{getContent(data.index)}}</div>
  </div>`,
  computed: {},
  methods: {
    getContent: function (index) {
      var slides = ["Slide 1", "Slide 2", "Slide 3", "Slide 4", "Slide
5"];
      return slides[index];
    },
  },
});
export default {
  data: function () {
    return {
      indicatorsTemplate: function () {
        return {
```

```

        template: indicatorsVue,
      },
    },
  },
};
</script>
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-buttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-navigations/styles/material.css";
.e-carousel {
  background-color: #adb5bd;
  height: 300px !important;
  margin: 0 auto;
  width: 500px !important;
}
.e-carousel .slide-content {
  align-items: center;
  display: flex;
  font-size: 1.25rem;
  height: 100%;
  justify-content: center;
}
.e-carousel .e-carousel-items,
.e-carousel .e-carousel-navigators {
  height: calc(100% - 3rem);
}
.e-carousel .e-carousel-navigators .e-previous,
.e-carousel .e-carousel-navigators .e-next,
.e-carousel .e-carousel-navigators .nav-btn {
  padding: 0;
}
.e-carousel .e-carousel-navigators .nav-btn:active,
.e-carousel .e-carousel-navigators .nav-btn:focus,
.e-carousel .e-carousel-navigators .nav-btn:hover {
  background-color: transparent !important;
  color: inherit;
}
.e-carousel .e-carousel-navigators svg {
  fill: none;
  stroke: currentColor;
  stroke-linecap: square;
  stroke-width: 8px;
  height: 2rem;
  vertical-align: middle;
  width: 2rem;
}
.e-carousel .e-carousel-navigators .e-previous svg {
  transform: rotate(180deg);
}
.e-carousel .e-carousel-indicators .e-indicator-bars .e-indicator-bar
.indicator {
  background-color: #adb5bd;
  border: 1px solid black;
  border-radius: 0.25rem;
  cursor: pointer;
}

```

```

    height: 3.5rem;
    margin: 0.5rem;
    width: 5rem;
  }
  .e-carousel .e-carousel-indicators .e-indicator-bars .e-indicator-bar.e-
active .indicator {
    background-color: #c1cdda;
  }
  .preview-content {
    align-items: center;
    display: flex;
    height: 100%;
    justify-content: center;
  }
}
</style>

```

{% previewsample "page.domainurl/code-snippet/carousel/default-cs14" %}

### Indicators Types

Choose different types of indicators available using the [indicatorsType](#) property. The indicator types are categorized as follows:

- [Default Indicator](#)
- [Dynamic Indicator](#)
- [Fraction Indicator](#)
- [Progress Indicator](#)

### Default Indicator

A default indicator in a carousel is a set of dots that indicate the current position of the slide in the carousel. The Default indicator can be achieved by setting the [indicatorsType](#) to **Default**.

### APP.VUE

```

<template>
  <div class="control-container">
    <ejs-carousel :indicatorType="Default">
      <e-carousel-items>
        <e-carousel-item template="Cardinal"></e-carousel-item>
        <template v-slot:Cardinal>
          <figure class="img-container">
            
            <figcaption class="img-caption">Cardinal</figcaption>
          </figure>
        </template>
        <e-carousel-item template="kingfisher"></e-carousel-item>
        <template v-slot:kingfisher>
          <figure class="img-container">
            

```



```

        <figcaption class="img-caption">kingfisher</figcaption>
      </figure>
    </template>
    <e-carousel-item template="keel-billed-toucan"></e-carousel-item>
    <template v-slot:keel-billed-toucan>
      <figure class="img-container">
        
        <figcaption class="img-caption">keel-billed-toucan</figcaption>
      </figure>
    </template>
    <e-carousel-item template="yellow-warbler"></e-carousel-item>
    <template v-slot:yellow-warbler>
      <figure class="img-container">
        
        <figcaption class="img-caption">yellow-warbler</figcaption>
      </figure>
    </template>
    <e-carousel-item template="bee-eater"></e-carousel-item>
    <template v-slot:bee-eater>
      <figure class="img-container">
        
        <figcaption class="img-caption">bee-eater</figcaption>
      </figure>
    </template>
  </e-carousel-items>
</ejs-carousel>
</div>
</template>
<script>
import Vue from "vue";
import { CarouselPlugin } from "@syncfusion/ej2-vue-navigations";
Vue.use(CarouselPlugin);
export default {};
</script>
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-buttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-navigations/styles/material.css";
.control-container {
  height: 360px;
  margin: 0 auto;
  width: 600px;
}
.img-container {
  height: 100%;
  margin: 0;
}
.img-caption {

```

```

color: #fff;
font-size: 1rem;
position: absolute;
bottom: 3rem;
width: 100%;
text-align: center;
}
</style>

```

```
{% previewsample "page.domainurl/code-snippet/carousel/indicator-type-cs0" %}
```

### Dynamic Indicator

A dynamic indicator in a carousel provides visual cues or markers that dynamically change or update to indicate the current position. The Dynamic indicator can be achieved by setting the [indicatorsType](#) to **Dynamic**.

### APP.VUE

```

<template>
  <div class="control-container">
    <ejs-carousel :indicatorsType="Dynamic">
      <e-carousel-items>
        <e-carousel-item template="Cardinal"></e-carousel-item>
        <template v-slot:Cardinal>
          <figure class="img-container">
            
            <figcaption class="img-caption">Cardinal</figcaption>
          </figure>
        </template>
        <e-carousel-item template="kingfisher"></e-carousel-item>
        <template v-slot:kingfisher>
          <figure class="img-container">
            
            <figcaption class="img-caption">kingfisher</figcaption>
          </figure>
        </template>
        <e-carousel-item template="keel-billed-toucan"></e-carousel-item>
        <template v-slot:keel-billed-toucan>
          <figure class="img-container">
            
            <figcaption class="img-caption">keel-billed-toucan</figcaption>
          </figure>
        </template>
        <e-carousel-item template="yellow-warbler"></e-carousel-item>
        <template v-slot:yellow-warbler>
          <figure class="img-container">

```

```

        
        <figcaption class="img-caption">yellow-warbler</figcaption>
    </figure>
</template>
<e-carousel-item template="bee-eater"></e-carousel-item>
<template v-slot:bee-eater>
    <figure class="img-container">
        
        <figcaption class="img-caption">bee-eater</figcaption>
    </figure>
</template>
</e-carousel-items>
</ejs-carousel>
</div>
</template>
<script>
import Vue from "vue";
import { CarouselPlugin } from "@syncfusion/ej2-vue-navigations";
Vue.use(CarouselPlugin);
export default {};
</script>
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-buttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-navigations/styles/material.css";
.control-container {
    height: 360px;
    margin: 0 auto;
    width: 600px;
}
.img-container {
    height: 100%;
    margin: 0;
}
.img-caption {
    color: #fff;
    font-size: 1rem;
    position: absolute;
    bottom: 3rem;
    width: 100%;
    text-align: center;
}
</style>

```

{% previewsample "page.domainurl/code-snippet/carousel/indicator-type-cs1" %}

#### Fraction Indicator

The fraction indicator type displays the current slide index and total slide count as a fraction. The Fraction indicator can be achieved by setting the [indicatorsType](#) to [Fraction](#).

**APP.VUE**

```

<template>
  <div class="control-container">
    <ejs-carousel :indicatorsType="Fraction">
      <e-carousel-items>
        <e-carousel-item template="Cardinal"></e-carousel-item>
        <template v-slot:Cardinal>
          <figure class="img-container">
            
            <figcaption class="img-caption">Cardinal</figcaption>
          </figure>
        </template>
        <e-carousel-item template="kingfisher"></e-carousel-item>
        <template v-slot:kingfisher>
          <figure class="img-container">
            
            <figcaption class="img-caption">kingfisher</figcaption>
          </figure>
        </template>
        <e-carousel-item template="keel-billed-toucan"></e-carousel-item>
        <template v-slot:keel-billed-toucan>
          <figure class="img-container">
            
            <figcaption class="img-caption">keel-billed-toucan</figcaption>
          </figure>
        </template>
        <e-carousel-item template="yellow-warbler"></e-carousel-item>
        <template v-slot:yellow-warbler>
          <figure class="img-container">
            
            <figcaption class="img-caption">yellow-warbler</figcaption>
          </figure>
        </template>
        <e-carousel-item template="bee-eater"></e-carousel-item>
        <template v-slot:bee-eater>
          <figure class="img-container">
            
            <figcaption class="img-caption">bee-eater</figcaption>
          </figure>
        </template>
      </e-carousel-items>
    </ejs-carousel>
  </div>
</template>

```

```

    </ejs-carousel>
  </div>
</template>
<script>
import Vue from "vue";
import { CarouselPlugin } from "@syncfusion/ej2-vue-navigations";
Vue.use(CarouselPlugin);
export default {};
</script>
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-buttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-navigations/styles/material.css";
.control-container {
  height: 360px;
  margin: 0 auto;
  width: 600px;
}
.img-container {
  height: 100%;
  margin: 0;
}
.img-caption {
  color: #fff;
  font-size: 1rem;
  position: absolute;
  bottom: 3rem;
  width: 100%;
  text-align: center;
}
</style>

```

{% previewsample "page.domainurl/code-snippet/carousel/indicator-type-cs2" %}

### Progress Indicator

The Progress Indicator type displays the current slide as a progress bar. The Progress indicator can be achieved by setting the [indicatorsType](#) to **Progress**.

### APP.VUE

```

<template>
  <div class="control-container">
    <ejs-carousel :indicatorsType="Progress">
      <e-carousel-items>
        <e-carousel-item template="Cardinal"></e-carousel-item>
        <template v-slot:Cardinal>
          <figure class="img-container">
            
            <figcaption class="img-caption">Cardinal</figcaption>
          </figure>
        </template>
        <e-carousel-item template="kingfisher"></e-carousel-item>
        <template v-slot:kingfisher>

```

```

        <figure class="img-container">
          
          <figcaption class="img-caption">kingfisher</figcaption>
        </figure>
      </template>
      <e-carousel-item template="keel-billed-toucan"></e-carousel-item>
      <template v-slot:keel-billed-toucan>
        <figure class="img-container">
          
          <figcaption class="img-caption">keel-billed-toucan</figcaption>
        </figure>
      </template>
      <e-carousel-item template="yellow-warbler"></e-carousel-item>
      <template v-slot:yellow-warbler>
        <figure class="img-container">
          
          <figcaption class="img-caption">yellow-warbler</figcaption>
        </figure>
      </template>
      <e-carousel-item template="bee-eater"></e-carousel-item>
      <template v-slot:bee-eater>
        <figure class="img-container">
          
          <figcaption class="img-caption">bee-eater</figcaption>
        </figure>
      </template>
    </e-carousel-items>
  </ejs-carousel>
</div>
</template>
<script>
import Vue from "vue";
import { CarouselPlugin } from "@syncfusion/ej2-vue-navigations";
Vue.use(CarouselPlugin);
export default {};
</script>
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-buttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-navigations/styles/material.css";
.control-container {
  height: 360px;
  margin: 0 auto;
  width: 600px;
}

```

```
.img-container {
  height: 100%;
  margin: 0;
}
.img-caption {
  color: #fff;
  font-size: 1rem;
  position: absolute;
  bottom: 3rem;
  width: 100%;
  text-align: center;
}
</style>
```

{% previewsample "page.domainurl/code-snippet/carousel/indicator-type-cs3" %}

### Play button

#### Show or hide the play button

In the carousel, [autoPlay](#) actions have been controlled by using the [showPlayButton](#) property in the user interface. When you enable this property, the slide transitions are controlled using this play and pause button. This property depends on the [buttonsVisibility](#) property. The following example depicts the code to show the play button in the carousel.

### APP.VUE

```
<template>
  <div class="control-container">
    <ejs-carousel :showPlayButton="true">
      <e-carousel-items>
        <e-carousel-item template="Cardinal"></e-carousel-item>
        <template v-slot:Cardinal>
          <figure class="img-container">
            
            <figcaption class="img-caption">Cardinal</figcaption>
          </figure>
        </template>
        <e-carousel-item template="kingfisher"></e-carousel-item>
        <template v-slot:kingfisher>
          <figure class="img-container">
            
            <figcaption class="img-caption">kingfisher</figcaption>
          </figure>
        </template>
        <e-carousel-item template="keel-billed-toucan"></e-carousel-item>
        <template v-slot:keel-billed-toucan>
          <figure class="img-container">
            
            <figcaption class="img-caption">keel-billed-toucan</figcaption>
          </figure>
        </template>
      </e-carousel-items>
    </ejs-carousel>
  </div>
</template>
```

```

        style="height: 100%; width: 100%" />
        <figcaption class="img-caption">keel-billed-toucan</figcaption>
      </figure>
    </template>
    <e-carousel-item template="yellow-warbler"></e-carousel-item>
    <template v-slot:yellow-warbler>
      <figure class="img-container">
        
        <figcaption class="img-caption">yellow-warbler</figcaption>
      </figure>
    </template>
    <e-carousel-item template="bee-eater"></e-carousel-item>
    <template v-slot:bee-eater>
      <figure class="img-container">
        
        <figcaption class="img-caption">bee-eater</figcaption>
      </figure>
    </template>
  </e-carousel-items>
</ejs-carousel>
</div>
</template>
<script>
import Vue from "vue";
import { CarouselPlugin } from "@syncfusion/ej2-vue-navigations";
Vue.use(CarouselPlugin);
export default {};
</script>
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-buttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-navigations/styles/material.css";
.control-container {
  height: 360px;
  margin: 0 auto;
  width: 600px;
}
.img-container {
  height: 100%;
  margin: 0;
}
.img-caption {
  color: #fff;
  font-size: 1rem;
  position: absolute;
  bottom: 3rem;
  width: 100%;
  text-align: center;
}
</style>

```



```
{% previewsample "page.domainurl/code-snippet/carousel/default-cs15" %}
```

### Play button template

Template option is provided to customize the play button by using the [playButtonTemplate](#) property. The following example depicts the code for applying a template to play Button in the carousel.

### APP.VUE

```
<template>
  <div class="control-container">
    <ejs-carousel :showPlayButton="true"
    :playButtonTemplate="'playTemplate'">
      <template v-slot:playTemplate>
        <ejs-button ref="playBtn" cssClass="e-info playBtn" content="Pause"
        v-on:click="btnClick"></ejs-button>
      </template>
    <e-carousel-items>
      <e-carousel-item template="Cardinal"></e-carousel-item>
      <template v-slot:Cardinal>
        <figure class="img-container">
          
          <figcaption class="img-caption">Cardinal</figcaption>
        </figure>
      </template>
      <e-carousel-item template="kingfisher"></e-carousel-item>
      <template v-slot:kingfisher>
        <figure class="img-container">
          
          <figcaption class="img-caption">kingfisher</figcaption>
        </figure>
      </template>
      <e-carousel-item template="keel-billed-toucan"></e-carousel-item>
      <template v-slot:keel-billed-toucan>
        <figure class="img-container">
          
          <figcaption class="img-caption">keel-billed-toucan</figcaption>
        </figure>
      </template>
      <e-carousel-item template="yellow-warbler"></e-carousel-item>
      <template v-slot:yellow-warbler>
        <figure class="img-container">
          
          <figcaption class="img-caption">yellow-warbler</figcaption>
        </figure>
      </template>
    </e-carousel-items>
  </div>
</template>
```

```

    </figure>
  </template>
  <e-carousel-item template="bee-eater"></e-carousel-item>
  <template v-slot:bee-eater>
    <figure class="img-container">
      
      <figcaption class="img-caption">bee-eater</figcaption>
    </figure>
  </template>
</e-carousel-items>
</ejs-carousel>
</div>
</template>
<script>
import Vue from "vue";
import { CarouselPlugin } from "@syncfusion/ej2-vue-navigations";
import { ButtonPlugin } from "@syncfusion/ej2-vue-buttons";
Vue.use(ButtonPlugin);
Vue.use(CarouselPlugin);
export default {
  methods: {
    btnClick: function () {
      if (this.$refs.Carousel_instance.ej2Instances.autoPlay) {
        this.$refs.Carousel_instance.ej2Instances.autoPlay = false;
        this.$refs.playBtn.ej2Instances.content = "Play";
      } else {
        this.$refs.Carousel_instance.ej2Instances.autoPlay = true;
        this.$refs.playBtn.ej2Instances.content = "Pause";
      }
    },
  },
};
</script>
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-buttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-navigations/styles/material.css";
.control-container {
  height: 360px;
  margin: 0 auto;
  width: 600px;
}
.img-container {
  height: 100%;
  margin: 0;
}
.img-caption {
  color: #fff;
  font-size: 1rem;
  position: absolute;
  bottom: 3rem;
  width: 100%;
  text-align: center;
}

```

```
</style>
```

```
{% previewsample "page.domainurl/code-snippet/carousel/default-cs16" %}
```

## Animations and transitions in Vue Carousel component

### Animations

#### *Fade animation*

In Carousel, two built-in animations are provided for slide transitions. You can disable animation using the [animationEffect](#) property. By default, Slide animation is applied for the transition between slides.

The following demo depicts the example for fade animation,

#### **APP.VUE**

```
<template>
  <div class="control-container">
    <ejs-carousel animationEffect='Fade'>
      <e-carousel-items>
        <e-carousel-item template="Cardinal"></e-carousel-item>
        <template v-slot:Cardinal>
          <figure class="img-container">
            
            <figcaption class="img-caption">Cardinal</figcaption>
          </figure>
        </template>
        <e-carousel-item template="kingfisher"></e-carousel-item>
        <template v-slot:kingfisher>
          <figure class="img-container">
            
            <figcaption class="img-caption">kingfisher</figcaption>
          </figure>
        </template>
        <e-carousel-item template="keel-billed-toucan"></e-carousel-item>
        <template v-slot:keel-billed-toucan>
          <figure class="img-container">
            
            <figcaption class="img-caption">keel-billed-toucan</figcaption>
          </figure>
        </template>
        <e-carousel-item template="yellow-warbler"></e-carousel-item>
        <template v-slot:yellow-warbler>
          <figure class="img-container">
            
            <figcaption class="img-caption">yellow-warbler</figcaption>
          </figure>
        </template>
      </e-carousel-items>
    </ejs-carousel>
  </div>
</template>
```

```

        </figure>
      </template>
    <e-carousel-item template="bee-eater"></e-carousel-item>
    <template v-slot:bee-eater>
      <figure class="img-container">
        
        <figcaption class="img-caption">bee-eater</figcaption>
      </figure>
    </template>
  </e-carousel-items>
</ejs-carousel>
</div>
</template>
<script>
import Vue from "vue";
import { CarouselPlugin } from "@syncfusion/ej2-vue-navigations";
Vue.use(CarouselPlugin);
export default {};
</script>
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-buttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-navigations/styles/material.css";
.control-container {
  height: 360px;
  margin: 0 auto;
  width: 600px;
}
.img-container {
  height: 100%;
  margin: 0;
}
.img-caption {
  color: #fff;
  font-size: 1rem;
  position: absolute;
  bottom: 3rem;
  width: 100%;
  text-align: center;
}
</style>

```

```
{% previewsample "page.domainurl/code-snippet/carousel/default-cs1" %}
```

### Custom animation

In Carousel, you can use customized animation effects for slide transitions using the [Custom](#) option of the [animationEffect](#) property and apply custom animation css via [cssClass](#) property.

The following demo depicts the example for **parallax** custom animation,

### APP.VUE

```
<template>
```

```

<div class="control-container">
  <ejs-carousel animationEffect="Custom" cssClass="parallax">
    <e-carousel-items>
      <e-carousel-item template="Cardinal"></e-carousel-item>
      <template v-slot:Cardinal>
        <figure class="img-container">
          
          <figcaption class="img-caption">Cardinal</figcaption>
        </figure>
      </template>
      <e-carousel-item template="kingfisher"></e-carousel-item>
      <template v-slot:kingfisher>
        <figure class="img-container">
          
          <figcaption class="img-caption">kingfisher</figcaption>
        </figure>
      </template>
      <e-carousel-item template="keel-billed-toucan"></e-carousel-item>
      <template v-slot:keel-billed-toucan>
        <figure class="img-container">
          
          <figcaption class="img-caption">keel-billed-toucan</figcaption>
        </figure>
      </template>
      <e-carousel-item template="yellow-warbler"></e-carousel-item>
      <template v-slot:yellow-warbler>
        <figure class="img-container">
          
          <figcaption class="img-caption">yellow-warbler</figcaption>
        </figure>
      </template>
      <e-carousel-item template="bee-eater"></e-carousel-item>
      <template v-slot:bee-eater>
        <figure class="img-container">
          
          <figcaption class="img-caption">bee-eater</figcaption>
        </figure>
      </template>
    </e-carousel-items>
  </ejs-carousel>
</div>
</template>

```

```

<script>
import Vue from "vue";
import { CarouselPlugin } from "@syncfusion/ej2-vue-navigations";
Vue.use(CarouselPlugin);
export default {};
</script>
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-buttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-navigations/styles/material.css";
.control-container {
    height: 360px;
    margin: 0 auto;
    width: 600px;
}
.img-container {
    height: 100%;
    margin: 0;
}
.img-caption {
    color: #fff;
    font-size: 1rem;
    position: absolute;
    bottom: 3rem;
    width: 100%;
    text-align: center;
}
</style>

```

{% previewsample "page.domainurl/code-snippet/carousel/default-cs2" %}

### Intervals between slides

Using the items property, you can set different intervals for each item to transition between slides. The default interval is **5000 ms** (5 seconds). The following example depicts the code for setting the different intervals between each item.

### APP.VUE

```

<template>
  <div class="control-container">
    <ejs-carousel>
      <e-carousel-items>
        <e-carousel-item template="Cardinal"></e-carousel-item>
        <template v-slot:Cardinal>
          <figure class="img-container">
            
            <figcaption class="img-caption">Cardinal</figcaption>
          </figure>
        </template>
        <e-carousel-item template="kingfisher"></e-carousel-item>
        <template v-slot:kingfisher>
          <figure class="img-container">

```

```

        
        <figcaption class="img-caption">kingfisher</figcaption>
    </figure>
</template>
<e-carousel-item template="keel-billed-toucan"></e-carousel-item>
<template v-slot:keel-billed-toucan>
    <figure class="img-container">
        
        <figcaption class="img-caption">keel-billed-toucan</figcaption>
    </figure>
</template>
<e-carousel-item template="yellow-warbler"></e-carousel-item>
<template v-slot:yellow-warbler>
    <figure class="img-container">
        
        <figcaption class="img-caption">yellow-warbler</figcaption>
    </figure>
</template>
<e-carousel-item template="bee-eater"></e-carousel-item>
<template v-slot:bee-eater>
    <figure class="img-container">
        
        <figcaption class="img-caption">bee-eater</figcaption>
    </figure>
</template>
</e-carousel-items>
</ejs-carousel>
</div>
</template>
<script>
import Vue from "vue";
import { CarouselPlugin } from "@syncfusion/ej2-vue-navigations";
Vue.use(CarouselPlugin);
export default {
    data: function () {
        return {
            animation: { effect: "Fade" },
            firstInterval: 3000,
            secondInterval: 1000,
            thirdInterval: 2000,
            fourthInterval: 5000,
            fifthInterval: 6000,
        };
    },
};

```

```

</script>
<style>
  @import "../node_modules/@syncfusion/ej2-base/styles/material.css";
  @import "../node_modules/@syncfusion/ej2-buttons/styles/material.css";
  @import "../node_modules/@syncfusion/ej2-navigations/styles/material.css";
  .control-container {
    height: 360px;
    margin: 0 auto;
    width: 600px;
  }
  .img-container {
    height: 100%;
    margin: 0;
  }
  .img-caption {
    color: #fff;
    font-size: 1rem;
    position: absolute;
    bottom: 3rem;
    width: 100%;
    text-align: center;
  }
</style>

```

{% previewsample "page.domainurl/code-snippet/carousel/default-cs3" %}

**Note:** Interval property can accept value in terms of milliseconds.

### Auto play slides

In the carousel, all slides transitions are performed continuously after the specified or default intervals. You can enable or disable the auto slide transition using the [autoPlay](#) property. The following example depicts the code to enable or disable the auto slide transitions.

### APP.VUE

```

<template>
  <div class="control-container">
    <ejs-carousel :autoPlay="true">
      <e-carousel-items>
        <e-carousel-item template="Cardinal"></e-carousel-item>
        <template v-slot:Cardinal>
          <figure class="img-container">
            
            <figcaption class="img-caption">Cardinal</figcaption>
          </figure>
        </template>
        <e-carousel-item template="kingfisher"></e-carousel-item>
        <template v-slot:kingfisher>
          <figure class="img-container">
            

```



```

        <figcaption class="img-caption">kingfisher</figcaption>
      </figure>
    </template>
    <e-carousel-item template="keel-billed-toucan"></e-carousel-item>
    <template v-slot:keel-billed-toucan>
      <figure class="img-container">
        
        <figcaption class="img-caption">keel-billed-toucan</figcaption>
      </figure>
    </template>
    <e-carousel-item template="yellow-warbler"></e-carousel-item>
    <template v-slot:yellow-warbler>
      <figure class="img-container">
        
        <figcaption class="img-caption">yellow-warbler</figcaption>
      </figure>
    </template>
    <e-carousel-item template="bee-eater"></e-carousel-item>
    <template v-slot:bee-eater>
      <figure class="img-container">
        
        <figcaption class="img-caption">bee-eater</figcaption>
      </figure>
    </template>
  </e-carousel-items>
</ejs-carousel>
</div>
</template>
<script>
  import Vue from "vue";
  import { CarouselPlugin } from "@syncfusion/ej2-vue-navigations";
  Vue.use(CarouselPlugin);
  export default {};
</script>
<style>
  @import "../node_modules/@syncfusion/ej2-base/styles/material.css";
  @import "../node_modules/@syncfusion/ej2-buttons/styles/material.css";
  @import "../node_modules/@syncfusion/ej2-navigations/styles/material.css";
  .control-container {
    height: 360px;
    margin: 0 auto;
    width: 600px;
  }
  .img-container {
    height: 100%;
    margin: 0;
  }
  .img-caption {

```

```

    color: #fff;
    font-size: 1rem;
    position: absolute;
    bottom: 3rem;
    width: 100%;
    text-align: center;
  }
</style>

```

```
{% previewsample "page.domainurl/code-snippet/carousel/default-cs4" %}
```

### Pause on hover

By default, Slide transitions are paused when hovering the mouse pointer over the Carousel element. You can enable or disable this functionality using the [pauseOnHover](#) property.

The following example depicts the code to play the slides when hovering the mouse pointer over the Carousel element.

### APP.VUE

```

<template>
  <div class="control-container">
    <ejs-carousel :pauseOnHover="false">
      <e-carousel-items>
        <e-carousel-item template="Cardinal"></e-carousel-item>
        <template v-slot:Cardinal>
          <figure class="img-container">
            
            <figcaption class="img-caption">Cardinal</figcaption>
          </figure>
        </template>
        <e-carousel-item template="kingfisher"></e-carousel-item>
        <template v-slot:kingfisher>
          <figure class="img-container">
            
            <figcaption class="img-caption">kingfisher</figcaption>
          </figure>
        </template>
        <e-carousel-item template="keel-billed-toucan"></e-carousel-item>
        <template v-slot:keel-billed-toucan>
          <figure class="img-container">
            
            <figcaption class="img-caption">keel-billed-toucan</figcaption>
          </figure>
        </template>
        <e-carousel-item template="yellow-warbler"></e-carousel-item>
        <template v-slot:yellow-warbler>

```

```

        <figure class="img-container">
          
          <figcaption class="img-caption">yellow-warbler</figcaption>
        </figure>
      </template>
      <e-carousel-item template="bee-eater"></e-carousel-item>
      <template v-slot:bee-eater>
        <figure class="img-container">
          
          <figcaption class="img-caption">bee-eater</figcaption>
        </figure>
      </template>
    </e-carousel-items>
  </ejs-carousel>
</div>
</template>
<script>
  import Vue from "vue";
  import { CarouselPlugin } from "@syncfusion/ej2-vue-navigations";
  Vue.use(CarouselPlugin);
  export default {};
</script>
<style>
  @import "../node_modules/@syncfusion/ej2-base/styles/material.css";
  @import "../node_modules/@syncfusion/ej2-buttons/styles/material.css";
  @import "../node_modules/@syncfusion/ej2-navigations/styles/material.css";
  .control-container {
    height: 360px;
    margin: 0 auto;
    width: 600px;
  }
  .img-container {
    height: 100%;
    margin: 0;
  }
  .img-caption {
    color: #fff;
    font-size: 1rem;
    position: absolute;
    bottom: 3rem;
    width: 100%;
    text-align: center;
  }
</style>

```

{% previewsample "page.domainurl/code-snippet/carousel/default-cs5" %}

### Looping slides

In the carousel, slides transitions are repeated continuously when you reach the last slide by default. You can enable or disable the infinite slide transition using the [loop](#) property. The following example depicts the code to enable or disable the infinite slide transitions.

#### APP.VUE

```
<template>
  <div class="control-container">
    <ejs-carousel :loop="true">
      <e-carousel-items>
        <e-carousel-item template="Cardinal"></e-carousel-item>
        <template v-slot:Cardinal>
          <figure class="img-container">
            
            <figcaption class="img-caption">Cardinal</figcaption>
          </figure>
        </template>
        <e-carousel-item template="kingfisher"></e-carousel-item>
        <template v-slot:kingfisher>
          <figure class="img-container">
            
            <figcaption class="img-caption">kingfisher</figcaption>
          </figure>
        </template>
        <e-carousel-item template="keel-billed-toucan"></e-carousel-item>
        <template v-slot:keel-billed-toucan>
          <figure class="img-container">
            
            <figcaption class="img-caption">keel-billed-toucan</figcaption>
          </figure>
        </template>
        <e-carousel-item template="yellow-warbler"></e-carousel-item>
        <template v-slot:yellow-warbler>
          <figure class="img-container">
            
            <figcaption class="img-caption">yellow-warbler</figcaption>
          </figure>
        </template>
        <e-carousel-item template="bee-eater"></e-carousel-item>
        <template v-slot:bee-eater>
          <figure class="img-container">
            
            <figcaption class="img-caption">bee-eater</figcaption>
          </figure>
        </template>
      </e-carousel-items>
    </ejs-carousel>
  </div>
</template>
```

```

        style="height: 100%; width: 100%" />
        <figcaption class="img-caption">bee-eater</figcaption>
      </figure>
    </template>
  </e-carousel-items>
</ejs-carousel>
</div>
</template>
<script>
import Vue from "vue";
import { CarouselPlugin } from "@syncfusion/ej2-vue-navigations";
Vue.use(CarouselPlugin);
export default {};
</script>
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-buttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-navigations/styles/material.css";
.control-container {
  height: 360px;
  margin: 0 auto;
  width: 600px;
}
.img-container {
  height: 100%;
  margin: 0;
}
.img-caption {
  color: #fff;
  font-size: 1rem;
  position: absolute;
  bottom: 3rem;
  width: 100%;
  text-align: center;
}
</style>

```

{% previewsample "page.domainurl/code-snippet/carousel/default-cs6" %}

### Slide changing events

Using the [slideChanging](#) or [slideChanged](#) events of the Carousel component, you can perform sample end customization while the carousel items are switched.

The following demo depicts the example for carousel events,

### APP.VUE

```

<template>
  <div class="control-container">
    <ejs-carousel :slideChanging="onSlideChanging"
    :slideChanged="onSlideChanged">
      <e-carousel-items>
        <e-carousel-item template="Cardinal"></e-carousel-item>
        <template v-slot:Cardinal>
          <figure class="img-container">

```

```

        
        <figcaption class="img-caption">Cardinal</figcaption>
    </figure>
</template>
<e-carousel-item template="kingfisher"></e-carousel-item>
<template v-slot:kingfisher>
    <figure class="img-container">
        
        <figcaption class="img-caption">kingfisher</figcaption>
    </figure>
</template>
<e-carousel-item template="keel-billed-toucan"></e-carousel-item>
<template v-slot:keel-billed-toucan>
    <figure class="img-container">
        
        <figcaption class="img-caption">keel-billed-toucan</figcaption>
    </figure>
</template>
<e-carousel-item template="yellow-warbler"></e-carousel-item>
<template v-slot:yellow-warbler>
    <figure class="img-container">
        
        <figcaption class="img-caption">yellow-warbler</figcaption>
    </figure>
</template>
<e-carousel-item template="bee-eater"></e-carousel-item>
<template v-slot:bee-eater>
    <figure class="img-container">
        
        <figcaption class="img-caption">bee-eater</figcaption>
    </figure>
</template>
</e-carousel-items>
</ejs-carousel>
</div>
</template>
<script>
import Vue from "vue";
import { CarouselPlugin } from "@syncfusion/ej2-vue-navigations";
Vue.use(CarouselPlugin);
export default {
    methods: {

```

```

        onSlideChanging: function (args) {
            console.log(args.currentSlide); // You can customize the slide
before changing.
        },
        onSlideChanged: function (args) {
            console.log(args.currentSlide); // You can customize the slide after
changed.
        },
    },
};
</script>
<style>
    @import "../node_modules/@syncfusion/ej2-base/styles/material.css";
    @import "../node_modules/@syncfusion/ej2-buttons/styles/material.css";
    @import "../node_modules/@syncfusion/ej2-navigations/styles/material.css";
    .control-container {
        height: 360px;
        margin: 0 auto;
        width: 600px;
    }
    .img-container {
        height: 100%;
        margin: 0;
    }
    .img-caption {
        color: #fff;
        font-size: 1rem;
        position: absolute;
        bottom: 3rem;
        width: 100%;
        text-align: center;
    }
</style>

```

{% previewsample "page.domainurl/code-snippet/carousel/default-cs7" %}

### Disable touch swiping

In the carousel, you can swipe the carousel slides using touch actions by default. The swipe action can be enabled or disabled using the [enableTouchSwipe](#) property. The following example depicts the code to disable the swipe action for the slide.

### APP.VUE

```

<template>
    <div class="control-container">
        <ejs-carousel :enableTouchSwipe="false">
            <e-carousel-items>
                <e-carousel-item template="Cardinal"></e-carousel-item>
                <template v-slot:Cardinal>
                    <figure class="img-container">
                        
                        <figcaption class="img-caption">Cardinal</figcaption>
                    </figure>
                </template>
            </e-carousel-items>
        </ejs-carousel>
    </div>

```

```

    </template>
    <e-carousel-item template="kingfisher"></e-carousel-item>
    <template v-slot:kingfisher>
      <figure class="img-container">
        
        <figcaption class="img-caption">kingfisher</figcaption>
      </figure>
    </template>
    <e-carousel-item template="keel-billed-toucan"></e-carousel-item>
    <template v-slot:keel-billed-toucan>
      <figure class="img-container">
        
        <figcaption class="img-caption">keel-billed-toucan</figcaption>
      </figure>
    </template>
    <e-carousel-item template="yellow-warbler"></e-carousel-item>
    <template v-slot:yellow-warbler>
      <figure class="img-container">
        
        <figcaption class="img-caption">yellow-warbler</figcaption>
      </figure>
    </template>
    <e-carousel-item template="bee-eater"></e-carousel-item>
    <template v-slot:bee-eater>
      <figure class="img-container">
        
        <figcaption class="img-caption">bee-eater</figcaption>
      </figure>
    </template>
  </e-carousel-items>
</ejs-carousel>
</div>
</template>
<script>
import Vue from "vue";
import { CarouselPlugin } from "@syncfusion/ej2-vue-navigations";
Vue.use(CarouselPlugin);
export default {};
</script>
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-buttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-navigations/styles/material.css";
.control-container {
  height: 360px;

```



```

margin: 0 auto;
width: 600px;
}
.img-container {
height: 100%;
margin: 0;
}
.img-caption {
color: #fff;
font-size: 1rem;
position: absolute;
bottom: 3rem;
width: 100%;
text-align: center;
}
</style>

```

```
{% previewsample "page.domainurl/code-snippet/carousel/default-cs8" %}
```

### Swipe Modes

In the carousel, the [swipeMode](#) property allows specifying whether the slide transition should occur while performing swiping via touch or mouse. The slide swiping is enabled or disabled using the bitwise operator.

The following are the different swipe modes available in the carousel:

- CarouselSwipeMode.Touch - Allows the user to slide the slides using touch actions.
- CarouselSwipeMode.Mouse - Allows the user to slide the slides using mouse actions.
- CarouselSwipeMode.Touch & CarouselSwipeMode.Mouse - Allows the user to slide the slides using both touch and mouse actions.
- ~CarouselSwipeMode.Touch & ~CarouselSwipeMode.Mouse - Disables both touch and mouse actions.

### APP.VUE

```

<template>
  <div class="control-container">
    <ejs-carousel :swipeMode=swipeModes>
      <e-carousel-items>
        <e-carousel-item template="Cardinal"></e-carousel-item>
        <template v-slot:Cardinal>
          <figure class="img-container">
            
            <figcaption class="img-caption">Cardinal</figcaption>
          </figure>
        </template>
        <e-carousel-item template="kingfisher"></e-carousel-item>
        <template v-slot:kingfisher>
          <figure class="img-container">

```

```

        
        <figcaption class="img-caption">kingfisher</figcaption>
    </figure>
</template>
<e-carousel-item template="keel-billed-toucan"></e-carousel-item>
<template v-slot:keel-billed-toucan>
    <figure class="img-container">
        
        <figcaption class="img-caption">keel-billed-toucan</figcaption>
    </figure>
</template>
<e-carousel-item template="yellow-warbler"></e-carousel-item>
<template v-slot:yellow-warbler>
    <figure class="img-container">
        
        <figcaption class="img-caption">yellow-warbler</figcaption>
    </figure>
</template>
<e-carousel-item template="bee-eater"></e-carousel-item>
<template v-slot:bee-eater>
    <figure class="img-container">
        
        <figcaption class="img-caption">bee-eater</figcaption>
    </figure>
</template>
</e-carousel-items>
</ejs-carousel>
</div>
</template>
<script>
import Vue from "vue";
import { CarouselPlugin, CarouselSwipeMode } from "@syncfusion/ej2-vue-
navigations";
Vue.use(CarouselPlugin);
export default {
    data: function () {
        return {
            swipeModes: CarouselSwipeMode.Touch & CarouselSwipeMode.Mouse
        };
    }
};
</script>
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-buttons/styles/material.css";

```

```
@import "../node_modules/@syncfusion/ej2-navigations/styles/material.css";
.control-container {
  height: 360px;
  margin: 0 auto;
  width: 600px;
}
.img-container {
  height: 100%;
  margin: 0;
}
.img-caption {
  color: #fff;
  font-size: 1rem;
  position: absolute;
  bottom: 3rem;
  width: 100%;
  text-align: center;
}
</style>
```

{% previewsample "page.domainurl/code-snippet/carousel/swipe-cs1" %}

### Accessibility in Vue Carousel component

The Carousel component has been designed, keeping in mind the [WAI-ARIA](#) specifications, and applying the WAI-ARIA roles, states and properties along with keyboard support for people who use assistive devices. WAI-ARIA accessibility support is achieved through attributes like `aria-roledescription`, `aria-label`, `aria-current`, `aria-live`, `aria-role` and `aria-hidden`. It provides information about elements in a document for assistive technology. The component implements keyboard navigation support by following the [WAI-ARIA practices](#) and has been tested in major screen readers.

The accessibility compliance for the Carousel component is outlined below.

| Accessibility Criteria | Compatibility |

| -- | -- |

| [WCAG 2.2](#) Support |  |

| [Section 508](#) Support |  |

| Screen Reader Support |  |

| Right-To-Left Support |  |

| Color Contrast |  |

| Mobile Device Support |  |

| Keyboard Navigation Support |  |

| [Accessibility Checker](#) Validation |  |

| [Axe-core](#) Accessibility Validation |  |

<style>

.post .post-content img {

display: inline-block;

margin: 0.5em 0;

}

</style>

<div> - All features of the component meet the requirement.</div>

<div> - Some features of the component do not meet the requirement.</div>

<div> - The component does not meet the requirement.</div>

### ARIA attributes

The Carousel component is designed by considering [WAI-ARIA](#) standard. Carousel is supported with ARIA Accessibility which is accessible by on-screen readers and other assistive technology devices. The following list of attributes is added to the Carousel.

#### | Roles and Attributes | Functionalities

|

| ----- | -----  
----- |

| aria-roledescription | The role description attribute has been added for the root element (Carousel) and each Carousel slide item (slide). |

| aria-label | Previous, next and play/pause buttons and all indicator elements.

|

| aria-current | For the active item indicator element, `aria-current` is set to `true`.

|

| aria-hidden | For all carousel elements except the currently visible item, `aria-hidden` is set to `true`. |

| aria-live | For Carousel items element, when `autoPlay` is `true`, `aria-live` is set to `off`; when `autoPlay` is `false`, `aria-live` is set to `polite`. |

| aria-role | For carousel slide item, `aria-role` has been grouped.

|

### Keyboard interaction

By default, keyboard navigation is enabled. This component implements keyboard navigation support by following the WAI-ARIA practices. Once focused on the active Carousel element, you can use the following key combination for interacting with the Carousel.

Key	Description
Alt + J	Keys to focus the Carousel component (done at application end).
Arrows	Keys to navigate between slides.
Home	To navigate to the first slide.
End	To navigate to the last slide.
Space	To play/pause the slide transitions.
Enter	To perform the respective action on its focus.
Tab	To Move focus through the interactive elements.
Shift + Tab	To Move focus through the interactive elements.

### Ensuring accessibility

The Carousel component's accessibility levels are ensured through an [accessibility-checker](#) and [axe-core](#) software tools during automated testing.

The accessibility compliance of the Carousel component is shown in the following sample. Open the [sample](#) in a new window to evaluate the accessibility of the Carousel component with accessibility tools.

```
{% previewsample "https://ej2.syncfusion.com/accessibility/carousel.html" %}
```

See also

- [Accessibility in Syncfusion Vue components](#)

### Styles and appearance in Vue Carousel component

To modify the Carousel appearance, you need to override the default CSS of Carousel component. Please find the list of CSS classes and its corresponding section in Carousel component. Also, you have an option to create your own custom theme for the controls using our [Theme Studio](#).

### CSS Structure in Vue Carousel Control

The following content provides the exact CSS structure that can be used to modify the control's appearance based on user preference.

CSS Class | Purpose of Class

.e-carousel .e-carousel-item	To customize the carousel item
.e-carousel-item.e-active	To customize the active carousel item
.e-carousel .e-carousel-indicators	To customize the indicators
.e-carousel .e-carousel-indicators .e-indicator-bars .e-indicator-bar	To customize the indicator bars

|.e-carousel .e-carousel-indicators .e-indicator-bars .e-indicator-bar .e-indicator|To customize the individual indicator appearance

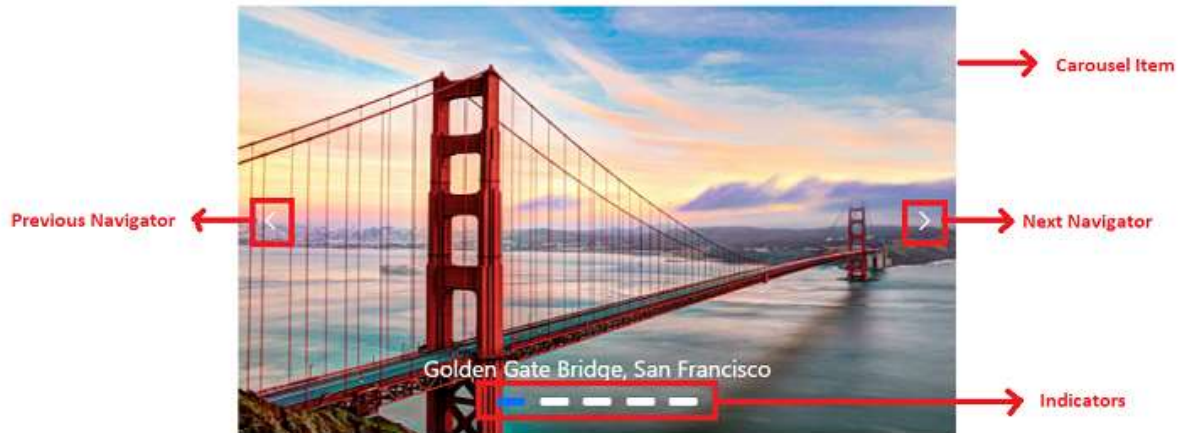
|.e-carousel .e-carousel-navigators|To customize the navigators

|.e-carousel .e-carousel-navigators .e-previous|To customize the previous button

|.e-carousel .e-carousel-navigators .e-next|To customize the next button

|.e-carousel .e-carousel-navigators .e-play-pause|To customize the play and pause button

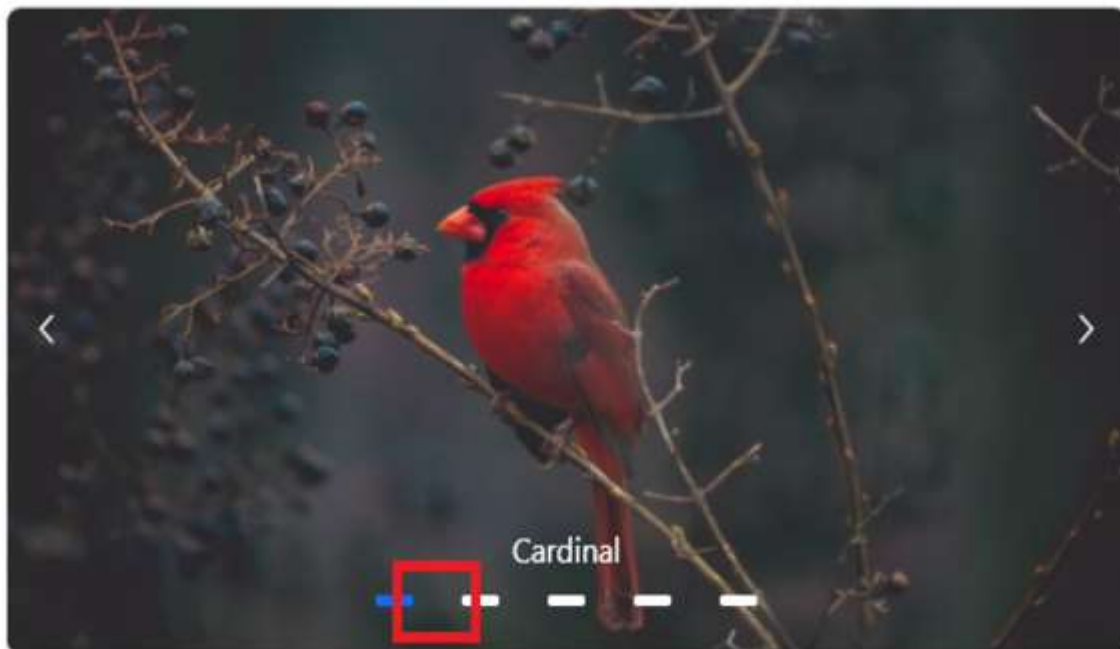
|.e-carousel.e-partial .e-carousel-slide-container|To customize the partial visible slides



### Customizing the indicators

Use the following CSS to customize the space between indicators by overriding the `.e-indicator-bar` CSS class.

```
`css
.e-carousel .e-carousel-indicators .e-indicator-bars .e-indicator-bar {
padding: 8px;
}
`
```



Use the following CSS to customize the indicators appearance by overriding the `.e-indicator` CSS class.

```
`css
```

```
.e-carousel .e-carousel-indicators .e-indicator-bars .e-indicator-bar .e-indicator {
```

```
width: 20px;
```

```
border-radius: 100%;
```

```
}
```

```
,
```

```
(./images/indicators-size.png)
```

Use the following CSS to render the indicators outside the carousel items by overriding the `.e-carousel-indicators` CSS class.

```
`css
```

```
.e-carousel .e-carousel-indicators {
```

```
bottom: auto;
```

```
}
```

```
,
```

```
(./images/indicator-outside.png)
```

### Customizing the navigators

Use the following CSS to customize the previous and next icon size and colors.

```
`css
```

```
.e-carousel .e-carousel-navigators .e-next .e-btn:not(:disabled) .e-btn-icon,  
.e-carousel .e-carousel-navigators .e-previous .e-btn:not(:disabled) .e-btn-icon  
{  
  color: greenyellow;  
  font-size: 25px;  
}
```



Use the following CSS to customize the navigators position to bottom by overriding the `.e-carousel-navigators` CSS class.

```
`css  
.e-carousel .e-carousel-navigators {  
  top: 120px;  
}
```





Use the following CSS to render the previous and next icon to outside the carousel items by overriding the `.e-previous` and `.e-next` CSS class.

```
`css
.e-carousel .e-carousel-navigators .e-previous,
.e-carousel .e-carousel-navigators .e-next
{
margin: -60px;
background: black;
}
`
```



### Customizing partial slides size

You can customize the partial slide size by overriding the `.e-carousel-slide-container` CSS class.

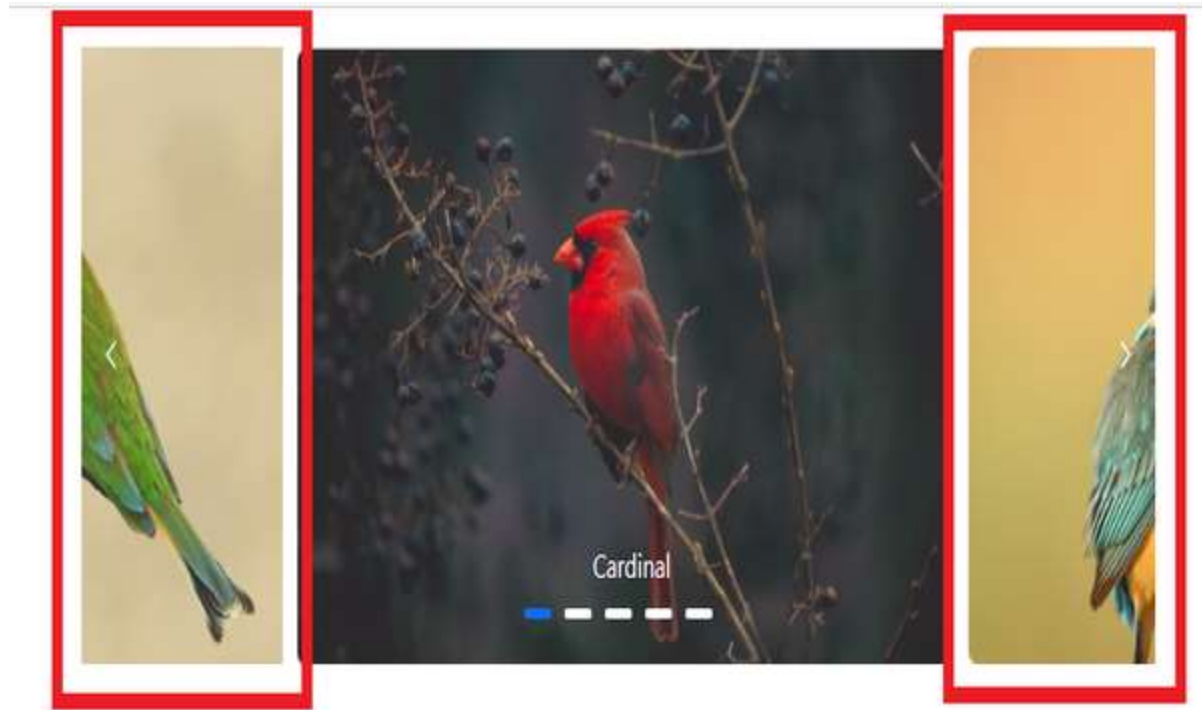
```
`css
```

```
.e-carousel.e-partial .e-carousel-slide-container{
```

```
padding: 0 150px;
```

```
}
```

```
`
```



## 3D Chart

### Getting started with the Vue 3D chart component in Vue 2

This article provides a step-by-step guide for setting up a Vue 2 project using [Vue-CLI](#) and integrating the Syncfusion Vue 3D Chart component.

#### Prerequisites

[System requirements for Syncfusion Vue UI components](#)

#### Dependencies

Below is the list of minimum dependencies required to use the 3D Chart component.

```
`javascript
|-- @syncfusion/ej2-vue-charts
|-- @syncfusion/ej2-base
|-- @syncfusion/ej2-data
|-- @syncfusion/ej2-pdf-export
|-- @syncfusion/ej2-file-utils
|-- @syncfusion/ej2-compression
|-- @syncfusion/ej2-charts
|-- @syncfusion/ej2-vue-base
|-- @syncfusion/ej2-svg-base
`,`
```

### Setting up the Vue 2 project

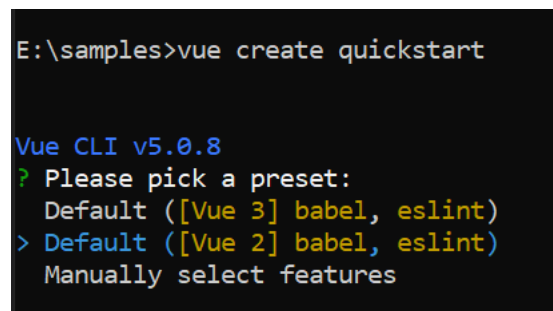
To generate a Vue 2 project using Vue-CLI, use the [vue create](#) command. Follow these steps to install Vue CLI and create a new project:

```
`bash
npm install -g @vue/cli
vue create quickstart
cd quickstart
npm run serve
`
```

or

```
`bash
yarn global add @vue/cli
vue create quickstart
cd quickstart
yarn run serve
`
```

When creating a new project, choose the option `Default ([Vue 2] babel, eslint)` from the menu.



```
E:\samples>vue create quickstart

Vue CLI v5.0.8
? Please pick a preset:
  Default ([Vue 3] babel, eslint)
> Default ([Vue 2] babel, eslint)
  Manually select features
```

Once the `quickstart` project is set up with default settings, proceed to add Syncfusion components to the project.

### Add Syncfusion Vue packages

Syncfusion packages are available at [npmjs.com](https://www.npmjs.com). To use Vue components, install the required npm package.

This article uses the `Vue 3D Charts component` as an example. Install the `@syncfusion/ej2-vue-charts` package by running the following command:

```
`bash
npm install @syncfusion/ej2-vue-charts --save
`

or
```

```
`bash
```

```
yarn add @syncfusion/ej2-vue-charts
```

```
,
```

The **--save** will instruct NPM to include the chart package inside of the **dependencies** section of the **package.json**.

### Add Syncfusion Vue component

Follow the below steps to add the Vue 3D Chart component:

1\ First, import and register the 3D Chart component in the **script** section of the **src/App.vue** file.

#### ~/SRC/APP.VUE

```
<script>
import { Chart3DComponent } from '@syncfusion/ej2-vue-charts';
export default {
  components: {
    'ejs-chart3d': Chart3DComponent
  }
}
</script>
```

2\ In the **template** section, define the 3D Chart component.

#### ~/SRC/APP.VUE

```
<template>
<div id="app">
  <ejs-chart3d id="container"> </ejs-chart3d>
</div>
</template>
```

Here is the summarized code for the above steps in the **src/App.vue** file:

#### ~/SRC/APP.VUE

```
<template>
  <div id="app">
    <ejs-chart3d id="container"> </ejs-chart3d>
  </div>
</template>
<script>
import { Chart3DComponent } from '@syncfusion/ej2-vue-charts';
export default {
  components: {
    'ejs-chart3d': Chart3DComponent
  },
  data () {
    return {
    }
  }
}
</script>
<style>
```

```
#container{  
  height: 350px;  
}  
</style>
```

### Run the project

To run the project, use the following command:

```
`bash
```

```
npm run serve
```

```
,
```

```
or
```

```
`bash
```

```
yarn run serve
```

```
,
```

```
{% previewsample "page.domainurl/code-snippet/chart3d/getting-started/sample-cs1" %}
```

### Module injection

3D Chart component are segregated into individual feature-wise modules. In order to use a particular feature, you need to inject its feature service in the AppModule. In the current application, we are going to modify the above basic 3D Chart to visualize sales data for a particular year. For this application we are going to use column series, tooltip, data label, category axis and legend feature of the 3D Chart. Please find relevant feature service name and description as follows. *ColumnSeries3D - Inject this provider to use column series.* **Legend3D** - Inject this provider to use legend feature. *Tooltip3D - Inject this provider to use tooltip feature.* **DataLabel3D** - Inject this provider to use datalabel feature. \* **Category3D** - Inject this provider to use category feature.

These modules should be injected to the provide section as follows,

```
`javascript
```

```
import { Chart3DComponent, ColumnSeries3D } from "@syncfusion/ej2-vue-charts";
```

```
export default {
```

```
  components: {
```

```
    'ejs-chart3d': Chart3DComponent
```

```
  },
```

```
  provide: {
```

```
    chart3d: [ColumnSeries3D]
```

```
  }
```

```
};
```

```
</script>
```

```
,
```

### Populate 3D chart with data

This section explains how to plot below JSON data to the 3D Chart.

```
`javascript
export default {
  data() {
    return {
      seriesData: [
        { month: 'Jan', sales: 35 }, { month: 'Feb', sales: 28 },
        { month: 'Mar', sales: 34 }, { month: 'Apr', sales: 32 },
        { month: 'May', sales: 40 }, { month: 'Jun', sales: 32 },
        { month: 'Jul', sales: 35 }, { month: 'Aug', sales: 55 },
        { month: 'Sep', sales: 38 }, { month: 'Oct', sales: 30 },
        { month: 'Nov', sales: 25 }, { month: 'Dec', sales: 32 }
      ]
    };
  }
};
```

- Add a series object to the 3D Chart by using [series](#) property. Now map the field names `month` and `sales` in the JSON data to the [xName](#) and [yName](#) properties of the series, then set the JSON data to [dataSource](#) property.

Since the JSON contains category data, set the [valueType](#) for horizontal axis to `Category`. By default, the axis valueType is `Numeric`.

### ~/SRC/APP.VUE

```
<template>
  <div id="app">
    <ejs-chart3d id="container" :primaryXAxis='primaryXAxis'>
      <e-chart3d-series-collection>
        <e-chart3d-series :dataSource='seriesData' type='Column'
xName='month' yName='sales' name='Sales'> </e-chart3d-series>
      </e-chart3d-series-collection>
    </ejs-chart3d>
  </div>
</template>
<script>
import { Chart3DComponent, Chart3DSeriesCollectionDirective,
Chart3DSeriesDirective, ColumnSeries3D, Category3D } from "@syncfusion/ej2-
vue-charts";
export default {
  components: {
```

```

    'ejs-chart3d': Chart3DComponent,
    'e-chart3d-series-collection': Chart3DSeriesCollectionDirective,
    'e-chart3d-series': Chart3DSeriesDirective
  },
  data() {
    return {
      seriesData: [
        { month: 'Jan', sales: 35 }, { month: 'Feb', sales: 28 },
        { month: 'Mar', sales: 34 }, { month: 'Apr', sales: 32 },
        { month: 'May', sales: 40 }, { month: 'Jun', sales: 32 },
        { month: 'Jul', sales: 35 }, { month: 'Aug', sales: 55 },
        { month: 'Sep', sales: 38 }, { month: 'Oct', sales: 30 },
        { month: 'Nov', sales: 25 }, { month: 'Dec', sales: 32 }
      ],
      primaryXAxis: {
        valueType: 'Category'
      }
    };
  },
  provide: {
    chart3d: [ColumnSeries3D, Category3D]
  }
};
</script>
<style>
#container {
  height: 350px;
}
</style>

```

{% previewsample "page.domainurl/code-snippet/chart3d/getting-started/sample-cs2" %}

- The sales data are in thousands, so format the vertical axis label by adding \$ as a prefix and K as a suffix to each label. This can be achieved by setting the `labelFormat` property of axis. Here, `{value}` act as a placeholder for each axis label.

#### Add 3D chart title

You can add a title using `title` property to the 3D Chart to provide quick information to the user about the data plotted in the 3D Chart.

#### ~/SRC/APP.VUE

```

<template>
  <div id="app">
    <ejs-chart3d id="container" :title='title'
      :primaryXAxis='primaryXAxis' :primaryYAxis='primaryYAxis'>
      <e-chart3d-series-collection>
        <e-chart3d-series :dataSource='seriesData' type='Column'
          xName='month' yName='sales' name='Sales'> </e-chart3d-series>
      </e-chart3d-series-collection>
    </ejs-chart3d>
  </div>
</template>
<script>

```



```

import { Chart3DComponent, Chart3DSeriesCollectionDirective,
Chart3DSeriesDirective, ColumnSeries3D, Category3D } from "@syncfusion/ej2-
vue-charts";
export default {
  components: {
    'ejs-chart3d': Chart3DComponent,
    'e-chart3d-series-collection': Chart3DSeriesCollectionDirective,
    'e-chart3d-series': Chart3DSeriesDirective
  },
  data() {
    return {
      seriesData: [
        { month: 'Jan', sales: 35 }, { month: 'Feb', sales: 28 },
        { month: 'Mar', sales: 34 }, { month: 'Apr', sales: 32 },
        { month: 'May', sales: 40 }, { month: 'Jun', sales: 32 },
        { month: 'Jul', sales: 35 }, { month: 'Aug', sales: 55 },
        { month: 'Sep', sales: 38 }, { month: 'Oct', sales: 30 },
        { month: 'Nov', sales: 25 }, { month: 'Dec', sales: 32 }
      ],
      primaryXAxis: {
        valueType: 'Category'
      },
      primaryYAxis: {
        labelFormat: '${value}K'
      },
      title: "Sales Analysis"
    };
  },
  provide: {
    chart3d: [ColumnSeries3D, Category3D]
  }
};
</script>
<style>
  #container{
    height: 350px;
  }
</style>

```

{% previewsample "page.domainurl/code-snippet/chart3d/getting-started/sample-cs3" %}

### Enable legend

You can use legend for the 3D Chart by setting the [visible](#) property to true in [legendSettings](#) object and by injecting the `Legend3D` into the `provide`.

### ~/SRC/APP.VUE

```

<template>
  <div id="app">
    <ejs-chart3d id="container" :title='title'
:primaryXAxis='primaryXAxis' :primaryYAxis='primaryYAxis'
:legendSettings='legendSettings'>
      <e-chart3d-series-collection>
        <e-chart3d-series :dataSource='seriesData' type='Column'
xName='month' yName='sales' name='Sales'> </e-chart3d-series>
      </e-chart3d-series-collection>
    </div>

```

```

        </ejs-chart3d>
      </div>
    </template>
    <script>
    import { Chart3DComponent, Chart3DSeriesCollectionDirective,
    Chart3DSeriesDirective, ColumnSeries3D, Category3D, Legend3D } from
    "@syncfusion/ej2-vue-charts";
    export default {
      components: {
        'ejs-chart3d': Chart3DComponent,
        'e-chart3d-series-collection': Chart3DSeriesCollectionDirective,
        'e-chart3d-series': Chart3DSeriesDirective
      },
      data() {
        return {
          seriesData: [
            { month: 'Jan', sales: 35 }, { month: 'Feb', sales: 28 },
            { month: 'Mar', sales: 34 }, { month: 'Apr', sales: 32 },
            { month: 'May', sales: 40 }, { month: 'Jun', sales: 32 },
            { month: 'Jul', sales: 35 }, { month: 'Aug', sales: 55 },
            { month: 'Sep', sales: 38 }, { month: 'Oct', sales: 30 },
            { month: 'Nov', sales: 25 }, { month: 'Dec', sales: 32 }
          ],
          primaryXAxis: {
            valueType: 'Category'
          },
          primaryYAxis: {
            labelFormat: '${value}K'
          },
          legendSettings: {
            visible: true
          },
          title: "Sales Analysis"
        };
      },
      provide: {
        chart3d: [ColumnSeries3D, Category3D, Legend3D]
      }
    };
    </script>
    <style>
    #container {
      height: 350px;
    }
    </style>
  
```

{% previewsample "page.domainurl/code-snippet/chart3d/getting-started/sample-cs4" %}

#### Add data label

You can add data labels to improve the readability of the 3D Chart. This can be achieved by setting the [visible](#) property to true in the [dataLabel](#) object and by injecting `DataLabel3D` into the `provide`.

#### ~/SRC/APP.VUE

```

<template>
  <div id="app">
  
```

```

    <ejs-chart3d id="container" :title='title'
:primaryXAxis='primaryXAxis' :primaryYAxis='primaryYAxis'
:legendSettings='legendSettings'>
      <e-chart3d-series-collection>
        <e-chart3d-series :dataSource='seriesData' type='Column'
xName='month' yName='sales' name='Sales' :dataLabel='dataLabel'> </e-
chart3d-series>
      </e-chart3d-series-collection>
    </ejs-chart3d>
  </div>
</template>
<script>
import { Chart3DComponent, Chart3DSeriesCollectionDirective,
Chart3DSeriesDirective, ColumnSeries3D, Category3D, Legend3D, DataLabel3D }
from "@syncfusion/ej2-vue-charts";
export default {
  components: {
    'ejs-chart3d': Chart3DComponent,
    'e-chart3d-series-collection': Chart3DSeriesCollectionDirective,
    'e-chart3d-series': Chart3DSeriesDirective
  },
  data() {
    return {
      seriesData: [
        { month: 'Jan', sales: 35 }, { month: 'Feb', sales: 28 },
        { month: 'Mar', sales: 34 }, { month: 'Apr', sales: 32 },
        { month: 'May', sales: 40 }, { month: 'Jun', sales: 32 },
        { month: 'Jul', sales: 35 }, { month: 'Aug', sales: 55 },
        { month: 'Sep', sales: 38 }, { month: 'Oct', sales: 30 },
        { month: 'Nov', sales: 25 }, { month: 'Dec', sales: 32 }
      ],
      primaryXAxis: {
        valueType: 'Category'
      },
      primaryYAxis:{
        labelFormat: '${value}K'
      },
      legendSettings: {
        visible: true
      },
      dataLabel:{
        visible: true
      },
      title: "Sales Analysis"
    };
  },
  provide: {
    chart3d: [ColumnSeries3D, Category3D, Legend3D, DataLabel3D]
  }
};
</script>
<style>
  #container {
    height: 350px;
  }
</style>

```

```
{% previewsample "page.domainurl/code-snippet/chart3d/getting-started/sample-cs5" %}
```

### Enable tooltip

The tooltip is useful when you cannot display information by using the data labels due to space constraints. You can enable tooltip by setting the [enable](#) property as true in [tooltip](#) object and by injecting `Tooltip3D` into the `provide`.

#### ~/SRC/APP.VUE

```
<template>
  <div id="app">
    <ejs-chart3d id="container" :title='title'
:primaryXAxis='primaryXAxis' :primaryYAxis='primaryYAxis'
:tooltip='tooltip'>
      <e-chart3d-series-collection>
        <e-chart3d-series :dataSource='seriesData' type='Column'
xName='month' yName='sales' name='Sales' :dataLabel='dataLabel'> </e-
chart3d-series>
      </e-chart3d-series-collection>
    </ejs-chart3d>
  </div>
</template>
<script>
import { Chart3DComponent, Chart3DSeriesCollectionDirective,
Chart3DSeriesDirective, ColumnSeries3D, Category3D, DataLabel3D, Tooltip3D }
from "@syncfusion/ej2-vue-charts";
export default {
  components: {
    'ejs-chart3d': Chart3DComponent,
    'e-chart3d-series-collection': Chart3DSeriesCollectionDirective,
    'e-chart3d-series': Chart3DSeriesDirective
  },
  data() {
    return {
      seriesData: [
        { month: 'Jan', sales: 35 }, { month: 'Feb', sales: 28 },
        { month: 'Mar', sales: 34 }, { month: 'Apr', sales: 32 },
        { month: 'May', sales: 40 }, { month: 'Jun', sales: 32 },
        { month: 'Jul', sales: 35 }, { month: 'Aug', sales: 55 },
        { month: 'Sep', sales: 38 }, { month: 'Oct', sales: 30 },
        { month: 'Nov', sales: 25 }, { month: 'Dec', sales: 32 }
      ],
      primaryXAxis: {
        valueType: 'Category'
      },
      primaryYAxis:{
        labelFormat: '${value}K'
      },
      dataLabel:{
        visible: true
      },
      tooltip:{
        enable: true
      },
      title: "Sales Analysis"
    }
  }
}
```

```
    };  
  },  
  provide: {  
    chart3d: [ColumnSeries3D, Category3D, DataLabel3D, Tooltip3D]  
  }  
};  
</script>  
<style>  
  #container {  
    height: 350px;  
  }  
</style>
```

{% previewsample "page.domainurl/code-snippet/chart3d/getting-started/sample-cs6" %}

You can refer to our [Vue 3D Charts](#) feature tour page for its groundbreaking feature representations. You can also explore our [Vue 3D Charts example](#) that shows various 3D Chart types and how to represent time-dependent data, showing trends in data at equal intervals.

### Getting started with the Vue 3D chart component in Vue 3

This article provides a step-by-step guide for setting up a [Vite](#) project with a JavaScript environment and integrating the Syncfusion Vue 3D Chart component using the [Composition API](#) / [Options API](#).

The **Composition API** is a new feature introduced in Vue.js 3 that provides an alternative way to organize and reuse component logic. It allows developers to write components as functions that use smaller, reusable functions called composition functions to manage their properties and behavior.

The **Options API** is the traditional way of writing Vue.js components, where the component logic is organized into a series of options that define the component's properties and behavior. These options include data, methods, computed properties, watchers, lifecycle hooks, and more.

#### Prerequisites

##### [System requirements for Syncfusion Vue UI components](#)

##### Set up the Vite project

A recommended approach for beginning with Vue is to scaffold a project using [Vite](#). To create a new Vite project, use one of the commands that are specific to either NPM or Yarn.

```
`bash
```

```
npm create vite@latest
```

```
,
```

```
or
```

```
`bash
```

```
yarn create vite
```

```
,
```

Using one of the above commands will lead you to set up additional configurations for the project as below:

1. Define the project name: We can specify the name of the project directly. Let's specify the name of the project as **my-project** for this article.

```
`bash
```

```
? Project name: » my-project
```

```
,
```

2. Select **Vue** as the framework. It will create a Vue 3 project.

```
`bash
```

```
? Select a framework: » - Use arrow-keys. Return to submit.
```

```
Vanilla
```

```
Vue
```

```
React
```

```
Preact
```

```
Lit
```

```
Svelte
```

```
Others
```

```
,
```

3. Choose **JavaScript** as the framework variant to build this Vite project using JavaScript and Vue.

```
`bash
```

```
? Select a variant: » - Use arrow-keys. Return to submit.
```

```
JavaScript
```

```
TypeScript
```

```
Customize with create-vue ↗
```

```
Nuxt ↗
```

```
,
```

4. Upon completing the aforementioned steps to create the **my-project**, run the following command to install its dependencies:

```
`bash
```

```
cd my-project
```

```
npm install
```

```
,
```

```
or
```

```
`bash
```

```
cd my-project
```

```
yarn install
```

```
,
```

Now that `my-project` is ready to run with default settings, let's add Syncfusion components to the project.

#### Add Syncfusion Vue packages

Syncfusion Vue component packages are available at [npmjs.com](https://www.npmjs.com). To use Syncfusion Vue components in the project, install the corresponding npm package.

This article uses the `Vue 3D Chart component` as an example. To use the `Vue 3D Chart component` in the project, the `@syncfusion/ej2-vue-charts` package needs to be installed using the following command:

```
`bash
```

```
npm install @syncfusion/ej2-vue-charts --save
```

```
,
```

or

```
`bash
```

```
yarn add @syncfusion/ej2-vue-charts
```

```
,
```

#### Add Syncfusion Vue component

Follow the below steps to add the `Vue 3D Chart component` using `Composition API` or `Options API`:

1. First, import and register the `3D Chart component` and its child directives in the `script` section of the `src/App.vue` file. If you are using the `Composition API`, you should add the `setup` attribute to the `script` tag to indicate that Vue will be using the `Composition API`.

##### COMPOSITION API (~SRC/APP.VUE)

```
<script setup>
import { Chart3DComponent as EjsChart3d, Chart3DSeriesCollectionDirective as
EChart3dSeriesCollection, Chart3DSeriesDirective as EChart3dSeries,
ColumnSeries3D, Legend3D, Category3D } from "@syncfusion/ej2-vue-charts";
</script>
```

##### OPTIONS API (~SRC/APP.VUE)

```
<script>
import { Chart3DComponent, Chart3DSeriesCollectionDirective,
Chart3DSeriesDirective, ColumnSeries3D, Legend3D, Category3D } from
'@syncfusion/ej2-vue-charts';
//Component registration
export default {
  name: "App",
  components: {
    'ejs-chart3d' : Chart3DComponent,
    'e-chart3d-series-collection' : Chart3DSeriesCollectionDirective,
    'e-chart3d-series' : Chart3DSeriesDirective
  }
}
</script>
```

2. In the `template` section, define the 3D Chart component with the `dataSource` property.

#### ~/SRC/APP.VUE

```
<template>
<ejs-chart3d id="container" :title='title' :primaryXAxis='primaryXAxis'>
<e-chart3d-series-collection>
<e-chart3d-series :dataSource='seriesData' type='Column' xName='month'
yName='sales' name='Sales'>
</e-chart3d-series>
</e-chart3d-series-collection>
</ejs-chart3d>
</template>
```

3. Declare the values for the `dataSource` property in the `script` section.

#### COMPOSITION API (~/SRC/APP.VUE)

```
<script setup>
const seriesData = [
{ month: 'Jan', sales: 35 }, { month: 'Feb', sales: 28 },
{ month: 'Mar', sales: 34 }, { month: 'Apr', sales: 32 },
{ month: 'May', sales: 40 }, { month: 'Jun', sales: 32 },
{ month: 'Jul', sales: 35 }, { month: 'Aug', sales: 55 },
{ month: 'Sep', sales: 38 }, { month: 'Oct', sales: 30 },
{ month: 'Nov', sales: 25 }, { month: 'Dec', sales: 32 }
];
</script>
```

#### OPTIONS API (~/SRC/APP.VUE)

```
<script>
data() {
return {
seriesData: [
{ month: 'Jan', sales: 35 }, { month: 'Feb', sales: 28 },
{ month: 'Mar', sales: 34 }, { month: 'Apr', sales: 32 },
{ month: 'May', sales: 40 }, { month: 'Jun', sales: 32 },
{ month: 'Jul', sales: 35 }, { month: 'Aug', sales: 55 },
{ month: 'Sep', sales: 38 }, { month: 'Oct', sales: 30 },
{ month: 'Nov', sales: 25 }, { month: 'Dec', sales: 32 }
]
};
}
</script>
```

Here is the summarized code for the above steps in the `src/App.vue` file:

#### COMPOSITION API (~/SRC/APP.VUE)

```
<template>
<ejs-chart3d id="container" :title='title' :primaryXAxis='primaryXAxis'>
<e-chart3d-series-collection>
<e-chart3d-series :dataSource='seriesData' type='Column' xName='month'
yName='sales' name='Sales'>
</e-chart3d-series>
```



```

</e-chart3d-series-collection>
</ejs-chart3d>
</template>
<script setup>
import { provide } from 'vue';
import { Chart3DComponent as EjsChart3d, Chart3DSeriesCollectionDirective as EChart3dSeriesCollection, Chart3DSeriesDirective as EChart3dSeries, ColumnSeries3D, Category3D, Legend3D } from "@syncfusion/ej2-vue-charts";
const seriesData = [
  { month: 'Jan', sales: 35 }, { month: 'Feb', sales: 28 },
  { month: 'Mar', sales: 34 }, { month: 'Apr', sales: 32 },
  { month: 'May', sales: 40 }, { month: 'Jun', sales: 32 },
  { month: 'Jul', sales: 35 }, { month: 'Aug', sales: 55 },
  { month: 'Sep', sales: 38 }, { month: 'Oct', sales: 30 },
  { month: 'Nov', sales: 25 }, { month: 'Dec', sales: 32 }
];
const title = 'Sales Analysis';
const primaryXAxis = { valueType: 'Category' };
const chart3d = [ColumnSeries3D, Category3D, Legend3D];
provide('chart3d', chart3d);
</script>

```

### OPTIONS API (~SRC/APP.VUE)

```

<template>
<ejs-chart3d id="container" :title='title' :primaryXAxis='primaryXAxis'>
<e-chart3d-series-collection>
<e-chart3d-series :dataSource='seriesData' type='Column' xName='month'
yName='sales' name='Sales'>
</e-chart3d-series>
</e-chart3d-series-collection>
</ejs-chart3d>
</template>
<script>
import { Chart3DComponent, Chart3DSeriesCollectionDirective,
Chart3DSeriesDirective, ColumnSeries3D, Category3D, Legend3D } from
"@syncfusion/ej2-vue-charts";
export default {
name: "App",
components: {
'ejs-chart3d' : Chart3DComponent,
'e-chart3d-series-collection' : Chart3DSeriesCollectionDirective,
'e-chart3d-series' : Chart3DSeriesDirective
},
data() {
return {
primaryXAxis: {
valueType: 'Category'
},
title: 'Sales Analysis',
seriesData: [
{ month: 'Jan', sales: 35 }, { month: 'Feb', sales: 28 },
{ month: 'Mar', sales: 34 }, { month: 'Apr', sales: 32 },
{ month: 'May', sales: 40 }, { month: 'Jun', sales: 32 },
{ month: 'Jul', sales: 35 }, { month: 'Aug', sales: 55 },
{ month: 'Sep', sales: 38 }, { month: 'Oct', sales: 30 },

```

```
{ month: 'Nov', sales: 25 }, { month: 'Dec', sales: 32 }  
]  
};  
,  
provide: {  
  chart3d: [ ColumnSeries3D, Category3D, Legend3D ]  
},  
};  
</script>
```

### Run the project

To run the project, use the following command:

```
`bash
```

```
npm run dev
```

```
`
```

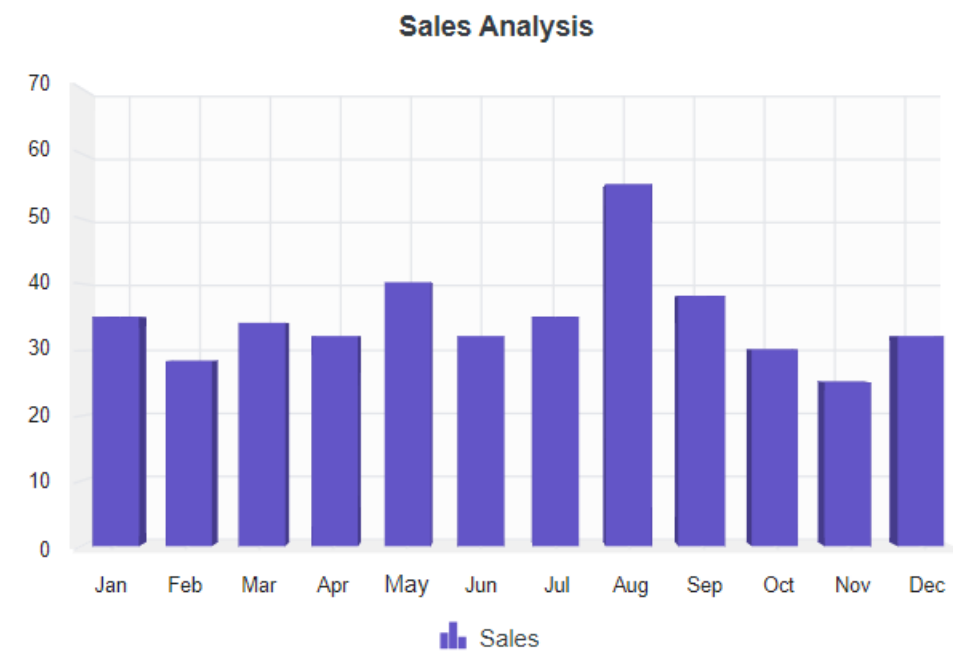
or

```
`bash
```

```
yarn run dev
```

```
`
```

The output will appear as follows:



**Sample:** vue-3-3d-chart-getting-started.

For migrating from Vue 2 to Vue 3, refer to the [migration](#) documentation.

See also

- [Getting Started with Vue UI Components using Composition API and TypeScript](#)

- [Getting Started with Vue UI Components using Options API and TypeScript](#)

## Chart

### Getting Started with the Vue Chart Component in Vue 2

This article provides a step-by-step guide for setting up a Vue 2 project using [Vue-CLI](#) and integrating the Syncfusion Vue Chart component.

#### Prerequisites

##### [System requirements for Syncfusion Vue UI components](#)

#### Dependencies

Below is the list of minimum dependencies required to use the chart component.

```
`javascript
|-- @syncfusion/ej2-vue-charts
|-- @syncfusion/ej2-base
|-- @syncfusion/ej2-data
|-- @syncfusion/ej2-pdf-export
|-- @syncfusion/ej2-file-utils
|-- @syncfusion/ej2-compression
|-- @syncfusion/ej2-charts
|-- @syncfusion/ej2-vue-base
|-- @syncfusion/ej2-svg-base
`,`
```

#### Setting up the Vue 2 project

To generate a Vue 2 project using Vue-CLI, use the [vue create](#) command. Follow these steps to install Vue CLI and create a new project:

```
`bash
npm install -g @vue/cli
vue create quickstart
cd quickstart
npm run serve
`,`
```

or

```
`bash
yarn global add @vue/cli
vue create quickstart
cd quickstart
```

```
yarn run serve
```

When creating a new project, choose the option `Default ([Vue 2] babel, eslint)` from the menu.

```
E:\samples>vue create quickstart

Vue CLI v5.0.8
? Please pick a preset:
  Default ([Vue 3] babel, eslint)
> Default ([Vue 2] babel, eslint)
  Manually select features
```

Once the `quickstart` project is set up with default settings, proceed to add Syncfusion components to the project.

#### Add Syncfusion Vue packages

Syncfusion packages are available at [npmjs.com](https://www.npmjs.com). To use Vue components, install the required npm package.

This article uses the [Vue Charts component](#) as an example. Install the `@syncfusion/ej2-vue-charts` package by running the following command:

```
`bash
```

```
npm install @syncfusion/ej2-vue-charts --save
```

or

```
`bash
```

```
yarn add @syncfusion/ej2-vue-charts
```

The `--save` will instruct NPM to include the chart package inside of the `dependencies` section of the `package.json`.

#### Add Syncfusion Vue component

Follow the below steps to add the Vue Chart component:

1\. First, import and register the Chart component in the `script` section of the `src/App.vue` file.

##### ~/SRC/APP.VUE

```
<script>
import { ChartComponent } from '@syncfusion/ej2-vue-charts';
export default {
  components: {
    'ejs-chart': ChartComponent
  }
}
</script>
```

2\ In the `template` section, define the Chart component.

**~/SRC/APP.VUE**

```
<template>
<div id="app">
<ejs-chart id="container"> </ejs-chart>
</div>
</template>
```

Here is the summarized code for the above steps in the **src/App.vue** file:

**~/SRC/APP.VUE**

```
<template>
  <div id="app">
    <ejs-chart id="container"> </ejs-chart>
  </div>
</template>
<script>
import { ChartComponent } from '@syncfusion/ej2-vue-charts';
export default {
  components: {
    'ejs-chart': ChartComponent
  },
  data () {
    return {
    }
  }
}
</script>
<style>
#container{
  height: 350px;
}
</style>
```

[Run the project](#)

To run the project, use the following command:

```
`bash
```

```
npm run serve
```

```
,
```

or

```
`bash
```

```
yarn run serve
```

```
,
```

```
{% previewsample "page.domainurl/code-snippet/chart/getting-started/initialize-cs1" %}
```

### Module Injection

Chart component are segregated into individual feature-wise modules. In order to use a particular feature,

you need to inject its feature service in the AppModule. In the current application, we are going to modify the above basic chart to visualize sales data for a particular year.

For this application we are going to use line series, tooltip, data label, category axis and legend feature of the chart. Please find relevant feature service name and description as follows.

- **LineSeries** - Inject this provider to use line series.
- **Legend** - Inject this provider to use legend feature.
- **Tooltip** - Inject this provider to use tooltip feature.
- **DataLabel** - Inject this provider to use datalabel feature.
- **Category** - Inject this provider to use category feature.

These modules should be injected to the provide section as follows,

```
`javascript
import { ChartComponent, LineSeries } from "@syncfusion/ej2-vue-charts";
export default {
  components: {
    'ejs-chart': ChartComponent
  },
  provide: {
    chart: [LineSeries]
  }
};
</script>
`
```

### Populate Chart with Data

This section explains how to plot below JSON data to the chart.

```
`javascript
export default {
  data() {
    return {
      seriesData: [
        { month: 'Jan', sales: 35 }, { month: 'Feb', sales: 28 },
```

```

{ month: 'Mar', sales: 34 }, { month: 'Apr', sales: 32 },
{ month: 'May', sales: 40 }, { month: 'Jun', sales: 32 },
{ month: 'Jul', sales: 35 }, { month: 'Aug', sales: 55 },
{ month: 'Sep', sales: 38 }, { month: 'Oct', sales: 30 },
{ month: 'Nov', sales: 25 }, { month: 'Dec', sales: 32 }
]
};
}
};
`

```

- Add a series object to the chart by using [series](#) property. Now map the field names `month` and `sales` in the JSON data to the [xName](#) and

[yName](#) properties of the series, then set the JSON data to [dataSource](#) property.

Since the JSON contains category data, set the [valueType](#) for horizontal axis to `Category`. By default, the axis `valueType` is `Numeric`.

#### ~/SRC/APP.VUE

```

<template>
  <div id="app">
    <ejs-chart id="container" :primaryXAxis='primaryXAxis'>
      <e-series-collection>
        <e-series :dataSource='seriesData' type='Line' xName='month'
yName='sales' name='Sales'> </e-series>
      </e-series-collection>
    </ejs-chart>
  </div>
</template>
<script>
import { ChartComponent, SeriesCollectionDirective, SeriesDirective,
LineSeries, Category } from "@syncfusion/ej2-vue-charts";
export default {
  components: {
    'ejs-chart': ChartComponent,
    'e-series-collection': SeriesCollectionDirective,
    'e-series': SeriesDirective
  },
  data() {
    return {
      seriesData: [
        { month: 'Jan', sales: 35 }, { month: 'Feb', sales: 28 },
        { month: 'Mar', sales: 34 }, { month: 'Apr', sales: 32 },
        { month: 'May', sales: 40 }, { month: 'Jun', sales: 32 },
        { month: 'Jul', sales: 35 }, { month: 'Aug', sales: 55 },
        { month: 'Sep', sales: 38 }, { month: 'Oct', sales: 30 },
        { month: 'Nov', sales: 25 }, { month: 'Dec', sales: 32 }
      ]
    }
  }
}

```

```

        ],
        primaryXAxis: {
          valueType: 'Category'
        }
      };
    },
    provide: {
      chart: [LineSeries, Category]
    }
  };
</script>
<style>
#container {
  height: 350px;
}
</style>

```

{% previewsample "page.domainurl/code-snippet/chart/getting-started/datasource-cs4" %}

- The sales data are in thousands, so format the vertical axis label by adding \$ as a prefix and K as a suffix to each label. This can be achieved by setting the `labelFormat` property of axis. Here, `{value}` act as a placeholder for each axis label.

### Add Chart Title

You can add a title using `title` property to the chart to provide quick information to the user about the data plotted in the chart.

### ~/SRC/APP.VUE

```

<template>
  <div id="app">
    <ejs-chart id="container" :title='title'
:primaryXAxis='primaryXAxis' :primaryYAxis='primaryYAxis'>
      <e-series-collection>
        <e-series :dataSource='seriesData' type='Line' xName='month'
yName='sales' name='Sales'> </e-series>
      </e-series-collection>
    </ejs-chart>
  </div>
</template>
<script>
import { ChartComponent, SeriesDirective, SeriesCollectionDirective,
LineSeries, Category } from "@syncfusion/ej2-vue-charts";
export default {
  components: {
    'ejs-chart': ChartComponent,
    'e-series-collection': SeriesCollectionDirective,
    'e-series': SeriesDirective
  },
  data() {
    return {
      seriesData: [
        { month: 'Jan', sales: 35 }, { month: 'Feb', sales: 28 },
        { month: 'Mar', sales: 34 }, { month: 'Apr', sales: 32 },
        { month: 'May', sales: 40 }, { month: 'Jun', sales: 32 },

```



```

        { month: 'Jul', sales: 35 }, { month: 'Aug', sales: 55 },
        { month: 'Sep', sales: 38 }, { month: 'Oct', sales: 30 },
        { month: 'Nov', sales: 25 }, { month: 'Dec', sales: 32 }
      ],
      primaryXAxis: {
        valueType: 'Category'
      },
      primaryYAxis: {
        labelFormat: '${value}K'
      },
      title: "Sales Analysis"
    };
  },
  provide: {
    chart: [LineSeries, Category]
  }
};
</script>
<style>
  #container{
    height: 350px;
  }
</style>

```

{% previewsample "page.domainurl/code-snippet/chart/getting-started/tooltip-cs1" %}

### Enable Legend

You can use legend for the chart by setting the `visible` property to true in [legendSettings](#) object and by injecting the `Legend` into the `provide`.

### ~/SRC/APP.VUE

```

<template>
  <div id="app">
    <ejs-chart id="container" :title='title'
    :primaryXAxis='primaryXAxis' :primaryYAxis='primaryYAxis'
    :legendSettings='legendSettings'>
      <e-series-collection>
        <e-series :dataSource='seriesData' type='Line' xName='month'
        yName='sales' name='Sales'> </e-series>
      </e-series-collection>
    </ejs-chart>
  </div>
</template>
<script>
import { ChartComponent, SeriesDirective, SeriesCollectionDirective,
LineSeries, Category, Legend } from "@syncfusion/ej2-vue-charts";
export default {
  components: {
    'ejs-chart': ChartComponent,
    'e-series-collection': SeriesCollectionDirective,
    'e-series': SeriesDirective
  },
  data() {
    return {
      seriesData: [

```

```

        { month: 'Jan', sales: 35 }, { month: 'Feb', sales: 28 },
        { month: 'Mar', sales: 34 }, { month: 'Apr', sales: 32 },
        { month: 'May', sales: 40 }, { month: 'Jun', sales: 32 },
        { month: 'Jul', sales: 35 }, { month: 'Aug', sales: 55 },
        { month: 'Sep', sales: 38 }, { month: 'Oct', sales: 30 },
        { month: 'Nov', sales: 25 }, { month: 'Dec', sales: 32 }
      ],
      primaryXAxis: {
        valueType: 'Category'
      },
      primaryYAxis: {
        labelFormat: '${value}K'
      },
      legendSettings: {
        visible: true
      },
      title: "Sales Analysis"
    };
  },
  provide: {
    chart: [LineSeries, Category, Legend]
  }
};
</script>
<style>
  #container {
    height: 350px;
  }
</style>

```

{% previewsample "page.domainurl/code-snippet/chart/getting-started/legend-cs1" %}

### Add Data Label

You can add data labels to improve the readability of the chart. This can be achieved by setting the visible property to true in the `dataLabel` object and by injecting `DataLabel` into the `provide`.

### ~/SRC/APP.VUE

```

<template>
  <div id="app">
    <ejs-chart id="container" :title='title'
    :primaryXAxis='primaryXAxis' :primaryYAxis='primaryYAxis'
    :legendSettings='legendSettings'>
      <e-series-collection>
        <e-series :dataSource='seriesData' type='Line' xName='month'
        yName='sales' name='Sales' :marker='marker'> </e-series>
      </e-series-collection>
    </ejs-chart>
  </div>
</template>
<script>
import { ChartComponent, SeriesDirective, SeriesCollectionDirective,
LineSeries, Category, DataLabel, Legend } from "@syncfusion/ej2-vue-charts";
export default {
  components: {
    'ejs-chart': ChartComponent,

```

```

    'e-series-collection': SeriesCollectionDirective,
    'e-series': SeriesDirective
  },
  data() {
    return {
      seriesData: [
        { month: 'Jan', sales: 35 }, { month: 'Feb', sales: 28 },
        { month: 'Mar', sales: 34 }, { month: 'Apr', sales: 32 },
        { month: 'May', sales: 40 }, { month: 'Jun', sales: 32 },
        { month: 'Jul', sales: 35 }, { month: 'Aug', sales: 55 },
        { month: 'Sep', sales: 38 }, { month: 'Oct', sales: 30 },
        { month: 'Nov', sales: 25 }, { month: 'Dec', sales: 32 }
      ],
      primaryXAxis: {
        valueType: 'Category'
      },
      primaryYAxis: {
        labelFormat: '${value}K'
      },
      legendSettings: {
        visible: true
      },
      marker: {
        dataLabel: {
          visible: true
        }
      },
      title: "Sales Analysis"
    };
  },
  provide: {
    chart: [LineSeries, Category, DataLabel, Legend]
  }
};
</script>
<style>
  #container {
    height: 350px;
  }
</style>

```

{% previewsample "page.domainurl/code-snippet/chart/getting-started/datalabel-cs1" %}

### Enable Tooltip

The tooltip is useful when you cannot display information by using the data labels due to space constraints. You can enable tooltip by setting the enable property as true in [tooltip](#) object and by injecting **Tooltip** into the **provide**.

### ~/SRC/APP.VUE

```

<template>
  <div id="app">
    <ejs-chart id="container" :title='title'
      :primaryXAxis='primaryXAxis' :primaryYAxis='primaryYAxis'
      :tooltip='tooltip'>
      <e-series-collection>

```

```

        <e-series :dataSource='seriesData' type='Line' xName='month'
yName='sales' name='Sales' :marker='marker'> </e-series>
    </e-series-collection>
</ejs-chart>
</div>
</template>
<script>
import { ChartComponent, SeriesDirective, SeriesCollectionDirective,
LineSeries, Category, DataLabel, Tooltip } from "@syncfusion/ej2-vue-
charts";
export default {
  components: {
    'ejs-chart': ChartComponent,
    'e-series-collection': SeriesCollectionDirective,
    'e-series': SeriesDirective
  },
  data() {
    return {
      seriesData: [
        { month: 'Jan', sales: 35 }, { month: 'Feb', sales: 28 },
        { month: 'Mar', sales: 34 }, { month: 'Apr', sales: 32 },
        { month: 'May', sales: 40 }, { month: 'Jun', sales: 32 },
        { month: 'Jul', sales: 35 }, { month: 'Aug', sales: 55 },
        { month: 'Sep', sales: 38 }, { month: 'Oct', sales: 30 },
        { month: 'Nov', sales: 25 }, { month: 'Dec', sales: 32 }
      ],
      primaryXAxis: {
        valueType: 'Category'
      },
      primaryYAxis:{
        labelFormat: '${value}K'
      },
      marker: {
        dataLabel:{
          visible: true
        }
      },
      tooltip:{ enable: true },
      title: "Sales Analysis"
    };
  },
  provide: {
    chart: [LineSeries, Category, DataLabel, Tooltip]
  }
};
</script>
<style>
#container {
  height: 350px;
}
</style>

```

{% previewsample "page.domainurl/code-snippet/chart/getting-started/tooltip-cs2" %}

You can refer to our [Vue Charts](#) feature tour page for its groundbreaking feature representations. You can also explore our [Vue Charts example](#) that shows various chart types and how to represent time-dependent data, showing trends in data at equal intervals.

### Getting Started with the Vue Chart Component in Vue 3

This article provides a step-by-step guide for setting up a [Vite](#) project with a JavaScript environment and integrating the Syncfusion Vue Chart component using the [Composition API](#) / [Options API](#).

The **Composition API** is a new feature introduced in Vue.js 3 that provides an alternative way to organize and reuse component logic. It allows developers to write components as functions that use smaller, reusable functions called composition functions to manage their properties and behavior.

The **Options API** is the traditional way of writing Vue.js components, where the component logic is organized into a series of options that define the component's properties and behavior. These options include data, methods, computed properties, watchers, lifecycle hooks, and more.

#### Prerequisites

##### [System requirements for Syncfusion Vue UI components](#)

#### Set up the Vite project

A recommended approach for beginning with Vue is to scaffold a project using [Vite](#). To create a new Vite project, use one of the commands that are specific to either NPM or Yarn.

```
`bash
```

```
npm create vite@latest
```

```
,
```

or

```
`bash
```

```
yarn create vite
```

```
,
```

Using one of the above commands will lead you to set up additional configurations for the project as below:

1. Define the project name: We can specify the name of the project directly. Let's specify the name of the project as **my-project** for this article.

```
`bash
```

```
? Project name: » my-project
```

```
,
```

2. Select **Vue** as the framework. It will create a Vue 3 project.

```
`bash
```

```
? Select a framework: » - Use arrow-keys. Return to submit.
```

```
Vanilla
```

```
Vue
```

```
React
```

Preact

Lit

Svelte

Others

,

3. Choose **JavaScript** as the framework variant to build this Vite project using JavaScript and Vue.

```
`bash
```

? Select a variant: » - Use arrow-keys. Return to submit.

JavaScript

TypeScript

Customize with create-vue ↗

Nuxt ↗

,

4. Upon completing the aforementioned steps to create the **my-project**, run the following command to install its dependencies:

```
`bash
```

```
cd my-project
```

```
npm install
```

,

or

```
`bash
```

```
cd my-project
```

```
yarn install
```

,

Now that **my-project** is ready to run with default settings, let's add Syncfusion components to the project.

#### Add Syncfusion Vue packages

Syncfusion Vue component packages are available at [npmjs.com](https://www.npmjs.com). To use Syncfusion Vue components in the project, install the corresponding npm package.

This article uses the [Vue Chart component](#) as an example. To use the Vue Chart component in the project, the **@syncfusion/ej2-vue-charts** package needs to be installed using the following command:

```
`bash
```

```
npm install @syncfusion/ej2-vue-charts --save
```

,

or

```
`bash
```

```
yarn add @syncfusion/ej2-vue-charts
```

```
,
```

### Add Syncfusion Vue component

Follow the below steps to add the Vue Chart component using **Composition API** or **Options API**:

1.First, import and register the Chart component and its child directives in the **script** section of the **src/App.vue** file. If you are using the **Composition API**, you should add the **setup** attribute to the **script** tag to indicate that Vue will be using the **Composition API**.

#### **COMPOSITION API (~SRC/APP.VUE)**

```
<script setup>
import { ChartComponent as EjsChart, SeriesCollectionDirective as
ESeriesCollection, SeriesDirective as ESeries, LineSeries, Legend, Category
} from "@syncfusion/ej2-vue-charts";
</script>
```

#### **OPTIONS API (~SRC/APP.VUE)**

```
<script>
import { ChartComponent, SeriesCollectionDirective, SeriesDirective,
LineSeries, Legend, Category } from '@syncfusion/ej2-vue-charts';
//Component registration
export default {
  name: "App",
  components: {
    'ejs-chart' : ChartComponent,
    'e-series-collection' : SeriesCollectionDirective,
    'e-series' : SeriesDirective
  }
}
</script>
```

2.In the **template** section, define the Chart component with the [dataSource](#) property.

#### **~/SRC/APP.VUE**

```
<template>
<ejs-chart id="container" :title='title' :primaryXAxis='primaryXAxis'>
<e-series-collection>
<e-series :dataSource='seriesData' type='Line' xName='month' yName='sales'
name='Sales'> </e-series>
</e-series-collection>
</ejs-chart>
</template>
```

3.Declare the values for the **dataSource** property in the **script** section.

#### **COMPOSITION API (~SRC/APP.VUE)**

```
<script setup>
const seriesData = [
  { month: 'Jan', sales: 35 }, { month: 'Feb', sales: 28 },
  { month: 'Mar', sales: 34 }, { month: 'Apr', sales: 32 },
  { month: 'May', sales: 40 }, { month: 'Jun', sales: 32 },
  { month: 'Jul', sales: 35 }, { month: 'Aug', sales: 55 },
  { month: 'Sep', sales: 38 }, { month: 'Oct', sales: 30 },
  { month: 'Nov', sales: 25 }, { month: 'Dec', sales: 32 }
];
</script>
```

### OPTIONS API (~SRC/APP.VUE)

```
<script>
data() {
  return {
    seriesData: [
      { month: 'Jan', sales: 35 }, { month: 'Feb', sales: 28 },
      { month: 'Mar', sales: 34 }, { month: 'Apr', sales: 32 },
      { month: 'May', sales: 40 }, { month: 'Jun', sales: 32 },
      { month: 'Jul', sales: 35 }, { month: 'Aug', sales: 55 },
      { month: 'Sep', sales: 38 }, { month: 'Oct', sales: 30 },
      { month: 'Nov', sales: 25 }, { month: 'Dec', sales: 32 }
    ]
  };
}
</script>
```

Here is the summarized code for the above steps in the **src/App.vue** file:

### COMPOSITION API (~SRC/APP.VUE)

```
<template>
<ejs-chart id="container" :title='title' :primaryXAxis='primaryXAxis'>
  <e-series-collection>
    <e-series :dataSource='seriesData' type='Line' xName='month' yName='sales'
    name='Sales'> </e-series>
  </e-series-collection>
</ejs-chart>
</template>
<script setup>
import { provide } from 'vue';
import { ChartComponent as EjsChart, SeriesCollectionDirective as
ESeriesCollection, SeriesDirective as ESeries, LineSeries, Legend, Category
} from "@syncfusion/ej2-vue-charts";
const seriesData = [
  { month: 'Jan', sales: 35 }, { month: 'Feb', sales: 28 },
  { month: 'Mar', sales: 34 }, { month: 'Apr', sales: 32 },
  { month: 'May', sales: 40 }, { month: 'Jun', sales: 32 },
  { month: 'Jul', sales: 35 }, { month: 'Aug', sales: 55 },
  { month: 'Sep', sales: 38 }, { month: 'Oct', sales: 30 },
  { month: 'Nov', sales: 25 }, { month: 'Dec', sales: 32 }
];
const title = 'Sales Analysis';
const primaryXAxis = {valueType: 'Category'};
const chart = [LineSeries, Legend, Category];
```



```
provide('chart', chart);  
</script>
```

### OPTIONS API (~SRC/APP.VUE)

```
<template>  
<ejs-chart id="container" :title='title' :primaryXAxis='primaryXAxis'>  
<e-series-collection>  
<e-series :dataSource='seriesData' type='Line' xName='month' yName='sales'  
name='Sales'> </e-series>  
</e-series-collection>  
</ejs-chart>  
</template>  
<script>  
import { ChartComponent, SeriesCollectionDirective, SeriesDirective,  
LineSeries, Legend, Category } from "@syncfusion/ej2-vue-charts";  
export default {  
  name: "App",  
  components: {  
    'ejs-chart' : ChartComponent,  
    'e-series-collection' : SeriesCollectionDirective,  
    'e-series' : SeriesDirective  
  },  
  data() {  
    return {  
      primaryXAxis: {  
        valueType: 'Category'  
      },  
      title: 'Sales Analysis',  
      seriesData: [  
        { month: 'Jan', sales: 35 }, { month: 'Feb', sales: 28 },  
        { month: 'Mar', sales: 34 }, { month: 'Apr', sales: 32 },  
        { month: 'May', sales: 40 }, { month: 'Jun', sales: 32 },  
        { month: 'Jul', sales: 35 }, { month: 'Aug', sales: 55 },  
        { month: 'Sep', sales: 38 }, { month: 'Oct', sales: 30 },  
        { month: 'Nov', sales: 25 }, { month: 'Dec', sales: 32 }  
      ]  
    };  
  },  
  provide: {  
    chart: [ LineSeries, Legend, Category ]  
  },  
};  
</script>
```

### Run the project

To run the project, use the following command:

```
`bash
```

```
npm run dev
```

```
`
```

or

```
`bash
```

```
yarn run dev
```

```
,
```

The output will appear as follows:



**Sample:** [vue-3-chart-getting-started](#).

For migrating from Vue 2 to Vue 3, refer to the [migration](#) documentation.

See also

- [Getting Started with Vue UI Components using Composition API and TypeScript](#)
- [Getting Started with Vue UI Components using Options API and TypeScript](#)

## Working with data in Vue Chart component

Chart can visualise data bound from local or remote data.

### Local Data

You can bind a simple JSON data to the chart using [dataSource](#) property in series. Now map the fields in JSON to [xName](#) and [yName](#) properties.

### APP.VUE

```
<template>
  <div id="app">
    <ejs-chart id="container" :title='title'
:primaryXAxis='primaryXAxis'>
      <e-series-collection>
        <e-series :dataSource='seriesData' type='Column'
xName='month' yName='sales' name='Sales'> </e-series>
      </e-series-collection>
    </ejs-chart>
  </div>
</template>
<script>
import Vue from "vue";
import { ChartPlugin, ColumnSeries, Category } from "@syncfusion/ej2-vue-
charts";
Vue.use(ChartPlugin);
export default {
  data() {
    return {
```

```

seriesData: [
  { month: 'Jan', sales: 35 }, { month: 'Feb', sales: 28 },
  { month: 'Mar', sales: 34 }, { month: 'Apr', sales: 32 },
  { month: 'May', sales: 40 }, { month: 'Jun', sales: 32 },
  { month: 'Jul', sales: 35 }, { month: 'Aug', sales: 55 },
  { month: 'Sep', sales: 38 }, { month: 'Oct', sales: 30 },
  { month: 'Nov', sales: 25 }, { month: 'Dec', sales: 32 }
],
primaryXAxis: {
  valueType: 'Category'
},
title: "Olympic Medals"
};
},
provide: {
  chart: [ColumnSeries, Category]
}
};
</script>
<style>
  #container {
    height: 350px;
  }
</style>

```

{% previewsample "page.domainurl/code-snippet/chart/series/column-cs19" %}

### Common Datasource

You can also bind a JSON data common to all series using [dataSource](#) property in chart.

### APP.VUE

```

<template>
  <div id="app">
    <ejs-chart id="container" :title='title'
:primaryXAxis='primaryXAxis' :dataSource='seriesData'>
      <e-series-collection>
        <e-series type='Column' xName='month' yName='sales' > </e-
series>
        <e-series type='Column' xName='month' yName='sales1' > </e-
series>
      </e-series-collection>
    </ejs-chart>
  </div>
</template>
<script>
import Vue from "vue";
import { ChartPlugin, ColumnSeries, Category } from "@syncfusion/ej2-vue-
charts";
Vue.use(ChartPlugin);
export default {
  data() {
    return {
      seriesData: [
        { month: 'Jan', sales: 35, sales1: 28 }, { month: 'Feb', sales: 28,
sales1: 35 },

```

```

    { month: 'Mar', sales: 34, sales1: 32 }, { month: 'Apr', sales: 32,
    sales1: 34 },
    { month: 'May', sales: 40, sales1: 32 }, { month: 'Jun', sales: 32,
    sales1: 40 },
    { month: 'Jul', sales: 35, sales1: 55 }, { month: 'Aug', sales: 55,
    sales1: 35 },
    { month: 'Sep', sales: 38, sales1: 30 }, { month: 'Oct', sales: 30,
    sales1: 38 },
    { month: 'Nov', sales: 25, sales1: 32 }, { month: 'Dec', sales: 32,
    sales1: 25 }
    ],
    primaryXAxis: {
      valueType: 'Category'
    },
    title: "Olympic Medals"
  };
},
provide: {
  chart: [ColumnSeries, Category]
}
};
</script>
<style>
  #container {
    height: 350px;
  }
</style>

```

{% previewsample "page.domainurl/code-snippet/chart/series/column-cs21" %}

### Remote Data

You can also bind remote data to the chart using **DataManager**. The DataManager requires minimal information

like webservice URL, adaptor and crossDomain to interact with service endpoint properly. Assign the instance

of DataManager to the [dataSource](#) property in series and map the fields of data to [xName](#) and [yName](#) properties. You can also use the [query](#) property of the series to filter the data.

### APP.VUE

```

<template>
  <div id="app">
    <ejs-chart id="container" :primaryXAxis='primaryXAxis'
    :primaryYAxis='primaryYAxis' :title='title'>
      <e-series-collection>
        <e-series :dataSource='seriesData' type='Column'
        xName='CustomerID' yName='Freight' :query='queries'> </e-series>
      </e-series-collection>
    </ejs-chart>
  </div>
</template>
<script>
import Vue from "vue";

```

```

import { ChartPlugin, ColumnSeries, Category } from "@syncfusion/ej2-vue-charts";
import { DataManager, Query } from '@syncfusion/ej2-data';
Vue.use(ChartPlugin);
let dataManager = new DataManager({
  url: 'https://services.syncfusion.com/vue/production/api/orders'
});
let query = new Query().take(5).where('Estimate', 'lessThan', 3, false);
export default {
  data() {
    return {
      seriesData: dataManager,
      queries: query,
      primaryXAxis: {
        valueType: 'Category',
      },
      primaryYAxis: {
        title: 'Freight rate in U.S. dollars'
      },
      title: 'Container freight rate'
    };
  },
  provide: {
    chart: [ColumnSeries, Category]
  }
};
</script>
<style>
#container {
  height: 350px;
}
</style>

```

{% previewsample "page.domainurl/code-snippet/chart/series/column-cs22" %}

### Empty points

The Data points that uses the `null` or `undefined` as value are considered as empty points. Empty data points are ignored and not plotted in the Chart. When the data is provided by using the points property, By using `emptyPointSettings` property in series, you can customize the empty point. Default `mode` of the empty point is `Gap`.

### APP.VUE

```

<template>
  <div id="app">
    <ejs-chart id="container" :title='title'
    :primaryXAxis='primaryXAxis'>
      <e-series-collection>
        <e-series :dataSource='seriesData' type='Column'
        xName='month' yName='sales' name='Sales'
        :emptyPointSettings='emptyPointSettings1'></e-series>
        <e-series :dataSource='seriesData' type='Line' xName='month'
        yName='sales' name='Sales Compare'
        :emptyPointSettings='emptyPointSettings2'></e-series>
      </e-series-collection>
    </div>
  </template>

```

```

        </ejs-chart>
      </div>
    </template>
    <script>
    import Vue from "vue";
    import { ChartPlugin, ColumnSeries, LineSeries, Category } from
"@syncfusion/ej2-vue-charts";
    Vue.use(ChartPlugin);
    export default {
      data() {
        return {
          seriesData: [
            { month: 'Jan', sales: 35 }, { month: 'Feb', sales: null },
            { month: 'Mar', sales: 34 }, { month: 'Apr', sales: 32 },
            { month: 'May', sales: 40 }, { month: 'Jun', sales: 32 },
            { month: 'Jul', sales: undefined }, { month: 'Aug', sales: 55 },
            { month: 'Sep', sales: 38 }, { month: 'Oct', sales: 30 },
            { month: 'Nov', sales: 25 }, { month: 'Dec', sales: 32 }
          ],
          primaryXAxis: {
            valueType: 'Category'
          },
          emptyPointSettings1: {
            mode: "Average",
            fill: 'grey'
          },
          emptyPointSettings2: {
            mode: "Gap"
          },
          title: "Olympic Medals"
        };
      },
      provide: {
        chart: [ColumnSeries, LineSeries, Category]
      }
    };
    </script>
    <style>
    #container {
      height: 350px;
    }
    </style>
  
```

{% previewsample "page.domainurl/code-snippet/chart/axis/category-cs43" %}

- Customizing empty point

Specific color for empty point can be set by **fill** property in **emptyPointSettings**. Border for a empty point can be set by **border** property.

#### APP.VUE

```

<template>
  <div id="app">

```

```

        <ejs-chart id="container" :title='title'
:primaryXAxis='primaryXAxis'>
            <e-series-collection>
                <e-series :dataSource='seriesData' type='Column'
xName='month' yName='sales' name='Sales'
:emptyPointSettings='emptyPointSettings'></e-series>
            </e-series-collection>
        </ejs-chart>
    </div>
</template>
<script>
import Vue from "vue";
import { ChartPlugin, ColumnSeries, LineSeries, Category } from
"@syncfusion/ej2-vue-charts";
Vue.use(ChartPlugin);
export default {
    data() {
        return {
            seriesData: [
                { month: 'Jan', sales: 35 }, { month: 'Feb', sales: null },
                { month: 'Mar', sales: 34 }, { month: 'Apr', sales: 32 },
                { month: 'May', sales: 40 }, { month: 'Jun', sales: 32 },
                { month: 'Jul', sales: undefined }, { month: 'Aug', sales: 55 },
                { month: 'Sep', sales: 38 }, { month: 'Oct', sales: 30 },
                { month: 'Nov', sales: 25 }, { month: 'Dec', sales: 32 }
            ],
            primaryXAxis: {
                valueType: 'Category'
            },
            emptyPointSettings: {
                mode: "Average",
                fill: 'red',
                border: { width: 2, color: 'violet' }
            },
            title: "Olympic Medals"
        };
    },
    provide: {
        chart: [ColumnSeries, LineSeries, Category]
    }
};
</script>
<style>
    #container {
        height: 350px;
    }
</style>

```

{% previewsample "page.domainurl/code-snippet/chart/axis/category-cs44" %}

## Chart dimensions in Vue Chart component

### Size for Container

Chart can render to its container size. You can set the size via inline or CSS as demonstrated below.

### APP.VUE

```

<template>
  <div id="app">
    <ejs-chart id="container" width='650px' height='350px'
:primaryXAxis='primaryXAxis'>
      <e-series-collection>
        <e-series :dataSource='seriesData' type='Line' xName='month'
yName='sales' name='Sales'> </e-series>
      </e-series-collection>
    </ejs-chart>
  </div>
</template>
<script>
import Vue from "vue";
import { ChartPlugin, LineSeries, Category } from "@syncfusion/ej2-vue-
charts";
Vue.use(ChartPlugin);
export default {
  data() {
    return {
      seriesData: [
        { month: 'Jan', sales: 35 }, { month: 'Feb', sales: 28 },
        { month: 'Mar', sales: 34 }, { month: 'Apr', sales: 32 },
        { month: 'May', sales: 40 }, { month: 'Jun', sales: 32 },
        { month: 'Jul', sales: 35 }, { month: 'Aug', sales: 55 },
        { month: 'Sep', sales: 38 }, { month: 'Oct', sales: 30 },
        { month: 'Nov', sales: 25 }, { month: 'Dec', sales: 32 }
      ],
      primaryXAxis: {
        valueType: 'Category'
      }
    };
  },
  provide: {
    chart: [LineSeries, Category]
  }
};
</script>
<style>
  #container {
    height: 350px;
  }
</style>

```

{% previewsample "page.domainurl/code-snippet/chart/getting-started/datasource-cs1" %}

### Size for Chart

You can also set size for chart directly through [width](#) and [height](#) properties.

<!-- markdownlint-disable MD036 -->

- In Pixel

<!-- markdownlint-disable MD036 -->

You can set the size of chart in pixel as demonstrated below.



**APP.VUE**

```

<template>
  <div id="app">
    <ejs-chart id="container" width='650px' height='350px'
:primaryXAxis='primaryXAxis'>
      <e-series-collection>
        <e-series :dataSource='seriesData' type='Line' xName='month'
yName='sales' name='Sales'> </e-series>
      </e-series-collection>
    </ejs-chart>
  </div>
</template>
<script>
import Vue from "vue";
import { ChartPlugin, LineSeries, Category } from "@syncfusion/ej2-vue-
charts";
Vue.use(ChartPlugin);
export default {
  data() {
    return {
      seriesData: [
        { month: 'Jan', sales: 35 }, { month: 'Feb', sales: 28 },
        { month: 'Mar', sales: 34 }, { month: 'Apr', sales: 32 },
        { month: 'May', sales: 40 }, { month: 'Jun', sales: 32 },
        { month: 'Jul', sales: 35 }, { month: 'Aug', sales: 55 },
        { month: 'Sep', sales: 38 }, { month: 'Oct', sales: 30 },
        { month: 'Nov', sales: 25 }, { month: 'Dec', sales: 32 }
      ],
      primaryXAxis: {
        valueType: 'Category'
      }
    };
  },
  provide: {
    chart: [LineSeries, Category]
  }
};
</script>
<style>
#container {
  height: 350px;
}
</style>

```

{% previewsample "page.domainurl/code-snippet/chart/getting-started/datasource-cs2" %}

- In Percentage

By setting value in percentage, chart gets its dimension with respect to its container. For example, when the height is '50%', chart renders to half of the container height.

**APP.VUE**

```

<template>
  <div id="app">

```

```

        <ejs-chart id="container" :primaryXAxis='primaryXAxis' width='80%'
height='90%'>
            <e-series-collection>
                <e-series :dataSource='seriesData' type='Line' xName='month'
yName='sales' name='Sales'> </e-series>
            </e-series-collection>
        </ejs-chart>
    </div>
</template>
<script>
import Vue from "vue";
import { ChartPlugin, LineSeries, Category } from "@syncfusion/ej2-vue-
charts";
Vue.use(ChartPlugin);
export default {
    data() {
        return {
            seriesData: [
                { month: 'Jan', sales: 35 }, { month: 'Feb', sales: 28 },
                { month: 'Mar', sales: 34 }, { month: 'Apr', sales: 32 },
                { month: 'May', sales: 40 }, { month: 'Jun', sales: 32 },
                { month: 'Jul', sales: 35 }, { month: 'Aug', sales: 55 },
                { month: 'Sep', sales: 38 }, { month: 'Oct', sales: 30 },
                { month: 'Nov', sales: 25 }, { month: 'Dec', sales: 32 }
            ],
            primaryXAxis: {
                valueType: 'Category'
            }
        };
    },
    provide: {
        chart: [LineSeries, Category]
    }
};
</script>
<style>
    #container {
        height: 350px;
    }
</style>

```

{% previewsample "page.domainurl/code-snippet/chart/getting-started/datasource-cs3" %}

Note: When you do not specify the size, it takes 450px as the height and window size as its width.

### Category axis in Vue Chart component

<!-- markdownlint-disable MD036 -->

Category axis are used to represent, the string values instead of numbers.

#### APP.VUE

```

<template>
    <div id="app">
        <ejs-chart id="container" :title='title'
:primaryXAxis='primaryXAxis' :primaryYAxis='primaryYAxis'>
            <e-series-collection>

```

```

        <e-series :dataSource='seriesData' type='Column'
        xName='country' yName='gold' name='Gold'> </e-series>
      </e-series-collection>
    </ejs-chart>
  </div>
</template>
<script>
import Vue from "vue";
import { ChartPlugin, ColumnSeries, Category } from "@syncfusion/ej2-vue-
charts";
Vue.use(ChartPlugin);
export default {
  data() {
    return {
      seriesData: [
        { country: "USA", gold: 50 },
        { country: "China", gold: 40 },
        { country: "Japan", gold: 70 },
        { country: "Australia", gold: 60 },
        { country: "France", gold: 50 },
        { country: "Germany", gold: 40 },
        { country: "Italy", gold: 40 },
        { country: "Sweden", gold: 30 }
      ],
      primaryXAxis: {
        valueType: 'Category',
        title: 'Countries'
      },
      primaryYAxis:
      {
        minimum: 0, maximum: 80,
        interval: 20, title: 'Medals'
      },
      title: "Olympic Medals"
    };
  },
  provide: {
    chart: [ColumnSeries, Category]
  }
};
</script>
<style>
#container {
  height: 350px;
}
</style>

```

{% previewsample "page.domainurl/code-snippet/chart/axis/category-cs20" %}

Note: To use category axis, we need to inject **Category** into the **provide** and set the [valueType](#) of axis to **Category**.

<!-- markdownlint-disable MD036 -->

[Labels Placement](#)

<!-- markdownlint-disable MD036 -->

By default, category labels are placed between the ticks in an axis, this can also be placed on ticks using [labelPlacement](#) property.

#### APP.VUE

```
<template>
  <div id="app">
    <ejs-chart id="container" :title='title'
:primaryXAxis='primaryXAxis'>
      <e-series-collection>
        <e-series :dataSource='seriesData' type='Column'
xName='country' yName='gold' name='Gold'> </e-series>
      </e-series-collection>
    </ejs-chart>
  </div>
</template>
<script>
import Vue from "vue";
import { ChartPlugin, ColumnSeries, Category } from "@syncfusion/ej2-vue-
charts";
Vue.use(ChartPlugin);
export default {
  data() {
    return {
      seriesData: [
        { country: "USA", gold: 50 },
        { country: "China", gold: 40 },
        { country: "Japan", gold: 70 },
        { country: "Australia", gold: 60 },
        { country: "France", gold: 50 },
        { country: "Germany", gold: 40 },
        { country: "Italy", gold: 40 },
        { country: "Sweden", gold: 30 }
      ],
      primaryXAxis: {
        valueType: 'Category',
        title: 'Countries',
        labelPlacement: 'OnTicks'
      },
      title: "Olympic Medals"
    };
  },
  provide: {
    chart: [ColumnSeries, Category]
  },
};
</script>
<style>
#container {
  height: 350px;
}
</style>
```

{% previewsample "page.domainurl/code-snippet/chart/axis/category-cs21" %}

## Range

Range of the category axis can be customized using `minimum`, `maximum` and `interval` property of the axis.

### APP.VUE

```
<template>
  <div id="app">
    <ejs-chart id="container" :title='title'
:primaryXAxis='primaryXAxis'>
      <e-series-collection>
        <e-series :dataSource='seriesData' type='Column'
xName='country' yName='gold' name='Gold'> </e-series>
      </e-series-collection>
    </ejs-chart>
  </div>
</template>
<script>
import Vue from "vue";
import { ChartPlugin, ColumnSeries, Category } from "@syncfusion/ej2-vue-
charts";
Vue.use(ChartPlugin);
export default {
  data() {
    return {
      seriesData: [
        { country: "USA", gold: 50 },
        { country: "China", gold: 40 },
        { country: "Japan", gold: 70 },
        { country: "Australia", gold: 60 },
        { country: "France", gold: 50 },
        { country: "Germany", gold: 40 },
        { country: "Italy", gold: 40 },
        { country: "Sweden", gold: 30 }
      ],
      primaryXAxis: {
        valueType: 'Category',
        title: 'Countries',
        minimum: 1,
        maximum: 5,
        interval: 2
      },
      title: "Olympic Medals"
    };
  },
  provide: {
    chart: [ColumnSeries, Category]
  },
};
</script>
<style>
  #container {
    height: 350px;
  }
</style>
```

{% previewsample "page.domainurl/code-snippet/chart/axis/category-cs22" %}

### Indexed category axis

Category axis also can be rendered based on the index values of data source. This can be achieved by defining the `isIndexed` property to `true` in the axis.

#### APP.VUE

```
<template>
  <div id="app">
    <ejs-chart id="container" :title='title'
:primaryXAxis='primaryXAxis'>
      <e-series-collection>
        <e-series :dataSource='seriesData1' type='Column' xName='x'
yName='y'> </e-series>
        <e-series :dataSource='seriesData2' type='Column' xName='x'
yName='y'> </e-series>
      </e-series-collection>
    </ejs-chart>
  </div>
</template>
<script>
import Vue from "vue";
import { ChartPlugin, ColumnSeries, Category } from "@syncfusion/ej2-vue-
charts";
Vue.use(ChartPlugin);
export default {
  data() {
    return {
      seriesData1: [
        { x: 'Myanmar', y: 7.3 }, { x: 'India', y: 7.9 }, { x: 'Bangladesh',
y: 6.8 },
        { x: 'Cambodia', y: 7.0 }, { x: 'China', y: 6.9 }
      ],
      seriesData2: [
        { x: 'Poland', y: 2.7 }, { x: 'Australia', y: 2.5 }, { x: 'Singapore',
y: 2.0 },
        { x: 'Canada', y: 1.4 }, { x: 'Germany', y: 1.8 }
      ],
      primaryXAxis: {
        valueType: 'Category',
        isIndexed: true
      },
      title: "Olympic Medals"
    };
  },
  provide: {
    chart: [ColumnSeries, Category]
  },
};
</script>
<style>
#container {
  height: 350px;
}
</style>
```

```
{% previewsample "page.domainurl/code-snippet/chart/axis/category-cs23" %}
```

```
<!-- markdownlint-disable MD036 -->
```

## Numeric axis in Vue Chart component

You can use numeric axis to represent numeric values of data in chart. By default, the `valueType` of an axis is `Double`.

### APP.VUE

```
<template>
  <div id="app">
    <ejs-chart id="container" :title='title'>
      <e-series-collection>
        <e-series :dataSource='seriesData' type='Area' xName='x'
yName='y' name='England'> </e-series>
      </e-series-collection>
    </ejs-chart>
  </div>
</template>
<script>
import Vue from "vue";
import { ChartPlugin, AreaSeries } from "@syncfusion/ej2-vue-charts";
Vue.use(ChartPlugin);
export default {
  data() {
    return {
      seriesData:[
        { x: 1, y: 7 }, { x: 2, y: 1 }, { x: 3, y: 1 }, { x: 4, y: 14 }, { x: 5,
y: 1 }, { x: 6, y: 10 },
        { x: 7, y: 8 }, { x: 8, y: 6 }, { x: 9, y: 10 }, { x: 10, y: 10 }, { x:
11, y: 16 }, { x: 12, y: 6 },
        { x: 13, y: 14 }, { x: 14, y: 7 }, { x: 15, y: 5 }, { x: 16, y: 2 }, {
x: 17, y: 14 }, { x: 18, y: 7 },
        { x: 19, y: 7 }, { x: 20, y: 10 }],
      title: "England - Run Rate"
    };
  },
  provide: {
    chart: [AreaSeries]
  },
};
</script>
<style>
  #container {
    height: 350px;
  }
</style>
```

```
{% previewsample "page.domainurl/code-snippet/chart/axis/double-cs1" %}
```

## Range

Range for an axis, will be calculated automatically based on the provided data, you can also customize the range of the axis using [minimum](#), [maximum](#) and [interval](#) property of the axis.

### APP.VUE

```

<template>
  <div id="app">
    <ejs-chart id="container" :title='title'
:primaryXAxis='primaryXAxis'>
      <e-series-collection>
        <e-series :dataSource='seriesData' type='Area' xName='x'
yName='y' name='England'> </e-series>
      </e-series-collection>
    </ejs-chart>
  </div>
</template>
<script>
import Vue from "vue";
import { ChartPlugin, AreaSeries } from "@syncfusion/ej2-vue-charts";
Vue.use(ChartPlugin);
export default {
  data() {
    return {
      seriesData:[
        { x: 1, y: 7 }, { x: 2, y: 1 }, { x: 3, y: 1 }, { x: 4, y: 14 }, { x: 5,
y: 1 }, { x: 6, y: 10 },
        { x: 7, y: 8 }, { x: 8, y: 6 }, { x: 9, y: 10 }, { x: 10, y: 10 }, { x:
11, y: 16 }, { x: 12, y: 6 },
        { x: 13, y: 14 }, { x: 14, y: 7 }, { x: 15, y: 5 }, { x: 16, y: 2 }, {
x: 17, y: 14 }, { x: 18, y: 7 },
        { x: 19, y: 7 }, { x: 20, y: 10 }],
      primaryXAxis: {
        valueType: 'Double',
        minimum: 1,
        maximum: 20,
        interval: 5,
        title: 'Overs'
      },
      title: "England - Run Rate"
    };
  },
  provide: {
    chart: [AreaSeries]
  },
};
</script>
<style>
  #container {
    height: 350px;
  }
</style>

```

{% previewsample "page.domainurl/code-snippet/chart/axis/double-cs2" %}

### Range Padding

Padding can be applied to the minimum and maximum extremes of the axis range by using the [rangePadding](#) property. Numeric axis supports following types of padding.

- Round
- Normal



- Auto

### Numeric - None

When the [rangePadding](#) is set to **None**, minimum and maximum of an axis is based on the data.

### APP.VUE

```
<template>
  <div id="app">
    <ejs-chart id="container" :title='title'
:primaryXAxis='primaryXAxis' :primaryYAxis='primaryYAxis'>
      <e-series-collection>
        <e-series :dataSource='seriesData' type='Area' xName='x'
yName='y' name='England'> </e-series>
      </e-series-collection>
    </ejs-chart>
  </div>
</template>
<script>
import Vue from "vue";
import { ChartPlugin, AreaSeries } from "@syncfusion/ej2-vue-charts";
Vue.use(ChartPlugin);
export default {
  data() {
    return {
      seriesData:[
        { x: 1, y: 7 }, { x: 2, y: 1 }, { x: 3, y: 1 }, { x: 4, y: 14 }, { x: 5,
y: 1 }, { x: 6, y: 10 },
        { x: 7, y: 8 }, { x: 8, y: 6 }, { x: 9, y: 10 }, { x: 10, y: 10 }, { x:
11, y: 16 }, { x: 12, y: 6 },
        { x: 13, y: 14 }, { x: 14, y: 7 }, { x: 15, y: 5 }, { x: 16, y: 2 }, {
x: 17, y: 14 }, { x: 18, y: 7 },
        { x: 19, y: 7 }, { x: 20, y: 10 } ],
      primaryXAxis: {
        valueType: 'Double',
        title: 'Overs'
      },
      primaryYAxis: {
        title: 'Runs',
        valueType: 'Double',
        rangePadding: 'None'
      },
      title: "England - Run Rate"
    };
  },
  provide: {
    chart: [AreaSeries]
  },
};
</script>
<style>
#container {
  height: 350px;
}
</style>
```

```
{% previewsample "page.domainurl/code-snippet/chart/axis/double-cs3" %}
```

### Numeric - Round

When the [rangePadding](#) is set to **Round**, minimum and maximum will be

rounded to the nearest possible value divisible by interval. For example, when the minimum is 3.5 and the interval is 1, then the minimum will be rounded to 3.

### APP.VUE

```
<template>
  <div id="app">
    <ejs-chart id="container" :title='title'
:primaryXAxis='primaryXAxis' :primaryYAxis='primaryYAxis'>
      <e-series-collection>
        <e-series :dataSource='seriesData' type='Area' xName='x'
yName='y' name='England'> </e-series>
      </e-series-collection>
    </ejs-chart>
  </div>
</template>
<script>
import Vue from "vue";
import { ChartPlugin, AreaSeries } from "@syncfusion/ej2-vue-charts";
Vue.use(ChartPlugin);
export default {
  data() {
    return {
      seriesData:[
        { x: 1, y: 7 }, { x: 2, y: 1 }, { x: 3, y: 1 }, { x: 4, y: 14 }, { x: 5,
y: 1 }, { x: 6, y: 10 },
        { x: 7, y: 8 }, { x: 8, y: 6 }, { x: 9, y: 10 }, { x: 10, y: 10 }, { x:
11, y: 16 }, { x: 12, y: 6 },
        { x: 13, y: 14 }, { x: 14, y: 7 }, { x: 15, y: 5 }, { x: 16, y: 2 }, {
x: 17, y: 14 }, { x: 18, y: 7 },
        { x: 19, y: 7 }, { x: 20, y: 10 }],
      primaryXAxis: {
        valueType: 'Double',
        title: 'Overs'
      },
      primaryYAxis: {
        title: 'Runs',
        valueType: 'Double',
        rangePadding: 'Round'
      },
      title: "England - Run Rate"
    };
  },
  provide: {
    chart: [AreaSeries]
  },
};
</script>
<style>
  #container {
    height: 350px;
  }
</style>
```

```
</style>
```

```
{% previewsample "page.domainurl/code-snippet/chart/axis/double-cs4" %}
```

### Numeric - Additional

When the [rangePadding](#) is set to **Additional**, interval of an axis will be padded to the minimum and maximum of the axis.

### APP.VUE

```
<template>
  <div id="app">
    <ejs-chart id="container" :title='title'
:primaryXAxis='primaryXAxis' :primaryYAxis='primaryYAxis'>
      <e-series-collection>
        <e-series :dataSource='seriesData' type='Area' xName='x'
yName='y' name='England'> </e-series>
      </e-series-collection>
    </ejs-chart>
  </div>
</template>
<script>
import Vue from "vue";
import { ChartPlugin, AreaSeries } from "@syncfusion/ej2-vue-charts";
Vue.use(ChartPlugin);
export default {
  data() {
    return {
      seriesData:[
        { x: 1, y: 7 }, { x: 2, y: 1 }, { x: 3, y: 1 }, { x: 4, y: 14 }, { x: 5,
y: 1 }, { x: 6, y: 10 },
        { x: 7, y: 8 }, { x: 8, y: 6 }, { x: 9, y: 10 }, { x: 10, y: 10 }, { x:
11, y: 16 }, { x: 12, y: 6 },
        { x: 13, y: 14 }, { x: 14, y: 7 }, { x: 15, y: 5 }, { x: 16, y: 2 }, {
x: 17, y: 14 }, { x: 18, y: 7 },
        { x: 19, y: 7 }, { x: 20, y: 10 }],
      primaryXAxis: {
        valueType: 'Double',
        title: 'Overs'
      },
      primaryYAxis: {
        title: 'Runs',
        valueType: 'Double',
        rangePadding: 'Additional'
      },
      title: "England - Run Rate"
    };
  },
  provide: {
    chart: [AreaSeries]
  },
};
</script>
<style>
#container {
```

```

    height: 350px;
  }
</style>

```

{% previewsample "page.domainurl/code-snippet/chart/axis/double-cs5" %}

### Numeric - Normal

When the [rangePadding](#) is set to **Normal**, padding is applied to the axis based on default range calculation.

### APP.VUE

```

<template>
  <div id="app">
    <ejs-chart id="container" :title='title'
:primaryXAxis='primaryXAxis' :primaryYAxis='primaryYAxis'>
      <e-series-collection>
        <e-series :dataSource='seriesData' type='Area' xName='x'
yName='y' name='England'> </e-series>
      </e-series-collection>
    </ejs-chart>
  </div>
</template>
<script>
import Vue from "vue";
import { ChartPlugin, AreaSeries } from "@syncfusion/ej2-vue-charts";
Vue.use(ChartPlugin);
export default {
  data() {
    return {
      seriesData:[
        { x: 1, y: 7 }, { x: 2, y: 1 }, { x: 3, y: 1 }, { x: 4, y: 14 }, { x: 5,
y: 1 }, { x: 6, y: 10 },
        { x: 7, y: 8 }, { x: 8, y: 6 }, { x: 9, y: 10 }, { x: 10, y: 10 }, { x:
11, y: 16 }, { x: 12, y: 6 },
        { x: 13, y: 14 }, { x: 14, y: 7 }, { x: 15, y: 5 }, { x: 16, y: 2 }, {
x: 17, y: 14 }, { x: 18, y: 7 },
        { x: 19, y: 7 }, { x: 20, y: 10 }],
      primaryXAxis: {
        valueType: 'Double',
        title: 'Overs'
      },
      primaryYAxis: {
        title: 'Runs',
        valueType: 'Double',
        rangePadding: 'Normal'
      },
      title: "England - Run Rate"
    };
  },
  provide: {
    chart: [AreaSeries]
  },
};
</script>

```

```
<style>
  #container {
    height: 350px;
  }
</style>
```

```
{% previewsample "page.domainurl/code-snippet/chart/axis/double-cs6" %}
```

### Numeric - Auto

When the [rangePadding](#) is set to **Auto**, horizontal numeric axis takes

None as padding calculation, while the vertical numeric axis takes Normal as padding calculation.

### APP.VUE

```
<template>
  <div id="app">
    <ejs-chart id="container" :title='title'
:primaryXAxis='primaryXAxis' :primaryYAxis='primaryYAxis'>
      <e-series-collection>
        <e-series :dataSource='seriesData' type='Area' xName='x'
yName='y' name='England'> </e-series>
      </e-series-collection>
    </ejs-chart>
  </div>
</template>
<script>
import Vue from "vue";
import { ChartPlugin, AreaSeries } from "@syncfusion/ej2-vue-charts";
Vue.use(ChartPlugin);
export default {
  data() {
    return {
      seriesData:[
        { x: 1, y: 7 }, { x: 2, y: 1 }, { x: 3, y: 1 }, { x: 4, y: 14 }, { x: 5,
y: 1 }, { x: 6, y: 10 },
        { x: 7, y: 8 }, { x: 8, y: 6 }, { x: 9, y: 10 }, { x: 10, y: 10 }, { x:
11, y: 16 }, { x: 12, y: 6 },
        { x: 13, y: 14 }, { x: 14, y: 7 }, { x: 15, y: 5 }, { x: 16, y: 2 }, {
x: 17, y: 14 }, { x: 18, y: 7 },
        { x: 19, y: 7 }, { x: 20, y: 10 }],
      primaryXAxis: {
        valueType: 'Double',
        title: 'Overs'
      },
      primaryYAxis: {
        title: 'Runs',
        valueType: 'Double',
        rangePadding: 'Auto'
      },
      title: "England - Run Rate"
    };
  },
  provide: {
    chart: [AreaSeries]
  },
}
```

```
};
</script>
<style>
  #container {
    height: 350px;
  }
</style>
```

{% previewsample "page.domainurl/code-snippet/chart/axis/double-cs7" %}

## Label Format

### Numeric Label Format

Numeric labels can be formatted by using the [labelFormat](#) property.

Numeric labels supports all globalize format.

### APP.VUE

```
<template>
  <div id="app">
    <ejs-chart id="container" :title='title'
:primaryXAxis='primaryXAxis' :primaryYAxis='primaryYAxis'>
      <e-series-collection>
        <e-series :dataSource='seriesData' type='Line' xName='x'
yName='y' name='Product X' opacity=0.6> </e-series>
      </e-series-collection>
    </ejs-chart>
  </div>
</template>
<script>
import Vue from "vue";
import { ChartPlugin, LineSeries } from "@syncfusion/ej2-vue-charts";
Vue.use(ChartPlugin);
export default {
  data() {
    return {
      seriesData:[
        { x: 1900, y: 4, y1: 2.6, y2: 2.8 }, { x: 1920, y: 3.0, y1:
2.8, y2: 2.5 },
        { x: 1940, y: 3.8, y1: 2.6, y2: 2.8 }, { x: 1960, y: 3.4,
y1: 3, y2: 3.2 },
        { x: 1980, y: 3.2, y1: 3.6, y2: 2.9 }, { x: 2000, y: 3.9,
y1: 3, y2: 2 }
      ],
      primaryXAxis: {
        title: 'Year',
        edgeLabelPlacement: 'Shift'
      },
      primaryYAxis: {
        title: 'Sales Amount in Millions',
        labelFormat: 'c'
      },
      title: "Average Sales Comparison"
    };
  },
  provide: {
```

```

    chart: [LineSeries]
  },
};
</script>
<style>
#container {
  height: 350px;
}
</style>
```

{% previewsample "page.domainurl/code-snippet/chart/axis/double-cs8" %}

The following table describes the result of applying some commonly used label formats on numeric values.

<!-- markdownlint-disable MD033 -->

Label Value	Label Format property value	Result	Description
1000	n1	1000.0	The Number is rounded to 1 decimal place
1000	n2	1000.00	The Number is rounded to 2 decimal place
1000	n3	1000.000	The Number is rounded to 3 decimal place
0.01	p1	1.0%	The Number is converted to percentage with 1 decimal place
0.01	p2	1.00%	The Number is converted to percentage with 2 decimal place
0.01	p3	1.000%	The Number is converted to percentage with 3 decimal place
1000	c1	\$1000.0	The Currency symbol is appended to number and number is rounded to 1 decimal place
1000	c2	\$1000.00	The Currency symbol is appended to number and number is rounded to 2 decimal place

GroupingSeperator

To separate groups of thousands, use [useGroupingSeparator](#) property in chart.

APP.VUE

```

<template>
  <div id="app">
    <ejs-chart id="container" :title='title'
:primaryXAxis='primaryXAxis' :primaryYAxis='primaryYAxis'
useGroupingSeparator='true'>
      <e-series-collection>
        <e-series :dataSource='seriesData' type='Column' xName='x'
yName='y' name='Product X' opacity=0.6> </e-series>
      </e-series-collection>
    </ejs-chart>
  </div>
</template>
```

```

        </e-series-collection>
      </ejs-chart>
    </div>
  </template>
  <script>
import Vue from "vue";
import { ChartPlugin, ColumnSeries } from "@syncfusion/ej2-vue-charts";
Vue.use(ChartPlugin);
export default {
  data() {
    return {
      seriesData:[
        { x: 10, y: 7000 }, { x: 20, y: 1000 }, { x: 30, y: 12000 }, { x: 40,
y: 14000 }, { x: 50, y: 11000 }, { x: 60, y: 5000 },
        { x: 70, y: 7300 }, { x: 80, y: 9000 }, { x: 90, y: 12000 }, { x: 100,
y: 14000 }, { x: 110, y: 11000 }, { x: 120, y: 5000 },
      ],
      primaryXAxis: {
        title: 'Rate',
        edgeLabelPlacement: 'Shift'
      },
      primaryYAxis: {
        title: 'Sales Amount in Millions',
      },
      title: "Average Sales Comparison"
    };
  },
  provide: {
    chart: [ColumnSeries]
  },
};
  </script>
  <style>
    #container {
      height: 350px;
    }
  </style>

```

{% previewsample "page.domainurl/code-snippet/chart/axis/double-cs9" %}

### Custom Label Format

Axis also supports custom label format using placeholder like {value}°C, in which the value represent the axis

label e.g 20°C.

### APP.VUE

```

<template>
  <div id="app">
    <ejs-chart id="container" :title='title'
:primaryXAxis='primaryXAxis' :primaryYAxis='primaryYAxis'>
      <e-series-collection>
        <e-series :dataSource='seriesData' type='Line' xName='x'
yName='y' name='Product X' opacity=0.6> </e-series>
      </e-series-collection>
    </div>
  </template>

```



```

        </ejs-chart>
    </div>
</template>
<script>
import Vue from "vue";
import { ChartPlugin, LineSeries } from "@syncfusion/ej2-vue-charts";
Vue.use(ChartPlugin);
export default {
    data() {
        return {
            seriesData:[
                { x: 1900, y: 4, y1: 2.6, y2: 2.8 }, { x: 1920, y: 3.0, y1:
2.8, y2: 2.5 },
                { x: 1940, y: 3.8, y1: 2.6, y2: 2.8 }, { x: 1960, y: 3.4,
y1: 3, y2: 3.2 },
                { x: 1980, y: 3.2, y1: 3.6, y2: 2.9 }, { x: 2000, y: 3.9,
y1: 3, y2: 2 }
            ],
            primaryXAxis: {
                title: 'Year',
                edgeLabelPlacement: 'Shift'
            },
            primaryYAxis: {
                title: 'Sales Amount in Millions',
                labelFormat: '${value}k'
            },
            title: "Average Sales Comparison"
        };
    },
    provide: {
        chart: [LineSeries]
    },
};
</script>
<style>
    #container {
        height: 350px;
    }
</style>

```

{% previewsample "page.domainurl/code-snippet/chart/axis/double-cs10" %}

<!-- markdownlint-disable MD036 -->

## Date time axis in Vue Chart component

### DateTime Axis

Date time axis uses date time scale and displays the date time values as axis labels in the specified format.

### APP.VUE

```

<template>
    <div id="app">
        <ejs-chart id="container" :title='title'
:primaryXAxis='primaryXAxis' :primaryYAxis='primaryYAxis'>
            <e-series-collection>

```

```

        <e-series :dataSource='seriesData' type='Line' xName='x'
yName='y' name='Sales'> </e-series>
    </e-series-collection>
</ejs-chart>
</div>
</template>
<script>
import Vue from "vue";
import { ChartPlugin, LineSeries, DateTime } from "@syncfusion/ej2-vue-
charts";
Vue.use(ChartPlugin);
export default {
    data() {
        return {
            seriesData:[
                { x: new Date(2000, 6, 11), y: 10 }, { x: new Date(2002, 3,
7), y: 30 },
                { x: new Date(2004, 3, 6), y: 15 }, { x: new Date(2006, 3,
30), y: 65 },
                { x: new Date(2008, 3, 8), y: 90 }, { x: new Date(2010, 3,
8), y: 85 }
            ],
            primaryXAxis: {
                valueType: 'DateTime',
                title: 'Sales Across Years',
                labelFormat: 'yMMM'
            },
            primaryYAxis: {
                title: 'Sales Amount in millions(USD)',
            },
            title: "Average Sales Comparison"
        };
    },
    provide: {
        chart: [LineSeries, DateTime]
    }
};
</script>
<style>
    #container {
        height: 350px;
    }
</style>

```

{% previewsample "page.domainurl/code-snippet/chart/axis/datetime-cs2" %}

Note: To use datetime axis, we need to inject `DateTime` into the `provide` and set the `valueType` of axis to `DateTime`.

### DateTimeCategory Axis

Date-time category axis is used to display the date-time values with non-linear intervals. For example, the

business days alone have been depicted in a week here.

### APP.VUE

```

<template>
  <div id="app">
    <ejs-chart id="container" :title='title'
:primaryXAxis='primaryXAxis' :primaryYAxis='primaryYAxis'>
      <e-series-collection>
        <e-series :dataSource='seriesData' type='Line' xName='x'
yName='y' name='Sales'> </e-series>
      </e-series-collection>
    </ejs-chart>
  </div>
</template>
<script>
import Vue from "vue";
import { ChartPlugin, LineSeries, DateTimeCategory } from "@syncfusion/ej2-
vue-charts";
Vue.use(ChartPlugin);
export default {
  data() {
    return {
      seriesData:[{ x: new Date(2017, 11, 20), y: 21 }, { x: new Date(2017,
11, 21), y: 24 },
        { x: new Date(2017, 11, 22), y: 24 }, { x: new Date(2017,
11, 26), y: 70 },
        { x: new Date(2017, 11, 27), y: 75 }, { x: new Date(2018, 0,
2), y: 82 },
        { x: new Date(2018, 0, 3), y: 53 }, { x: new Date(2018, 0,
4), y: 54 },
        { x: new Date(2018, 0, 5), y: 53 }, { x: new Date(2018, 0,
8), y: 45 }
      ],
      primaryXAxis: {
        valueType: 'DateTimeCategory'
      },
      primaryYAxis: {
        title: 'Sales Amount in millions(USD)',
      },
      title: "Average Sales Comparison"
    };
  },
  provide: {
    chart: [LineSeries, DateTimeCategory]
  }
};
</script>
<style>
#container {
  height: 350px;
}
</style>

```

{% previewsample "page.domainurl/code-snippet/chart/axis/datetime-cs3" %}

Note: To use datetime axis, we need to inject `DateTimeCategory` into the `provide` and set the `valueType` of axis to `DateTimeCategory`. Of the axis using `minimum`, `maximum` and `interval` property of the axis.

**APP.VUE**

```

<template>
  <div id="app">
    <ejs-chart id="container" :title='title'
:primaryXAxis='primaryXAxis'>
      <e-series-collection>
        <e-series :dataSource='seriesData' type='Line' xName='x'
yName='y' name='Sales'> </e-series>
      </e-series-collection>
    </ejs-chart>
  </div>
</template>
<script>
import Vue from "vue";
import { ChartPlugin, LineSeries, DateTime } from "@syncfusion/ej2-vue-
charts";
Vue.use(ChartPlugin);
export default {
  data() {
    return {
      seriesData:[
        { x: new Date(2000, 6, 11), y: 10 }, { x: new Date(2002, 3,
7), y: 30 },
        { x: new Date(2004, 3, 6), y: 15 }, { x: new Date(2006, 3,
30), y: 65 },
        { x: new Date(2008, 3, 8), y: 90 }, { x: new Date(2010, 3,
8), y: 85 }
      ],
      primaryXAxis: {
        valueType: 'DateTime',
        title: 'Sales Across Years',
        labelFormat: 'yMMM',
        minimum: new Date(2000, 6, 1),
        maximum: new Date(2010, 6, 1)
      },
      title: "Average Sales Comparison"
    };
  },
  provide: {
    chart: [LineSeries, DateTime]
  }
};
</script>
<style>
#container {
  height: 350px;
}
</style>

```

{% previewsample "page.domainurl/code-snippet/chart/axis/datetime-cs4" %}

*Interval Customization*

Date time intervals can be customized by using the [interval](#) and [intervalType](#) properties of the axis. For example, when you set interval as 2 and intervalType as years, it considers 2 years as interval.

DateTime axis supports following interval types,

- Auto
- Years
- Months
- Days
- Hours
- Minutes
- Seconds

### APP.VUE

```
<template>
  <div id="app">
    <ejs-chart id="container" :title='title'
:primaryXAxis='primaryXAxis'>
      <e-series-collection>
        <e-series :dataSource='seriesData' type='Line' xName='x'
yName='y' name='Sales'> </e-series>
      </e-series-collection>
    </ejs-chart>
  </div>
</template>
<script>
import Vue from "vue";
import { ChartPlugin, LineSeries, DateTime } from "@syncfusion/ej2-vue-
charts";
Vue.use(ChartPlugin);
export default {
  data() {
    return {
      seriesData:[
        { x: new Date(2000, 6, 11), y: 10 }, { x: new Date(2002, 3,
7), y: 30 },
        { x: new Date(2004, 3, 6), y: 15 }, { x: new Date(2006, 3,
30), y: 65 },
        { x: new Date(2008, 3, 8), y: 90 }, { x: new Date(2010, 3,
8), y: 85 }
      ],
      primaryXAxis: {
        valueType: 'DateTime',
        intervalType: 'Years'
      },
      title: "Average Sales Comparison"
    };
  },
  provide: {
    chart: [LineSeries, DateTime]
  }
};
</script>
<style>
#container {
  height: 350px;
}
```

```
</style>
```

```
{% previewsample "page.domainurl/code-snippet/chart/axis/datetime-cs5" %}
```

### Applying Padding to the Range

Padding can be applied to the minimum and maximum extremes of the range by using the [rangePadding](#) property. Date time axis supports the following types

- None
- Additional

### DateTime - None

When the [rangePadding](#) is set to **None**, minimum and maximum of the axis is based on the data.

### APP.VUE

```
<template>
  <div id="app">
    <ejs-chart id="container" :title='title'
:primaryXAxis='primaryXAxis'>
      <e-series-collection>
        <e-series :dataSource='seriesData' type='Line' xName='x'
yName='y' name='Sales'> </e-series>
      </e-series-collection>
    </ejs-chart>
  </div>
</template>
<script>
import Vue from "vue";
import { ChartPlugin, LineSeries, DateTime } from "@syncfusion/ej2-vue-
charts";
Vue.use(ChartPlugin);
export default {
  data() {
    return {
      seriesData:[
        { x: new Date(2000, 6, 11), y: 10 }, { x: new Date(2002, 3,
7), y: 30 },
        { x: new Date(2004, 3, 6), y: 15 }, { x: new Date(2006, 3,
30), y: 65 },
        { x: new Date(2008, 3, 8), y: 90 }, { x: new Date(2010, 3,
8), y: 85 }
      ],
      primaryXAxis: {
        valueType: 'DateTime',
        title: 'Sales Across Years',
        labelFormat: 'yMMM',
        rangePadding: 'None'
      },
      title: "Average Sales Comparison"
    };
  },
  provide: {
```

```

    chart: [LineSeries, DateTime]
  }
};
</script>
<style>
  #container {
    height: 350px;
  }
</style>

```

{% previewsample "page.domainurl/code-snippet/chart/axis/datetime-cs6" %}

### DateTime - Round

When the [rangePadding](#) is set to **Round**, minimum and maximum will be

rounded to the nearest possible value divisible by interval. For example, when the minimum is 15th Jan, interval is

1 and the interval type is 'month', then the axis minimum will be Jan 1st.

### APP.VUE

```

<template>
  <div id="app">
    <ejs-chart id="container" :title='title'
:primaryXAxis='primaryXAxis'>
      <e-series-collection>
        <e-series :dataSource='seriesData' type='Line' xName='x'
yName='y' name='Sales'> </e-series>
      </e-series-collection>
    </ejs-chart>
  </div>
</template>
<script>
import Vue from "vue";
import { ChartPlugin, LineSeries, DateTime } from "@syncfusion/ej2-vue-
charts";
Vue.use(ChartPlugin);
export default {
  data() {
    return {
      seriesData:[
        { x: new Date(2000, 6, 11), y: 10 }, { x: new Date(2002, 3,
7), y: 30 },
        { x: new Date(2004, 3, 6), y: 15 }, { x: new Date(2006, 3,
30), y: 65 },
        { x: new Date(2008, 3, 8), y: 90 }, { x: new Date(2010, 3,
8), y: 85 }
      ],
      primaryXAxis: {
        valueType: 'DateTime',
        title: 'Sales Across Years',
        labelFormat: 'yMMM',
        rangePadding: 'Round'
      },
      title: "Average Sales Comparison"
    }
  }
}

```

```

    };
  },
  provide: {
    chart: [LineSeries, DateTime]
  }
};
</script>
<style>
  #container {
    height: 350px;
  }
</style>

```

{% previewsample "page.domainurl/code-snippet/chart/axis/datetime-cs7" %}

### DateTime - Additional

When the [rangePadding](#) is set to **Additional**, interval of an axis will be padded to the minimum and maximum of the axis.

### APP.VUE

```

<template>
  <div id="app">
    <ejs-chart id="container" :title='title'
:primaryXAxis='primaryXAxis'>
      <e-series-collection>
        <e-series :dataSource='seriesData' type='Line' xName='x'
yName='y' name='Sales'> </e-series>
      </e-series-collection>
    </ejs-chart>
  </div>
</template>
<script>
import Vue from "vue";
import { ChartPlugin, LineSeries, DateTime } from "@syncfusion/ej2-vue-
charts";
Vue.use(ChartPlugin);
export default {
  data() {
    return {
      seriesData:[
        { x: new Date(2000, 6, 11), y: 10 }, { x: new Date(2002, 3,
7), y: 30 },
        { x: new Date(2004, 3, 6), y: 15 }, { x: new Date(2006, 3,
30), y: 65 },
        { x: new Date(2008, 3, 8), y: 90 }, { x: new Date(2010, 3,
8), y: 85 }
      ],
      primaryXAxis: {
        valueType: 'DateTime',
        title: 'Sales Across Years',
        labelFormat: 'yMMM',
        rangePadding: 'Additional'
      },
      title: "Average Sales Comparison"
    }
  }
}

```



```

    };
  },
  provide: {
    chart: [LineSeries, DateTime]
  }
};
</script>
<style>
  #container {
    height: 350px;
  }
</style>

```

{% previewsample "page.domainurl/code-snippet/chart/axis/datetime-cs8" %}

### Label Format

You can format and parse the date to all globalize format using [labelFormat](#) property in an axis.

### APP.VUE

```

<template>
  <div id="app">
    <ejs-chart id="container" :title='title'
:primaryXAxis='primaryXAxis' :primaryYAxis='primaryYAxis'>
      <e-series-collection>
        <e-series :dataSource='seriesData' type='Line' xName='x'
yName='y' name='Sales'> </e-series>
      </e-series-collection>
    </ejs-chart>
  </div>
</template>
<script>
import Vue from "vue";
import { ChartPlugin, LineSeries, DateTime } from "@syncfusion/ej2-vue-
charts";
Vue.use(ChartPlugin);
export default {
  data() {
    return {
      seriesData:[
        { x: new Date(2000, 6, 11), y: 10 }, { x: new Date(2002, 3,
7), y: 30 },
        { x: new Date(2004, 3, 6), y: 15 }, { x: new Date(2006, 3,
30), y: 65 },
        { x: new Date(2008, 3, 8), y: 90 }, { x: new Date(2010, 3,
8), y: 85 }
      ],
      primaryXAxis: {
        valueType: 'DateTime',
        title: 'Sales Across Years',
        labelFormat: 'yMd'
      },
      primaryYAxis: {
        title: 'Sales Amount in millions(USD)',
      },
      title: "Average Sales Comparison"
    }
  }
}

```

```

    };
  },
  provide: {
    chart: [LineSeries, DateTime]
  }
};
</script>
<style>
  #container {
    height: 350px;
  }
</style>

```

{% previewsample "page.domainurl/code-snippet/chart/axis/datetime-cs9" %}

The following table describes the result of applying some common date time formats to the labelFormat property

<!-- markdownlint-disable MD033 -->

Label Value	Label Format Property Value	Result	Description
new Date(2000, 03, 10)	EEEE	Monday	The Date is displayed in day format
new Date(2000, 03, 10)	yMd	04/10/2000	The Date is displayed in month/date/year format
new Date(2000, 03, 10)	MMM	Apr	The Shorthand month for the date is displayed
new Date(2000, 03, 10)	hm	12:00 AM	Time of the date value is displayed as label
new Date(2000, 03, 10)	hms	12:00:00 AM	The Label is displayed in hours:minutes:seconds format

<!-- markdownlint-disable MD033 -->

### Logarithmic axis in Vue Chart component

<!-- markdownlint-disable MD033 -->

Logarithmic axis uses logarithmic scale and it is very useful in visualizing data, when it has numeric values in both lower order of magnitude (eg:  $10^{-6}$ ) and higher order of magnitude (eg:  $10^6$ ).

### APP.VUE

```

<template>
  <div id="app">
    <ejs-chart id="container" :title='title'
    :primaryXAxis='primaryXAxis' :primaryYAxis='primaryYAxis'>
      <e-series-collection>
        <e-series :dataSource='seriesData' type='Line' xName='x'
        yName='y' name='Product X'> </e-series>
      </e-series-collection>
    </ejs-chart>
  </div>
</template>

```

```

        </e-series-collection>
      </ejs-chart>
    </div>
  </template>
  <script>
import Vue from "vue";
import { ChartPlugin, LineSeries, Logarithmic, DateTime } from
"@syncfusion/ej2-vue-charts";
Vue.use(ChartPlugin);
export default {
  data() {
    return {
      seriesData:[
        { x: new Date(1995, 0, 1), y: 80 }, { x: new Date(1996, 0,
1), y: 200 },
        { x: new Date(1997, 0, 1), y: 400 }, { x: new Date(1998, 0,
1), y: 600 },
        { x: new Date(1999, 0, 1), y: 700 }, { x: new Date(2000, 0,
1), y: 1400 },
        { x: new Date(2001, 0, 1), y: 2000 }, { x: new Date(2002, 0,
1), y: 4000 },
        { x: new Date(2003, 0, 1), y: 6000 }, { x: new Date(2004, 0,
1), y: 8000 },
        { x: new Date(2005, 0, 1), y: 11000 }
      ],
      primaryXAxis: {
        valueType: 'DateTime',
        title: 'Years',
        labelFormat: 'y'
      },
      primaryYAxis: {
        valueType: 'Logarithmic',
        title: 'Profit'
      },
      title: "Product X Growth [1995-2005]"
    };
  },
  provide: {
    chart: [LineSeries, Logarithmic, DateTime]
  },
};
  </script>
  <style>
    #container {
      height: 350px;
    }
  </style>

```

{% previewsample "page.domainurl/code-snippet/chart/axis/log-cs1" %}

Note: To use log axis, we need to inject `Logarithmic` into the `provide` and set the `valueType` of axis to `Logarithmic`.

## Range

Range of an axis, will be calculated automatically based on the provided data, you can also customize the range

of the axis using [minimum](#), [maximum](#) and [interval](#) property of the axis.

## APP.VUE

```
<template>
  <div id="app">
    <ejs-chart id="container" :title='title'
:primaryXAxis='primaryXAxis' :primaryYAxis='primaryYAxis'>
      <e-series-collection>
        <e-series :dataSource='seriesData' type='Line' xName='x'
yName='y' name='Product X'> </e-series>
      </e-series-collection>
    </ejs-chart>
  </div>
</template>
<script>
import Vue from "vue";
import { ChartPlugin, LineSeries, Logarithmic, DateTime } from
"@syncfusion/ej2-vue-charts";
Vue.use(ChartPlugin);
export default {
  data() {
    return {
      seriesData:[
        { x: new Date(1995, 0, 1), y: 80 }, { x: new Date(1996, 0,
1), y: 200 },
        { x: new Date(1997, 0, 1), y: 400 }, { x: new Date(1998, 0,
1), y: 600 },
        { x: new Date(1999, 0, 1), y: 700 }, { x: new Date(2000, 0,
1), y: 1400 },
        { x: new Date(2001, 0, 1), y: 2000 }, { x: new Date(2002, 0,
1), y: 4000 },
        { x: new Date(2003, 0, 1), y: 6000 }, { x: new Date(2004, 0,
1), y: 8000 },
        { x: new Date(2005, 0, 1), y: 11000 }
      ],
      primaryXAxis: {
        valueType: 'DateTime',
        title: 'Years',
        labelFormat: 'y'
      },
      primaryYAxis: {
        valueType: 'Logarithmic',
        title: 'Profit',
        minimum: 100,
        maximum: 10000
      },
      title: "Product X Growth [1995-2005]"
    };
  },
  provide: {
    chart: [LineSeries, Logarithmic, DateTime]
  },
}
```

```
};
</script>
<style>
  #container {
    height: 350px;
  }
</style>
```

{% previewsample "page.domainurl/code-snippet/chart/axis/log-cs2" %}

### Logarithmic Base

Logarithmic base can be customized by using the [logBase](#) property of the axis.

For example when the logBase is 5, the axis values follows  $5^{-2}$ ,  $5^{-1}$ ,  $5^0$ ,

$5^1$ ,  $5^2$  etc.

### APP.VUE

```
<template>
  <div id="app">
    <ejs-chart id="container" :title='title'
      :primaryXAxis='primaryXAxis' :primaryYAxis='primaryYAxis'>
      <e-series-collection>
        <e-series :dataSource='seriesData' type='Line' xName='x'
          yName='y' name='Product X'> </e-series>
      </e-series-collection>
    </ejs-chart>
  </div>
</template>
<script>
import Vue from "vue";
import { ChartPlugin, LineSeries, Logarithmic, DateTime } from
"@syncfusion/ej2-vue-charts";
Vue.use(ChartPlugin);
export default {
  data() {
    return {
      seriesData:[
        { x: new Date(1995, 0, 1), y: 80 }, { x: new Date(1996, 0,
1), y: 200 },
        { x: new Date(1997, 0, 1), y: 400 }, { x: new Date(1998, 0,
1), y: 600 },
        { x: new Date(1999, 0, 1), y: 700 }, { x: new Date(2000, 0,
1), y: 1400 },
        { x: new Date(2001, 0, 1), y: 2000 }, { x: new Date(2002, 0,
1), y: 4000 },
        { x: new Date(2003, 0, 1), y: 6000 }, { x: new Date(2004, 0,
1), y: 8000 },
        { x: new Date(2005, 0, 1), y: 11000 }
      ],
      primaryXAxis: {
        valueType: 'DateTime',
        title: 'Years',
        labelFormat: 'y'
      },
    }
  }
}
```

```

        primaryYAxis: {
            valueType: 'Logarithmic',
            title: 'Profit',
            logBase: 2
        },
        title: "Product X Growth [1995-2005]"
    };
};
</script>
<style>
    #container {
        height: 350px;
    }
</style>

```

{% previewsample "page.domainurl/code-snippet/chart/axis/log-cs3" %}

### Logarithmic Interval

Logarithmic axis interval can be customized by using the [interval](#)

property of the axis. When the logarithmic base is 10 and logarithmic interval is 2, then the axis labels are

placed at an interval of  $10^2$ . The default value of the interval is 1.

### APP.VUE

```

<template>
    <div id="app">
        <ejs-chart id="container" :title='title'
:primaryXAxis='primaryXAxis' :primaryYAxis='primaryYAxis'>
            <e-series-collection>
                <e-series :dataSource='seriesData' type='Line' xName='x'
yName='y' name='Product X'> </e-series>
            </e-series-collection>
        </ejs-chart>
    </div>
</template>
<script>
import Vue from "vue";
import { ChartPlugin, LineSeries, Logarithmic, DateTime } from
"@syncfusion/ej2-vue-charts";
Vue.use(ChartPlugin);
export default {
    data() {
        return {
            seriesData:[
                { x: new Date(1995, 0, 1), y: 80 }, { x: new Date(1996, 0,
1), y: 200 },
                { x: new Date(1997, 0, 1), y: 400 }, { x: new Date(1998, 0,
1), y: 600 },
                { x: new Date(1999, 0, 1), y: 700 }, { x: new Date(2000, 0,
1), y: 1400 },

```

```

    { x: new Date(2001, 0, 1), y: 2000 }, { x: new Date(2002, 0, 1), y: 4000 },
    { x: new Date(2003, 0, 1), y: 6000 }, { x: new Date(2004, 0, 1), y: 8000 },
    { x: new Date(2005, 0, 1), y: 11000 }
  ],
  primaryXAxis: {
    valueType: 'DateTime',
    title: 'Years',
    labelFormat: 'y'
  },
  primaryYAxis: {
    valueType: 'Logarithmic',
    title: 'Profit',
    interval: 2
  },
  title: "Product X Growth [1995-2005]"
};
},
provide: {
  chart: [LineSeries, Logarithmic, DateTime]
}
};
</script>
<style>
  #container {
    height: 350px;
  }
</style>

```

{% previewsample "page.domainurl/code-snippet/chart/axis/log-cs4" %}

## Axis labels in Vue Chart component

### Smart Axis Labels

When the axis labels overlap with each other, you can use [labelIntersectAction](#) property in the axis, to place them smartly.

When setting `labelIntersectAction` as `Hide`

### APP.VUE

```

<template>
  <div id="app">
    <ejs-chart id="container" :primaryXAxis='primaryXAxis'
width='350px'>
      <e-series-collection>
        <e-series :dataSource='seriesData' type='Column' xName='x'
yName='y' name='Internet'> </e-series>
      </e-series-collection>
    </ejs-chart>
  </div>
</template>
<script>
import Vue from "vue";
import { ChartPlugin, ColumnSeries, Category } from "@syncfusion/ej2-vue-
charts";

```

```

Vue.use(ChartPlugin);
export default {
  data() {
    return {
      seriesData: [
        { x: "South Korea", y: 39.4 }, { x: "India", y: 61.3 }, { x:
"Pakistan", y: 20.4 },
        { x: "Germany", y: 65.1 }, { x: "Australia", y: 15.8 }, { x:
"Italy", y: 29.2 },
        { x: "France", y: 44.6 }, { x: "Saudi Arabia", y: 9.7 }, {
x: "Russia", y: 40.8 },
        { x: "Mexico", y: 31 }, { x: "Brazil", y: 75.9 }, { x:
"China", y: 51.4 }
      ],
      primaryXAxis: {
        valueType: 'Category',
        labelIntersectAction: 'Hide'
      }
    };
  },
  provide: {
    chart: [ColumnSeries, Category]
  }
};
</script>
<style>
#container {
  height: 350px;
}
</style>

```

{% previewsample "page.domainurl/code-snippet/chart/axis/multiple-cs1" %}

When setting `labelIntersectAction` as `Rotate45`

### APP.VUE

```

<template>
  <div id="app">
    <ejs-chart id="container" :primaryXAxis='primaryXAxis'
width='350px'>
      <e-series-collection>
        <e-series :dataSource='seriesData' type='Column' xName='x'
yName='y' name='Internet'> </e-series>
      </e-series-collection>
    </ejs-chart>
  </div>
</template>
<script>
import Vue from "vue";
import { ChartPlugin, ColumnSeries, Category } from "@syncfusion/ej2-vue-
charts";
Vue.use(ChartPlugin);
export default {
  data() {
    return {
      seriesData: [

```



```

        { x: "South Korea", y: 39.4 }, { x: "India", y: 61.3 }, { x:
"Pakistan", y: 20.4 },
        { x: "Germany", y: 65.1 }, { x: "Australia", y: 15.8 }, { x:
"Italy", y: 29.2 },
        { x: "France", y: 44.6 }, { x: "Saudi Arabia", y: 9.7 }, {
x: "Russia", y: 40.8 },
        { x: "Mexico", y: 31 }, { x: "Brazil", y: 75.9 }, { x:
"China", y: 51.4 }
    ],
    primaryXAxis: {
        valueType: 'Category',
        labelIntersectAction: 'Rotate45'
    }
};
},
provide: {
    chart: [ColumnSeries, Category]
}
};
</script>
<style>
    #container {
        height: 350px;
    }
</style>

```

{% previewsample "page.domainurl/code-snippet/chart/axis/multiple-cs2" %}

When setting `labelIntersectAction` as `Rotate90`

### APP.VUE

```

<template>
    <div id="app">
        <ejs-chart id="container" :primaryXAxis='primaryXAxis'
width='350px'>
            <e-series-collection>
                <e-series :dataSource='seriesData' type='Column' xName='x'
yName='y' name='Internet'> </e-series>
            </e-series-collection>
        </ejs-chart>
    </div>
</template>
<script>
import Vue from "vue";
import { ChartPlugin, ColumnSeries, Category } from "@syncfusion/ej2-vue-
charts";
Vue.use(ChartPlugin);
export default {
    data() {
        return {
            seriesData: [
                { x: "South Korea", y: 39.4 }, { x: "India", y: 61.3 }, { x:
"Pakistan", y: 20.4 },
                { x: "Germany", y: 65.1 }, { x: "Australia", y: 15.8 }, { x:
"Italy", y: 29.2 },

```

```

        { x: "France", y: 44.6 }, { x: "Saudi Arabia", y: 9.7 }, {
x: "Russia", y: 40.8 },
        { x: "Mexico", y: 31 }, { x: "Brazil", y: 75.9 }, { x:
"China", y: 51.4 }
    ],
    primaryXAxis: {
        valueType: 'Category',
        labelIntersectAction: 'Rotate90'
    }
};
},
provide: {
    chart: [ColumnSeries, Category]
}
};
</script>
<style>
    #container {
        height: 350px;
    }
</style>

```

{% previewsample "page.domainurl/code-snippet/chart/axis/multiple-cs3" %}

### Axis Labels Positioning

By default, the axis labels can be placed at **outside** the axis line and this also can be placed at **inside** the axis line using the **labelPosition** property.

### APP.VUE

```

<template>
    <div id="app">
        <ejs-chart id="container" :title='title'
:primaryXAxis='primaryXAxis' width='350px'>
            <e-series-collection>
                <e-series :dataSource='seriesData' type='Column'
xName='country' yName='gold' name='Gold'> </e-series>
            </e-series-collection>
        </ejs-chart>
    </div>
</template>
<script>
import Vue from "vue";
import { ChartPlugin, ColumnSeries, Category } from "@syncfusion/ej2-vue-
charts";
Vue.use(ChartPlugin);
export default {
    data() {
        return {
            seriesData: [
                { country: "USA", gold: 50 },
                { country: "China", gold: 40 },
                { country: "Japan", gold: 70 },
                { country: "Australia", gold: 60 },
                { country: "France", gold: 50 },
            ]
        }
    }
}

```

```

        { country: "Germany", gold: 40 },
        { country: "Italy", gold: 40 },
        { country: "Sweden", gold: 30 }
      ],
      primaryXAxis: {
        valueType: 'Category',
        title: 'Countries',
        labelPosition: 'Inside'
      },
      title: "Olympic Medals"
    };
  },
  provide: {
    chart: [ColumnSeries, Category]
  },
};
</script>
<style>
  #container {
    height: 350px;
  }
</style>

```

{% previewsample "page.domainurl/code-snippet/chart/axis/category-cs10" %}

### Multilevel Labels

Any number of levels of labels can be added to an axis using the `multiLevelLabels` property. This property can be configured using the following properties:

- Categories
- Overflow
- Alignment
- Text style
- Border

Note: To use multilevel label feature, we need to inject `MultiLevelLabel` into the `provide`.

### Categories

Using the categories property, you can configure the `start`, `end`, `text`, and `maximumTextWidth` of multilevel labels.

### APP.VUE

```

<template>
  <div id="app">
    <ejs-chart id="container" :title='title'
    :primaryXAxis='primaryXAxis'>
      <e-series-collection>
        <e-series :dataSource='seriesData' type='Column'
        xName='country' yName='gold' name='Gold'> </e-series>
      </e-series-collection>
    </ejs-chart>
  </div>

```

```

</template>
<script>
import Vue from "vue";
import { ChartPlugin, ColumnSeries, Category, MultiLevelLabel } from
"@syncfusion/ej2-vue-charts";
Vue.use(ChartPlugin);
export default {
  data() {
    return {
      seriesData: [
        { country: "USA", gold: 50 },
        { country: "China", gold: 40 },
        { country: "Japan", gold: 70 },
        { country: "Australia", gold: 60 },
        { country: "France", gold: 50 },
        { country: "Germany", gold: 40 },
        { country: "Italy", gold: 40 },
        { country: "Sweden", gold: 30 }
      ],
      primaryXAxis: {
        valueType: 'Category',
        multiLevelLabels:[{ categories: [
          {
            //Start and end values of the multi-level labels
            accepts number, date and string values
            start: -0.5,
            end: 3.5,
            //Multi-level label's text.
            text: 'Half Yearly 1',
          },
          { start: 3.5, end: 7.5, text: 'Half Yearly 2' },
        ]}]
      },
      title: "Olympic Medals"
    };
  },
  provide: {
    chart: [ColumnSeries, Category, MultiLevelLabel]
  }
};
</script>
<style>
#container {
  height: 350px;
}
</style>

```

{% previewsample "page.domainurl/code-snippet/chart/axis/category-cs11" %}

### Overflow

Using the `overflow` property, you can `trim` or `wrap` the multilevel labels.

### APP.VUE

```

<template>
  <div id="app">

```

```

<ejs-chart id="container" :title='title'
:primaryXAxis='primaryXAxis'>
  <e-series-collection>
    <e-series :dataSource='seriesData' type='Column'
xName='country' yName='gold' name='Gold'> </e-series>
  </e-series-collection>
</ejs-chart>
</div>
</template>
<script>
import Vue from "vue";
import { ChartPlugin, ColumnSeries, Category, MultiLevelLabel } from
"@syncfusion/ej2-vue-charts";
Vue.use(ChartPlugin);
export default {
  data() {
    return {
      seriesData: [
        { country: "USA", gold: 50 },
        { country: "China", gold: 40 },
        { country: "Japan", gold: 70 },
        { country: "Australia", gold: 60 },
        { country: "France", gold: 50 },
        { country: "Germany", gold: 40 },
        { country: "Italy", gold: 40 },
        { country: "Sweden", gold: 30 }
      ],
      primaryXAxis: {
        valueType: 'Category',
        multiLevelLabels:[{
          categories: [{start: -0.5, end: 3.5, text: 'Half Yearly 1',
maximumTextWidth:50 },
{ start: 3.5, end: 7.5, text: 'Half Yearly
2',maximumTextWidth:50 }]],
          overflow:'Trim'
        }]
      },
      title: "Olympic Medals"
    };
  },
  provide: {
    chart: [ColumnSeries, Category, MultiLevelLabel]
  }
};
</script>
<style>
#container {
  height: 350px;
}
</style>

```

{% previewsample "page.domainurl/code-snippet/chart/axis/category-cs12" %}

### Alignment

The **alignment** property provides option to position the multilevel labels at **far**, **center**, or **near**.

**APP.VUE**

```

<template>
  <div id="app">
    <ejs-chart id="container" :title='title'
:primaryXAxis='primaryXAxis'>
      <e-series-collection>
        <e-series :dataSource='seriesData' type='Column'
xName='country' yName='gold' name='Gold'> </e-series>
      </e-series-collection>
    </ejs-chart>
  </div>
</template>
<script>
import Vue from "vue";
import { ChartPlugin, ColumnSeries, Category, MultiLevelLabel } from
"@syncfusion/ej2-vue-charts";
Vue.use(ChartPlugin);
export default {
  data() {
    return {
      seriesData: [
        { country: "USA", gold: 50 },
        { country: "China", gold: 40 },
        { country: "Japan", gold: 70 },
        { country: "Australia", gold: 60 },
        { country: "France", gold: 50 },
        { country: "Germany", gold: 40 },
        { country: "Italy", gold: 40 },
        { country: "Sweden", gold: 30 }
      ],
      primaryXAxis: {
        valueType: 'Category',
        multiLevelLabels:[{
          categories: [{start: -0.5, end: 3.5, text: 'Half Yearly 1' },
            { start: 3.5, end: 7.5, text: 'Half Yearly 2' }]],
          alignment : 'Far'
        }]
      },
      title: "Olympic Medals"
    };
  },
  provide: {
    chart: [ColumnSeries, Category, MultiLevelLabel]
  }
};
</script>
<style>
  #container {
    height: 350px;
  }
</style>

```

{% previewsample "page.domainurl/code-snippet/chart/axis/category-cs13" %}

*Text customization*

The `textStyle` property of multilevel labels provides options to customize the `size`, `color`, `fontFamily`, `fontWeight`, `fontStyle`, `opacity`, `textAlignment` and `textOverflow`.

**APP.VUE**

```
<template>
  <div id="app">
    <ejs-chart id="container" :title='title'
:primaryXAxis='primaryXAxis'>
      <e-series-collection>
        <e-series :dataSource='seriesData' type='Column'
xName='country' yName='gold' name='Gold'> </e-series>
      </e-series-collection>
    </ejs-chart>
  </div>
</template>
<script>
import Vue from "vue";
import { ChartPlugin, ColumnSeries, Category, MultiLevelLabel } from
"@syncfusion/ej2-vue-charts";
Vue.use(ChartPlugin);
export default {
  data() {
    return {
      seriesData: [
        { country: "USA", gold: 50 },
        { country: "China", gold: 40 },
        { country: "Japan", gold: 70 },
        { country: "Australia", gold: 60 },
        { country: "France", gold: 50 },
        { country: "Germany", gold: 40 },
        { country: "Italy", gold: 40 },
        { country: "Sweden", gold: 30 }
      ],
      primaryXAxis: {
        valueType: 'Category',
        multiLevelLabels:[{
          categories: [{start: -0.5, end: 3.5, text: 'Half Yearly 1' },
            { start: 3.5, end: 7.5, text: 'Half Yearly 2' }],
          textStyle:{size:'18px', color:'Red'}
        }]
      },
      title: "Olympic Medals"
    };
  },
  provide: {
    chart: [ColumnSeries, Category, MultiLevelLabel]
  }
};
</script>
<style>
#container {
  height: 350px;
}
</style>
```

{% previewsample "page.domainurl/code-snippet/chart/axis/category-cs14" %}

### Border customization

Using the **border** property, you can customize the **width**, **color**, and **type**. The **type** of border are **Rectangle**, **Brace**, **WithoutBorder**, **WithoutTopBorder**, **WithoutTopandBottomBorder** and **CurlyBrace**.

### APP.VUE

```
<template>
  <div id="app">
    <ejs-chart id="container" :title='title'
:primaryXAxis='primaryXAxis'>
      <e-series-collection>
        <e-series :dataSource='seriesData' type='Column'
xName='country' yName='gold' name='Gold'> </e-series>
      </e-series-collection>
    </ejs-chart>
  </div>
</template>
<script>
import Vue from "vue";
import { ChartPlugin, ColumnSeries, Category, MultiLevelLabel } from
"@syncfusion/ej2-vue-charts";
Vue.use(ChartPlugin);
export default {
  data() {
    return {
      seriesData: [
        { country: "USA", gold: 50 },
        { country: "China", gold: 40 },
        { country: "Japan", gold: 70 },
        { country: "Australia", gold: 60 },
        { country: "France", gold: 50 },
        { country: "Germany", gold: 40 },
        { country: "Italy", gold: 40 },
        { country: "Sweden", gold: 30 }
      ],
      primaryXAxis: {
        valueType: 'Category',
        multiLevelLabels:[{
          categories: [{start: -0.5, end: 3.5, text: 'Half Yearly 1' },
            { start: 3.5, end: 7.5, text: 'Half Yearly 2' }]],
          border:{type:'Brace', color:'Blue', width: 2},
        }]
      },
      title: "Olympic Medals"
    };
  },
  provide: {
    chart: [ColumnSeries, Category, MultiLevelLabel]
  }
};
</script>
```



```
<style>
  #container {
    height: 350px;
  }
</style>
```

```
{% previewsample "page.domainurl/code-snippet/chart/axis/category-cs15" %}
```

### Edge Label Placement

Labels with long text at the edges of an axis may appear partially in the chart. To avoid this, use [edgeLabelPlacement](#) property in axis, which moves the label inside the chart area for better appearance or hides it.

### APP.VUE

```
<template>
  <div id="app">
    <ejs-chart id="container" :title='title'
:primaryXAxis='primaryXAxis'>
      <e-series-collection>
        <e-series :dataSource='seriesData' type='Line' xName='x'
yName='y' name='Sales'> </e-series>
      </e-series-collection>
    </ejs-chart>
  </div>
</template>
<script>
import Vue from "vue";
import { ChartPlugin, LineSeries, DateTime } from "@syncfusion/ej2-vue-
charts";
Vue.use(ChartPlugin);
export default {
  data() {
    return {
      seriesData:[
        { x: new Date(2000, 6, 11), y: 10 }, { x: new Date(2002, 3,
7), y: 30 },
        { x: new Date(2004, 3, 6), y: 15 }, { x: new Date(2006, 3,
30), y: 65 },
        { x: new Date(2008, 3, 8), y: 90 }, { x: new Date(2010, 3,
8), y: 85 }
      ],
      primaryXAxis: {
        valueType: 'DateTime',
        title: 'Sales Across Years',
        labelFormat: 'yMMM',
        minimum: new Date(2000, 6, 1),
        maximum: new Date(2010, 6, 1),
        edgeLabelPlacement: 'Shift'
      },
      title: "Average Sales Comparison"
    };
  },
  provide: {
    chart: [LineSeries, DateTime]
  }
}
```

```
};
</script>
<style>
  #container {
    height: 350px;
  }
</style>
```

{% previewsample "page.domainurl/code-snippet/chart/axis/datetime-cs1" %}

### Labels Customization

The [labelStyle](#) property of an axis provides options to customize the color, font-family, font-size and font-weight of the axis labels.

### APP.VUE

```
<template>
  <div id="app">
    <ejs-chart id="container" :title='title'
:primaryXAxis='primaryXAxis' :primaryYAxis='primaryYAxis'>
      <e-series-collection>
        <e-series :dataSource='seriesData' type='Column'
xName='country' yName='gold' name='Gold'> </e-series>
        <e-series :dataSource='seriesData' type='Column'
xName='country' yName='silver' name='Silver'> </e-series>
        <e-series :dataSource='seriesData' type='Column'
xName='country' yName='bronze' name='Bronze'> </e-series>
      </e-series-collection>
    </ejs-chart>
  </div>
</template>
<script>
import Vue from "vue";
import { ChartPlugin, ColumnSeries, Category } from "@syncfusion/ej2-vue-
charts";
Vue.use(ChartPlugin);
export default {
  data() {
    return {
      seriesData: [
        { country: "USA", gold: 50, silver: 70, bronze: 45 },
        { country: "China", gold: 40, silver: 60, bronze: 55 },
        { country: "Japan", gold: 70, silver: 60, bronze: 50 },
        { country: "Australia", gold: 60, silver: 56, bronze: 40 },
        { country: "France", gold: 50, silver: 45, bronze: 35 },
        { country: "Germany", gold: 40, silver: 30, bronze: 22 },
        { country: "Italy", gold: 40, silver: 35, bronze: 37 },
        { country: "Sweden", gold: 30, silver: 25, bronze: 27 }
      ],
      primaryXAxis: {
        valueType: 'Category',
        title: 'Countries'
      },
      primaryYAxis: {
        minimum: 0, maximum: 80,
        interval: 20, title: 'Medals',
```

```

        labelFormat: '${value}K',
        titleStyle: {
            size: '16px', color: 'grey',
            fontFamily: 'Segoe UI', fontWeight: 'bold'
        },
        labelStyle: {
            size: '14px', color: 'blue',
            fontFamily: 'Segoe UI', fontWeight: 'bold'
        }
    },
    title: "Olympic Medals"
};
},
provide: {
    chart: [ColumnSeries, Category]
}
};
</script>
<style>
    #container {
        height: 350px;
    }
</style>

```

{% previewsample "page.domainurl/code-snippet/chart/axis/category-cs16" %}

### Customizing Specific Point

You can customize the specific text in the axis labels using `axisLabelRender` event.

### APP.VUE

```

<template>
    <div id="app">
        <ejs-chart id="container" :title='title'
        :primaryXAxis='primaryXAxis' >
            <e-series-collection>
                <e-series :dataSource='seriesData' type='Column'
                xName='country' yName='gold' name='Gold'> </e-series>
            </e-series-collection>
        </ejs-chart>
    </div>
</template>
<script>
import Vue from "vue";
import { ChartPlugin, ColumnSeries, Category } from "@syncfusion/ej2-vue-
charts";
Vue.use(ChartPlugin);
export default {
    data() {
        return {
            seriesData: [
                { country: "USA", gold: 50 },
                { country: "China", gold: 40 },
                { country: "Japan", gold: 70 },
                { country: "Australia", gold: 60 },
                { country: "France", gold: 50 },
            ]
        }
    }
}

```

```

        { country: "Germany", gold: 40 },
        { country: "Italy", gold: 40 },
        { country: "Sweden", gold: 30 }
      ],
      primaryXAxis: {
        valueType: 'Category',
        title: 'Countries'
      },
      title: "Olympic Medals"
    };
  },
  provide: {
    chart: [ColumnSeries, Category]
  },
};
</script>
<style>
  #container {
    height: 350px;
  }
</style>

```

{% previewsample "page.domainurl/code-snippet/chart/axis/category-cs17" %}

Trim using maximum label width

You can trim the label using [enableTrim](#) property and width of the labels can also be customized using [maximumLabelWidth](#) property in the axis, the value maximum label width is 34 by default.

### APP.VUE

```

<template>
  <div id="app">
    <ejs-chart id="container" :title='title'
    :primaryXAxis='primaryXAxis' :axisLabelRender='axisLabelRender'>
      <e-series-collection>
        <e-series :dataSource='seriesData' type='Column'
        xName='country' yName='gold' name='Gold'> </e-series>
      </e-series-collection>
    </ejs-chart>
  </div>
</template>
<script>
import Vue from "vue";
import { ChartPlugin, ColumnSeries, Category } from "@syncfusion/ej2-vue-charts";
Vue.use(ChartPlugin);
export default {
  data() {
    return {
      seriesData: [
        { country: "USA", gold: 50 },
        { country: "China", gold: 40 },
        { country: "Japan", gold: 70 },
        { country: "Australia", gold: 60 },
        { country: "France", gold: 50 },
        { country: "Germany", gold: 40 },

```

```

        { country: "Italy", gold: 40 },
        { country: "Sweden", gold: 30 }
      ],
      primaryXAxis: {
        valueType: 'Category',
        title: 'Countries',
        enableTrim: true,
        maximumLabelWidth: '22'
      },
      title: "Olympic Medals"
    };
  },
  provide: {
    chart: [ColumnSeries, Category]
  },
  methods: {
    axisLabelRender: function(args) {
      if(args.text === 'France') {
        args.labelStyle.color = 'Red';
      }
    }
  }
};
</script>
<style>
  #container {
    height: 350px;
  }
</style>

```

{% previewsample "page.domainurl/code-snippet/chart/axis/category-cs18" %}

### Line break support

Line break feature used to customize the long axis label text into multiple lines by using tag. Refer the below example in that dataSource x value contains long text, it breaks into two lines by using `<br>` tag.

### APP.VUE

```

<template>
  <div id="app">
    <ejs-chart id="container" :primaryXAxis='primaryXAxis'
    :primaryYAxis='primaryYAxis'>
      <e-series-collection>
        <e-series :dataSource='seriesData' type='Bar' xName='x'
        yName='y' name='Users' :marker='marker' tooltipMappingName='country'> </e-
        series>
      </e-series-collection>
    </ejs-chart>
  </div>
</template>
<script>
import Vue from "vue";
import { ChartPlugin, BarSeries, DateTime, Category, DataLabel } from
"@syncfusion/ej2-vue-charts";
Vue.use(ChartPlugin);
export default {

```

```

data() {
  return {
    seriesData:[
      { x: 'Germany', y: 72, country: 'GER: 72'},
      { x: 'Russia', y: 103.1, country: 'RUS: 103.1'},
      { x: 'Brazil', y: 139.1, country: 'BRZ: 139.1'},
      { x: 'India', y: 462.1, country: 'IND: 462.1'},
      { x: 'China', y: 721.4, country: 'CHN: 721.4'},
      { x: 'United States<br>Of America', y: 286.9, country:
'USA: 286.9'},
      { x: 'Great Britain', y: 115.1, country: 'GBR: 115.1'},
      { x: 'Nigeria', y: 97.2, country: 'NGR: 97.2'}
    ],
    primaryXAxis: {
      title: 'Country',
      valueType: 'Category',
      majorGridLines: { width: 0 },
      enableTrim: false,
    },
    primaryYAxis: {
      minimum: 0,
      maximum: 800,
      // labelFormat: Browser.isDevice ? '{value}' : '{value}M',
      labelStyle: {
        color: 'transparent'
      }
    },
    marker: {
      dataLabel: { visible: true, position: 'Top', font: {
fontWeight: '600',
        color: '#ffffff' } }
    }
  };
},
provide: {
  chart: [BarSeries, DateTime, Category, DataLabel]
}
};
</script>
<style>
  #container {
    height: 350px;
  }
</style>

```

{% previewsample "page.domainurl/code-snippet/chart/axis/category-cs19" %}

## Axis customization in Vue Chart component

### Axis Crossing

An axis can be positioned in the chart area using `crossesAt` and `crossesInAxis` properties. The `crossesAt`

property specifies the values (datetime, numeric, or logarithmic) at which the axis line has to be intersected

with the vertical axis or vice-versa, and the `crossesInAxis` property specifies the axis name with which the

axis line has to be crossed.

### APP.VUE

```
<template>
  <div id="app">
    <ejs-chart id="container" :title='title'
:primaryXAxis='primaryXAxis' :primaryYAxis='primaryYAxis'>
      <e-series-collection>
        <e-series :dataSource='seriesData' type='Column'
xName='country' yName='gold' name='Gold'> </e-series>
      </e-series-collection>
    </ejs-chart>
  </div>
</template>
<script>
import Vue from "vue";
import { ChartPlugin, ColumnSeries, Category } from "@syncfusion/ej2-vue-
charts";
Vue.use(ChartPlugin);
export default {
  data() {
    return {
      seriesData: [
        { country: "USA", gold: 50 },
        { country: "China", gold: 40 },
        { country: "Japan", gold: 70 },
        { country: "Australia", gold: 60 },
        { country: "France", gold: 50 },
        { country: "Germany", gold: 40 },
        { country: "Italy", gold: 40 },
        { country: "Sweden", gold: 30 }
      ],
      primaryXAxis: {
        valueType: 'Category',
        crossesAt : 15
      },
      primaryYAxis:
      {
        crossesAt : 5
      },
      title: "Olympic Medals"
    };
  },
  provide: {
    chart: [ColumnSeries, Category]
  }
};
</script>
<style>
#container {
  height: 350px;
}
</style>
```

{% previewsample "page.domainurl/code-snippet/chart/axis/category-cs2" %}

### Title

You can add a title to the axis using [title](#) property to provide quick information to the user about the data plotted in the axis.

### APP.VUE

```
<template>
  <div id="app">
    <ejs-chart id="container" :title='title'
:primaryXAxis='primaryXAxis' :primaryYAxis='primaryYAxis'>
      <e-series-collection>
        <e-series :dataSource='seriesData' type='Column'
xName='country' yName='gold' name='Gold'> </e-series>
      </e-series-collection>
    </ejs-chart>
  </div>
</template>
<script>
import Vue from "vue";
import { ChartPlugin, ColumnSeries, Category } from "@syncfusion/ej2-vue-
charts";
Vue.use(ChartPlugin);
export default {
  data() {
    return {
      seriesData: [
        { country: "USA", gold: 50 },
        { country: "China", gold: 40 },
        { country: "Japan", gold: 70 },
        { country: "Australia", gold: 60 },
        { country: "France", gold: 50 },
        { country: "Germany", gold: 40 },
        { country: "Italy", gold: 40 },
        { country: "Sweden", gold: 30 }
      ],
      primaryXAxis: {
        valueType: 'Category',
        title: 'Countries'
      },
      primaryYAxis:
      {
        minimum: 0, maximum: 80,
        interval: 20, title: 'Medals',
        labelFormat: '${value}K',
        titleStyle: {
          size: '16px', color: 'grey',
          fontFamily : 'Segoe UI', fontWeight : 'bold'
        }
      },
      title: "Olympic Medals"
    };
  },
  provide: {
```



```

    chart: [ColumnSeries, Category]
  },
};
</script>
<style>
  #container {
    height: 350px;
  }
</style>

```

{% previewsample "page.domainurl/code-snippet/chart/axis/category-cs3" %}

### Title Rotation

By using the [titleRotation](#) property, you can rotate the axis title from 0 to 360 degree.

### APP.VUE

```

<template>
  <div id="app">
    <ejs-chart id="container" :title='title'
:primaryXAxis='primaryXAxis' :primaryYAxis='primaryYAxis'>
      <e-series-collection>
        <e-series :dataSource='seriesData' type='Column'
xName='country' yName='gold' name='Gold'> </e-series>
      </e-series-collection>
    </ejs-chart>
  </div>
</template>
<script>
import Vue from "vue";
import { ChartPlugin, ColumnSeries, Category } from "@syncfusion/ej2-vue-
charts";
Vue.use(ChartPlugin);
export default {
  data() {
    return {
      seriesData: [
        { country: "USA", gold: 50 },
        { country: "China", gold: 40 },
        { country: "Japan", gold: 70 },
        { country: "Australia", gold: 60 },
        { country: "France", gold: 50 },
        { country: "Germany", gold: 40 },
        { country: "Italy", gold: 40 },
        { country: "Sweden", gold: 30 }
      ],
      primaryXAxis: {
        valueType: 'Category',
        title: 'Countries', titleRotation: 90,
        titleStyle: {
          size: '16px', color: 'grey',
          fontFamily : 'Segoe UI', fontWeight : 'bold'
        }
      },
      primaryYAxis:
    {

```

```

        minimum: 0, maximum: 80,
        interval: 20, title: 'Medals',
        labelFormat: '${value}K',
    },
    title: "Olympic Medals"
};
},
provide: {
    chart: [ColumnSeries, Category]
},
};
</script>
<style>
    #container {
        height: 350px;
    }
</style>

```

{% previewsample "page.domainurl/code-snippet/chart/axis/category-cs4" %}

### Tick Lines Customization

You can customize the **width**, **color** and **size** of the minor and major tick lines, using [majorTickLines](#) and [minorTickLines](#) properties in the axis.

### APP.VUE

```

<template>
    <div id="app">
        <ejs-chart id="container" :title='title'
        :primaryXAxis='primaryXAxis' :primaryYAxis='primaryYAxis'>
            <e-series-collection>
                <e-series :dataSource='seriesData' type='Column' xName='x'
                yName='y' name='Temperature'> </e-series>
            </e-series-collection>
        </ejs-chart>
    </div>
</template>
<script>
import Vue from "vue";
import { ChartPlugin, ColumnSeries, Category } from "@syncfusion/ej2-vue-
charts";
Vue.use(ChartPlugin);
export default {
    data() {
        return {
            seriesData: [
                { x: 'Jan', y: 60 }, { x: 'Feb', y: 50 }, { x: 'Mar', y: 64
            },
                { x: 'Apr', y: 63 }, { x: 'May', y: 81 }, { x: 'Jun', y: 64
            },
                { x: 'Jul', y: 82 }, { x: 'Aug', y: 96 }, { x: 'Sep', y: 78
            },
                { x: 'Oct', y: 60 }, { x: 'Nov', y: 58 }, { x: 'Dec', y: 56
            }
        ],
        primaryXAxis: {

```

```

        valueType: 'Category',
        majorTickLines : {
            color : 'blue',
            width : 5
        },
        minorTickLines : {
            color : 'red',
            width : 0
        }
    },
    primaryYAxis:
    {
        title: 'Temperature (Fahrenheit)',
        majorTickLines : {
            color : 'blue',
            width : 5
        },
        minorTickLines : {
            color : 'red',
            width : 0
        }
    },
    title: "Temperature flow over months"
};
},
provide: {
    chart: [ColumnSeries, Category]
},
};
</script>
<style>
    #container {
        height: 350px;
    }
</style>

```

{% previewsample "page.domainurl/code-snippet/chart/axis/category-cs5" %}

### Grid Lines Customization

You can customize the **width**, **color** and **dashArray** of the minor and major grid lines, using [majorGridLines](#) and [minorGridLines](#) properties in the axis.

### APP.VUE

```

<template>
    <div id="app">
        <ejs-chart id="container" :title='title'
        :primaryXAxis='primaryXAxis' :primaryYAxis='primaryYAxis'>
            <e-series-collection>
                <e-series :dataSource='seriesData' type='Column' xName='x'
                yName='y' name='Temperature'> </e-series>
            </e-series-collection>
        </ejs-chart>
    </div>
</template>
<script>

```

```

import Vue from "vue";
import { ChartPlugin, ColumnSeries, Category } from "@syncfusion/ej2-vue-charts";
Vue.use(ChartPlugin);
export default {
  data() {
    return {
      seriesData: [
        { x: 'Jan', y: 60 }, { x: 'Feb', y: 50 }, { x: 'Mar', y: 64 },
        { x: 'Apr', y: 63 }, { x: 'May', y: 81 }, { x: 'Jun', y: 64 },
        { x: 'Jul', y: 82 }, { x: 'Aug', y: 96 }, { x: 'Sep', y: 78 },
        { x: 'Oct', y: 60 }, { x: 'Nov', y: 58 }, { x: 'Dec', y: 56 }
      ],
      primaryXAxis: {
        valueType: 'Category',
        majorTickLines: {
          color: 'blue',
          width: 5
        },
        minorTickLines: {
          color: 'red',
          width: 0
        }
      },
      primaryYAxis: {
        majorGridLines: {
          color: 'blue',
          width: 1
        },
        minorGridLines: {
          color: 'red',
          width: 0
        }
      },
      title: "Temperature flow over months"
    };
  },
  provide: {
    chart: [ColumnSeries, Category]
  },
};
</script>
<style>
  #container {
    height: 350px;
  }
</style>

```

{% previewsample "page.domainurl/code-snippet/chart/axis/category-cs6" %}

## Multiple Axis

In addition to primary X and Y axis, we can add n number of axis to the chart. Series can be associated with

this axis, by mapping with axis's unique name.

### APP.VUE

```
<template>
  <div id="app">
    <ejs-chart id="container" :title='title'
:primaryXAxis='primaryXAxis' :axes='axes'>
      <e-series-collection>
        <e-series :dataSource='seriesData' type='Column' xName='x'
yName='y' name='Germany'> </e-series>
        <e-series :dataSource='seriesData' type='Line' xName='x'
yName='y1' yAxisName='yAxis' name='Japan'
:marker='marker'> </e-series>
      </e-series-collection>
    </ejs-chart>
  </div>
</template>
<script>
import Vue from "vue";
import { ChartPlugin, ColumnSeries, LineSeries, Category } from
"@syncfusion/ej2-vue-charts";
Vue.use(ChartPlugin);
export default {
  data() {
    return {
      seriesData: [
        { x: 'Jan', y: 15, y1: 33 }, { x: 'Feb', y: 20, y1: 31 }, {
x: 'Mar', y: 35, y1: 30 },
        { x: 'Apr', y: 40, y1: 28 }, { x: 'May', y: 80, y1: 29 }, {
x: 'Jun', y: 70, y1: 30 },
        { x: 'Jul', y: 65, y1: 33 }, { x: 'Aug', y: 55, y1: 32 }, {
x: 'Sep', y: 50, y1: 34 },
        { x: 'Oct', y: 30, y1: 32 }, { x: 'Nov', y: 35, y1: 32 }, {
x: 'Dec', y: 35, y1: 31 }
      ],
      primaryXAxis: {
        valueType: 'Category'
      },
      axes:
      [
        {
          majorGridLines: { width: 0 },
          rowIndex: 1, opposedPosition: true,
          lineStyle: { width: 0 },
          minimum: 24, maximum: 36, interval: 4,
          name: 'yAxis', title: 'Temperature (Celsius)',
          labelFormat: '{value}°C'
        }
      ],
      marker: { visible: true, width: 10, height: 10, border: { width: 2,
color: '#F8AB1D' } },
      title: "Weather Condition"
    }
  }
}
```

```

    };
  },
  provide: {
    chart: [ColumnSeries, Category]
  },
};
</script>
<style>
  #container {
    height: 350px;
  }
</style>

```

{% previewsample "page.domainurl/code-snippet/chart/axis/category-cs7" %}

### Inversed Axis

<!-- markdownlint-disable MD033 -->

When an axis is inversed, highest value of the axis comes closer to origin and vice versa. To place an axis in inversed manner set this property [isInversed](#) to true.

### APP.VUE

```

<template>
  <div id="app">
    <ejs-chart id="container" :title='title'
    :primaryYAxis='primaryYAxis'>
      <e-series-collection>
        <e-series :dataSource='seriesData' type='Column' xName='x'
        yName='y' name='Years'> </e-series>
      </e-series-collection>
    </ejs-chart>
  </div>
</template>
<script>
import Vue from "vue";
import { ChartPlugin, ColumnSeries } from "@syncfusion/ej2-vue-charts";
Vue.use(ChartPlugin);
export default {
  data() {
    return {
      seriesData: [
        { x: 2008, y: 15.1 }, { x: 2009, y: 16 }, { x: 2010, y: 21.4 },
        { x: 2011, y: 18 }, { x: 2012, y: 16.2 }, { x: 2013, y: 11 },
        { x: 2014, y: 7.6 }, { x: 2015, y: 1.5 }
      ],
      primaryYAxis:
        {
          isInversed: true
        },
      title: "Exchange Rate"
    };
  },
  provide: {
    chart: [ColumnSeries]
  },
};

```

```
};
</script>
<style>
  #container {
    height: 350px;
  }
</style>
```

```
{% previewsample "page.domainurl/code-snippet/chart/axis/category-cs8" %}
```

### Opposed Position

```
<!-- markdownlint-disable MD012 -->
```

To place an axis opposite from its original position, set [opposedPosition](#) property of the axis to true.

```
<!-- markdownlint-disable MD012 -->
```

### APP.VUE

```
<template>
  <div id="app">
    <ejs-chart id="container" :title='title'
:primaryXAxis='primaryXAxis' :primaryYAxis='primaryYAxis'>
      <e-series-collection>
        <e-series :dataSource='seriesData' type='Column' xName='x'
yName='y' name='Temperature'> </e-series>
      </e-series-collection>
    </ejs-chart>
  </div>
</template>
<script>
import Vue from "vue";
import { ChartPlugin, ColumnSeries, Category } from "@syncfusion/ej2-vue-
charts";
Vue.use(ChartPlugin);
export default {
  data() {
    return {
      seriesData: [
        { x: 'Jan', y: 60 }, { x: 'Feb', y: 50 }, { x: 'Mar', y: 64
},
        { x: 'Apr', y: 63 }, { x: 'May', y: 81 }, { x: 'Jun', y: 64
},
        { x: 'Jul', y: 82 }, { x: 'Aug', y: 96 }, { x: 'Sep', y: 78
},
        { x: 'Oct', y: 60 }, { x: 'Nov', y: 58 }, { x: 'Dec', y: 56
}
      ],
      primaryXAxis: {
        valueType: 'Category'
      },
      primaryYAxis: {
        title: 'Temperature (Fahrenheit)',
        opposedPosition: true
      },
    }
  }
}
```

```

        title: "Temperature flow over months"
      };
    },
    provide: {
      chart: [ColumnSeries, Category]
    }
  };
</script>
<style>
  #container {
    height: 350px;
  }
</style>

```

{% previewsample "page.domainurl/code-snippet/chart/axis/category-cs9" %}

<!-- markdownlint-disable MD036 -->

### Strip line in Vue Chart component

<!-- markdownlint-disable MD036 -->

EJ2 chart supports horizontal and vertical strip lines and customization of stripline in both orientation.

To use Stripline in axis, we need to inject **Stripline** into the **provide**

#### Horizontal Strip lines

You can create Horizontal stripline by adding the **stripline** in the vertical axis and set **visible** option to true. Striplines are rendered in the specified start to end range and you can add more than one stripline for an axis.

#### APP.VUE

```

<template>
  <div id="app">
    <ejs-chart id="container" :title='title'
    :primaryXAxis='primaryXAxis' :primaryYAxis='primaryYAxis'>
      <e-series-collection>
        <e-series :dataSource='seriesData' type='Column' xName='x'
        yName='y' name='Run Rates'> </e-series>
      </e-series-collection>
    </ejs-chart>
  </div>
</template>
<script>
import Vue from "vue";
import { ChartPlugin, ColumnSeries, StripLine } from "@syncfusion/ej2-vue-
charts";
Vue.use(ChartPlugin);
export default {
  data() {
    return {
      seriesData: [
        {x: 1, y: 20}, {x: 2, y: 22}, {x: 3, y: 0}, {x: 4, y: 12}, {x:
5, y: 5},
        {x: 6, y: 15}, {x: 7, y: 6}, {x: 8, y: 12}, {x: 9, y: 20}, {x:
10, y: 7}
      ],
    }
  }
}

```



```

        primaryYAxis:
        {
            title: 'Runs',
            stripLines:[
                { start: 15, end: 22, text: 'Good', color: '#ff512f', visible:
true, zIndex: 'Behind', opacity: 0.5 }
                { start: 8, end: 15, text: 'Medium', color: 'pink', opacity:
0.5, visible: true, zIndex: 'Behind' }
                { start: 0, end: 8, text:'Not enough', color: 'skyblue',
opacity: 0.5, visible: true, zIndex: 'Behind' }}
            ],
            primaryXAxis: {
                title: 'Overs'
            },
            title: "India Vs Australia 1st match"
        };
    },
    provide: {
        chart: [ColumnSeries, StripLine]
    },
};
</script>
<style>
    #container {
        height: 350px;
    }
</style>

```

{% previewsample "page.domainurl/code-snippet/chart/axis/strip-line-cs1" %}

### Vertical Striplines

You can create vertical stripline by adding the `stripline` in the horizontal axis and set `visible` option to true. Striplines are rendered in the specified start to end range and you can add more than one stripline for an axis.

### APP.VUE

```

<template>
    <div id="app">
        <ejs-chart id="container" :title='title'
:primaryXAxis='primaryXAxis':primaryYAxis='primaryYAxis'>
            <e-series-collection>
                <e-series :dataSource='seriesData' type='Column' xName='x'
yName='y' name='Run Rates'> </e-series>
            </e-series-collection>
        </ejs-chart>
    </div>
</template>
<script>
import Vue from "vue";
import { ChartPlugin, ColumnSeries, StripLine } from "@syncfusion/ej2-vue-
charts";
Vue.use(ChartPlugin);
export default {
    data() {
        return {

```

```

        seriesData: [
            {x: 1, y: 20}, {x: 2, y: 22}, {x: 3, y: 0}, {x: 4, y: 12}, {x:
5, y: 5},
            {x: 6, y: 15}, {x: 7, y: 6}, {x: 8, y: 12}, {x: 9, y: 20}, {x:
10, y: 7}
        ],
        primaryYAxis:
        {
            title: 'Runs'
        },
        primaryXAxis: {
            title: 'Overs',
            stripLines:[
                {start: 0, end: 5, text: 'powerplay 1', color: 'red', visible:
true, opacity: 0.5, rotation: 45, textStyle: { size: 20, color: 'black'}},
                {start: 5, end: 10, text: 'powerplay 2', color: 'blue', visible:
true, opacity: 0.5, rotation: 45, textStyle: { size: 20, color: 'black'}},
            ]
        },
        title: "India Vs Australia 1st match"
    };
},
provide: {
    chart: [ColumnSeries, StripLine]
},
}
</script>
<style>
    #container {
        height: 350px;
    }
</style>

```

{% previewsample "page.domainurl/code-snippet/chart/axis/strip-line-cs2" %}

### Customize the strip line

Starting value in specific strip line can be customized by **start** property in strip line. Similarly, ending value is customized by **end**. It can be also set for starting from the corresponding origin of the axis by **startFromOrigin**. Size of the strip line is customized by **size**. Border for the stripline is customized by **border**. Order of the strip line such that whether it should be rendered in behind or over the series elements

is customized by **zIndex**.

### APP.VUE

```

<template>
    <div id="app">
        <ejs-chart id="container" :title='title'
:primaryXAxis='primaryXAxis' :primaryYAxis='primaryYAxis'>
            <e-series-collection>
                <e-series :dataSource='seriesData' type='Column' xName='x'
yName='y' name='Run Rates'> </e-series>
            </e-series-collection>
        </ejs-chart>
    </div>
</template>

```

```

    </div>
  </template>
  <script>
  import Vue from "vue";
  import { ChartPlugin, ColumnSeries, StripLine } from "@syncfusion/ej2-vue-charts";
  Vue.use(ChartPlugin);
  export default {
    data: function() {
      return {
        seriesData: [
          {x: 1, y: 20},{x: 2, y: 22},{x: 3, y: 0},{x: 4, y: 12},{x:
5, y: 5},
          {x: 6, y: 15},{x: 7, y: 6},{x: 8, y: 12},{x: 9, y: 20},{x:
10, y: 7}
        ],
        primaryYAxis:
        {
          title: 'Runs'
        },
        primaryXAxis: {
          stripLines:[
            { startFromOrigin: true, size: 4, zIndex: 'Behind', opacity:
0.5, border: { width: 2, color:'red'}}
          ],
          title: 'Overs'
        },
        title: "India Vs Australia 1st match"
      };
    },
    provide: {
      chart: [ColumnSeries, StripLine]
    },
  };
  </script>
  <style>
  #container {
    height: 350px;
  }
  </style>

```

{% previewsample "page.domainurl/code-snippet/chart/axis/strip-line-cs3" %}

### Customize the stripline text

You can customize the text rendered in stripline by `textStyle` property. Rotation of the strip line text can be changed by `rotation` property. Horizontal and Vertical alignment of stripline text can be changed by `horizontalAlignment` and `verticalAlignment` property.

### APP.VUE

```

<template>
  <div id="app">
    <ejs-chart id="container" :title='title'
:primaryXAxis='primaryXAxis' :primaryYAxis='primaryYAxis'>
      <e-series-collection>

```

```

        <e-series :dataSource='seriesData' type='Column' xName='x'
yName='y' name='Run Rates'> </e-series>
    </e-series-collection>
</ejs-chart>
</div>
</template>
<script>
import Vue from "vue";
import { ChartPlugin, ColumnSeries, StripLine } from "@syncfusion/ej2-vue-
charts";
Vue.use(ChartPlugin);
export default {
    data() {
        return {
            seriesData: [
                {x: 1, y: 20},{x: 2, y: 22},{x: 3, y: 0},{x: 4, y: 12},{x:
5, y: 5},
                {x: 6, y: 15},{x: 7, y: 6},{x: 8, y: 12},{x: 9, y: 20},{x:
10, y: 7}
            ],
            primaryYAxis:
            {
                title: 'Runs'
            },
            primaryXAxis: {
                stripLines:[
                    { startFromOrigin: true, size: 4, zIndex: 'Behind', opacity:
0.5, text: 'Good', verticalAlignment: 'Middle', horizontalAlignment:
'Middle', rotation: 90, textStyle: { size: 15}},
                    { start: 5, end: 8, verticalAlignment: 'Start',
horizontalAlignment: 'End', rotation: 45, text: 'Poor'}
                ],
                title: 'Overs'
            },
            title: "India Vs Australia 1st match"
        };
    },
    provide: {
        chart: [ColumnSeries, StripLine]
    },
};
</script>
<style>
#container {
    height: 350px;
}
</style>

```

{% previewsample "page.domainurl/code-snippet/chart/axis/strip-line-cs4" %}

### Dash Array

You can create dash array stripline by using `dashArray` property. The Striplines are rendered with specified dash array values.

### APP.VUE

```

<template>
  <div id="app">
    <ejs-chart id="container" :primaryYAxis='primaryYAxis'>
      <e-series-collection>
        <e-series :dataSource='seriesData' type='Line' xName='x'
yName='y' name='Run Rates'> </e-series>
      </e-series-collection>
    </ejs-chart>
  </div>
</template>
<script>
import Vue from "vue";
import { ChartPlugin, ColumnSeries, StripLine, LineSeries } from
"@syncfusion/ej2-vue-charts";
Vue.use(ChartPlugin);
export default {
  data() {
    return {
      seriesData: [
        {x: 1, y: 5},{x: 2, y: 39},{x: 3, y: 21},{x: 4, y: 51},{x:
5, y: 30},
        {x: 6, y: 25},{x: 7, y: 10},{x: 8, y: 40},{x: 9, y: 50},{x:
10, y: 20}
      ],
      primaryYAxis:
      {
        minimum: 0,
        maximum: 60,
        interval: 10,
        stripLines:[
          { start: 30, size: 2, sizeType: 'Pixel', dashArray:"10,5",
color: "red"}
        ],
      },
    };
  },
  provide: {
    chart: [ColumnSeries, StripLine, LineSeries]
  },
};
</script>
<style>
#container {
  height: 350px;
}
</style>

```

{% previewsample "page.domainurl/code-snippet/chart/axis/strip-line-cs5" %}

### Recurrence Stripline

The strip lines to be drawn repeatedly at the regular intervals – this will be useful when you want to mark an event that occurs recursively along the timeline of the chart. Following properties are used to configure this feature.

- **isRepeat** - It is used for enable / disable the recurrence strip line.

- **repeatEvery** - It is used for mention the stripline interval.
- **repeatUntil** - It specifies the end value at which point strip line has to stop repeating.

### APP.VUE

```
<template>
  <div id="app">
    <ejs-chart id="container" :primaryXAxis='primaryXAxis'>
      <e-series-collection>
        <e-series :dataSource='seriesData' type='Line' xName='x'
yName='y' name='Run Rates'> </e-series>
      </e-series-collection>
    </ejs-chart>
  </div>
</template>
<script>
import Vue from "vue";
import { ChartPlugin, ColumnSeries, StripLine, LineSeries } from
"@syncfusion/ej2-vue-charts";
Vue.use(ChartPlugin);
export default {
  data() {
    return {
      seriesData: [
        {x: 1, y: 5},{x: 2, y: 39},{x: 3, y: 21},{x: 4, y: 51},{x:
5, y: 30},
        {x: 6, y: 25},{x: 7, y: 10},{x: 8, y: 40},{x: 9, y: 50},{x:
10, y: 20}
      ],
      primaryXAxis: {
        stripLines:[
          {start: 1, size: 1, isRepeat: true, repeatEvery: 2, color:
'rgba(167,169,171, 0.3)'}
        ],
      },
    };
  },
  provide: {
    chart: [ColumnSeries, StripLine, LineSeries]
  },
};
</script>
<style>
#container {
  height: 350px;
}
</style>
```

{% previewsample "page.domainurl/code-snippet/chart/axis/strip-line-cs6" %}

### Size Type

The **sizeType** property refers the size of the stripline. They are,

- **Auto**

- Pixel
- Years
- Months
- Days
- Hours
- Minutes
- Seconds

**APP.VUE**

```

<template>
  <div id="app">
    <ejs-chart id="container" :primaryXAxis='primaryXAxis'
:primaryYAxis='primaryYAxis'>
      <e-series-collection>
        <e-series :dataSource='seriesData' type='Line' xName='x'
yName='y' name='Run Rates'> </e-series>
      </e-series-collection>
    </ejs-chart>
  </div>
</template>
<script>
import Vue from "vue";
import { ChartPlugin, ColumnSeries, Category, LineSeries, DataLabel,
StripLine, DateTime } from "@syncfusion/ej2-vue-charts";
Vue.use(ChartPlugin);
export default {
  data() {
    return {
      seriesData: [
        { x: new Date(2000, 0, 1), y: 10 }, { x: new Date(2002, 0, 1),
y: 40 },
        { x: new Date(2004, 0, 1), y: 20 }, { x: new Date(2006, 0, 1),
y: 50 },
        { x: new Date(2008, 0, 1), y: 15 }, { x: new Date(2010, 0, 1),
y: 30 }
      ],
      primaryXAxis: {
        valueType: 'DateTime', intervalType: 'Years',
        stripLines:[
          {start:new Date(2002, 0, 1) , size: 2, sizeType: 'Years',
color: 'rgba(167,169,171, 0.3)'}
        ],
      },
      primaryYAxis: {
        minimum: 0, maximum: 60, interval: 10
      },
    };
  },
  provide: {
    chart: [ColumnSeries, Category, LineSeries, DataLabel, StripLine,
DateTime]
  },
};
</script>

```

```
<style>
  #container {
    height: 350px;
  }
</style>
```

{% previewsample "page.domainurl/code-snippet/chart/axis/strip-line-cs7" %}

### Segment Stripline

You can create the stripline in a particular region with respect to the segment. You can enable the segment stripline using `isSegmented` property. The start and end value of this type of stripline can be defined using `segmentStart` and `segmentEnd` properties.

- `isSegmented` - It is used for enable the segment stripline.
- `segmentStart` - Used to change the segment start value. Value correspond to associated axis.
- `segmentEnd` - Used to change the segment end value. Value correspond to associated axis.
- `segmentAxisName` - Used to specify the name of the associated axis.

### APP.VUE

```
<template>
  <div id="app">
    <ejs-chart id="container" :primaryYAxis='primaryYAxis'>
      <e-series-collection>
        <e-series :dataSource='seriesData' type='Line' xName='x'
yName='y' name='Run Rates'> </e-series>
      </e-series-collection>
    </ejs-chart>
  </div>
</template>
<script>
import Vue from "vue";
import { ChartPlugin, Chart, Category, ColumnSeries, LineSeries, DataLabel,
StripLine } from "@syncfusion/ej2-vue-charts";
Vue.use(ChartPlugin);
export default {
  data() {
    return {
      seriesData: [
        {x: 1, y: 5}, {x: 2, y: 39}, {x: 3, y: 21}, {x: 4, y: 51}, {x: 5,
y: 30},
        {x: 6, y: 25}, {x: 7, y: 10}, {x: 8, y: 40}, {x: 9, y: 50}, {x:
10, y: 20}
      ],
      primaryYAxis: {
        stripLines: [
          {start: 20, end: 40, isSegmented: true, segmentStart: 2,
segmentEnd: 4,
          color: 'rgba(167,169,171, 0.3)'}
        ],
      },
    };
  },
  provide: {
```



```

    chart: [ColumnSeries, Category, LineSeries, DataLabel, StripLine]
  },
};
</script>
<style>
#container {
  height: 350px;
}
</style>

```

{% previewsample "page.domainurl/code-snippet/chart/axis/strip-line-cs8" %}

## Multiple panes in Vue Chart component

Chart area can be divided into multiple panes using [rows](#) and [columns](#).

### Rows

To split the chart area vertically into number of rows, use [rows](#) property of the chart.

- You can allocate space for each row by using the [height](#) property. The value can be either in percentage or in pixel.
- To associate a vertical axis to a particular row, specify its index to

[rowIndex](#) property of the axis.

- To customize each row's bottom line, use [border](#) property.

### APP.VUE

```

<template>
  <div id="app">
    <ejs-chart id="container" :title='title'
:primaryXAxis='primaryXAxis' :primaryYAxis='primaryYAxis' :axes='axes'
:rows='rows'>
      <e-series-collection>
        <e-series :dataSource='seriesData' type='Column' xName='x'
yName='y' name='Germany'> </e-series>
        <e-series :dataSource='seriesData' type='Line' xName='x'
yName='y1' yAxisName='yAxis' name='Japan'
:marker='marker'> </e-series>
      </e-series-collection>
    </ejs-chart>
  </div>
</template>
<script>
import Vue from "vue";
import { ChartPlugin, ColumnSeries, LineSeries, Category } from
"@syncfusion/ej2-vue-charts";
Vue.use(ChartPlugin);
export default {
  data() {
    return {
      seriesData: [
        { x: 'Jan', y: 15, y1: 33 }, { x: 'Feb', y: 20, y1: 31 }, {
x: 'Mar', y: 35, y1: 30 },

```

```

      { x: 'Apr', y: 40, y1: 28 }, { x: 'May', y: 80, y1: 29 }, {
x: 'Jun', y: 70, y1: 30 },
      { x: 'Jul', y: 65, y1: 33 }, { x: 'Aug', y: 55, y1: 32 }, {
x: 'Sep', y: 50, y1: 34 },
      { x: 'Oct', y: 30, y1: 32 }, { x: 'Nov', y: 35, y1: 32 }, {
x: 'Dec', y: 35, y1: 31 }
    ],
    primaryXAxis: {
      title: 'Months',
      valueType: 'Category',
      interval: 1
    },
    primaryYAxis: {
      minimum: 0, maximum: 90, interval: 20,
      lineStyle: { width: 0 },
      title: 'Temperature (Fahrenheit)',
      labelFormat: '{value}°F'
    },
    axes:
    [
      {
        majorGridLines: { width: 0 },
        rowIndex: 1, opposedPosition: true,
        lineStyle: { width: 0 },
        minimum: 24, maximum: 36, interval: 4,
        name: 'yAxis', title: 'Temperature (Celsius)',
        labelFormat: '{value}°C'
      }
    ],
    rows:[
      {
        height: '50%'
      },{
        height: '50%'
      }
    ],
    marker: { visible: true, width: 10, height: 10, border: { width: 2,
color: '#F8AB1D' } },
    title: "Weather Condition"
  };
},
provide: {
  chart: [ColumnSeries, LineSeries, Category]
},
};
</script>
<style>
  #container {
    height: 350px;
  }
</style>

```

{% previewsample "page.domainurl/code-snippet/chart/axis/multiple-panes-cs1" %}

For spanning the vertical axis along multiple row, you can use [span](#) property of an axis.

**APP.VUE**

```

<template>
  <div id="app">
    <ejs-chart id="container" :title='title'
:primaryXAxis='primaryXAxis' :primaryYAxis='primaryYAxis' :axes='axes'
:rows='rows'>
      <e-series-collection>
        <e-series :dataSource='seriesData' type='Column' xName='x'
yName='y' name='Germany'> </e-series>
        <e-series :dataSource='seriesData' type='Line' xName='x'
yName='y1' yAxisName='yAxis' name='Japan'
:marker='marker'> </e-series>
      </e-series-collection>
    </ejs-chart>
  </div>
</template>
<script>
import Vue from "vue";
import { ChartPlugin, ColumnSeries, LineSeries, Category } from
"@syncfusion/ej2-vue-charts";
Vue.use(ChartPlugin);
export default {
  data() {
    return {
      seriesData: [
        { x: 'Jan', y: 15, y1: 33 }, { x: 'Feb', y: 20, y1: 31 }, {
x: 'Mar', y: 35, y1: 30 },
        { x: 'Apr', y: 40, y1: 28 }, { x: 'May', y: 80, y1: 29 }, {
x: 'Jun', y: 70, y1: 30 },
        { x: 'Jul', y: 65, y1: 33 }, { x: 'Aug', y: 55, y1: 32 }, {
x: 'Sep', y: 50, y1: 34 },
        { x: 'Oct', y: 30, y1: 32 }, { x: 'Nov', y: 35, y1: 32 }, {
x: 'Dec', y: 35, y1: 31 }
      ],
      primaryXAxis: {
        title: 'Months',
        valueType: 'Category',
        interval: 1
      },
      primaryYAxis: {
        minimum: 0, maximum: 90, interval: 10,
        lineStyle: { width: 0 },
        title: 'Temperature (Fahrenheit)',
        labelFormat: '{value}°F',
        //Span for chart axis
        span: 2
      },
      axes:
      [
        {
          majorGridLines: { width: 0 },
          rowIndex: 1, opposedPosition: true,
          lineStyle: { width: 0 },
          minimum: 24, maximum: 36, interval: 2,
          name: 'yAxis', title: 'Temperature (Celsius)',
          labelFormat: '{value}°C'
        }
      ]
    }
  }
}

```

```

    },
    rows: [
      {
        height: '50%'
      }, {
        height: '50%'
      }
    ],
    marker: { visible: true, width: 10, height: 10, border: { width: 2,
color: '#F8AB1D' } },
    title: "Weather Condition"
  };
},
provide: {
  chart: [ColumnSeries, LineSeries, Category]
},
};
</script>
<style>
  #container {
    height: 350px;
  }
</style>

```

{% previewsample "page.domainurl/code-snippet/chart/axis/multiple-panes-cs2" %}

### Columns

To split the chart area horizontally into number of columns, use [columns](#) property of the chart.

- You can allocate space for each column by using the [width](#) property. The given width can be either in percentage or in pixel.
- To associate a horizontal axis to a particular column, specify its index to

[columnIndex](#) property of the axis.

- To customize each column's bottom line, use [border](#)

property.

### APP.VUE

```

<template>
  <div id="app">
    <ejs-chart id="container" :title='title'
:primaryXAxis='primaryXAxis' :primaryYAxis='primaryYAxis' :axes='axes'
:columns='columns'>
      <e-series-collection>
        <e-series :dataSource='seriesData' type='Column' xName='x'
yName='y' name='Germany'> </e-series>
        <e-series :dataSource='seriesData' type='Line' xName='x'
yName='y1' xAxisName='xAxis' name='Japan'
          :marker='marker'> </e-series>
      </e-series-collection>
    </div>
  </template>

```

```

        </ejs-chart>
    </div>
</template>
<script>
import Vue from "vue";
import { ChartPlugin, ColumnSeries, LineSeries, Category } from
"@syncfusion/ej2-vue-charts";
Vue.use(ChartPlugin);
export default {
    data() {
        return {
            seriesData: [
                { x: 'Jan', y: 15, y1: 33 }, { x: 'Feb', y: 20, y1: 31 }, {
x: 'Mar', y: 35, y1: 30 },
                { x: 'Apr', y: 40, y1: 28 }, { x: 'May', y: 80, y1: 29 }, {
x: 'Jun', y: 70, y1: 30 },
                { x: 'Jul', y: 65, y1: 33 }, { x: 'Aug', y: 55, y1: 32 }, {
x: 'Sep', y: 50, y1: 34 },
                { x: 'Oct', y: 30, y1: 32 }, { x: 'Nov', y: 35, y1: 32 }, {
x: 'Dec', y: 35, y1: 31 }
            ],
            primaryXAxis: {
                title: 'Months',
                valueType: 'Category',
                interval: 1
            },
            primaryYAxis: {
                minimum: 0, maximum: 90, interval: 20,
                lineStyle: { width: 0 },
                title: 'Temperature (Fahrenheit)',
                labelFormat: '{value}°F'
            },
            columns:[
                {
                    width: '50%'
                },{
                    width: '50%'
                }
            ],
            axes:[
                {
                    majorGridLines: { width: 0 },
                    columnIndex: 1,
                    valueType: 'Category',
                    lineStyle: { width: 0 },
                    name: 'xAxis'
                }
            ],
            marker: { visible: true, width: 10, height: 10, border: { width: 2,
color: '#F8AB1D' } },
            title: "Weather Condition"
        };
    },
    provide: {
        chart: [ColumnSeries,LineSeries, Category]
    },
};

```

```

</script>
<style>
  #container {
    height: 350px;
  }
</style>

```

{% previewsample "page.domainurl/code-snippet/chart/axis/multiple-panes-cs3" %}

For spanning the horizontal axis along multiple column, you can use [span](#) property of an axis.

### APP.VUE

```

<template>
  <ejs-chart id="container" :title='title'
:primaryXAxis='primaryXAxis' :primaryYAxis='primaryYAxis' :axes='axes'
:columns='columns'>
    <e-series-collection>
      <e-series :dataSource='seriesData' type='Column' xName='x'
yName='y' name='Germany'> </e-series>
      <e-series :dataSource='seriesData' type='Line' xName='x'
yName='y1' xAxisName='xAxis' name='Japan'
:marker='marker'> </e-series>
    </e-series-collection>
  </ejs-chart>
</div>
</template>
<script>
import Vue from "vue";
import { ChartPlugin, ColumnSeries, LineSeries, Category } from
"@syncfusion/ej2-vue-charts";
Vue.use(ChartPlugin);
export default {
  data() {
    return {
      seriesData: [
        { x: 'Jan', y: 15, y1: 33 }, { x: 'Feb', y: 20, y1: 31 }, {
x: 'Mar', y: 35, y1: 30 },
        { x: 'Apr', y: 40, y1: 28 }, { x: 'May', y: 80, y1: 29 }, {
x: 'Jun', y: 70, y1: 30 },
        { x: 'Jul', y: 65, y1: 33 }, { x: 'Aug', y: 55, y1: 32 }, {
x: 'Sep', y: 50, y1: 34 },
        { x: 'Oct', y: 30, y1: 32 }, { x: 'Nov', y: 35, y1: 32 }, {
x: 'Dec', y: 35, y1: 31 }
      ],
      primaryXAxis: {
        title: 'Months',
        valueType: 'Category',
        interval: 1,
        span: 2
      },
      primaryYAxis: {
        minimum: 0, maximum: 90, interval: 20,
        lineStyle: { width: 0 },
        title: 'Temperature (Fahrenheit)',
        labelFormat: '{value}°F'
      },
    }
  },

```

```

        columns:[
            {
                width: '50%'
            },{
                width: '50%'
            }
        ],
        axes:[
            {
                majorGridLines: { width: 0 },
                columnIndex: 1,
                valueType: 'Category',
                lineStyle: { width: 0 },
                name: 'xAxis'
            }
        ],
        marker: { visible: true, width: 10, height: 10, border: { width: 2,
color: '#F8AB1D' } },
        title: "Weather Condition"
    };
    },
    provide: {
        chart: [ColumnSeries, LineSeries, Category]
    },
};
</script>
<style>
#container {
    height: 350px;
}
</style>

```

{% previewsample "page.domainurl/code-snippet/chart/axis/multiple-panes-cs4" %}

## Chart Types

### Line Chart in Vue Chart component

#### Line

To render a line series, use series [type](#) as `Line` and inject `LineSeries` into the `provide`.

#### APP.VUE

```

<template>
  <div id="app">
    <ejs-chart id="container" :title='title'>
      <e-series-collection>
        <e-series :dataSource='seriesData' type='Line' xName='x'
yName='y' name='India'> </e-series>
      </e-series-collection>
    </ejs-chart>
  </div>
</template>
<script>
import Vue from "vue";
import { ChartPlugin, LineSeries } from "@syncfusion/ej2-vue-charts";
Vue.use(ChartPlugin);

```

```

export default {
  data() {
    return {
      seriesData: [
        { x: 2005, y: 28 }, { x: 2006, y: 25 }, { x: 2007, y: 26 }, { x:
2008, y: 27 },
        { x: 2009, y: 32 }, { x: 2010, y: 35 }, { x: 2011, y: 30 }
      ],
      title: "Efficiency of oil-fired power production"
    };
  },
  provide: {
    chart: [LineSeries]
  }
};
</script>
<style>
  #container {
    height: 350px;
  }
</style>

```

{% previewsample "page.domainurl/code-snippet/chart/series/line-cs1" %}

### Multicolored Line

To render a multicolored line series, use the series type as **MultiColoredLine**, and inject the **MultiColoredLineSeries** into the **provide**. Here, the individual colors to the data can be mapped by using **pointColorMapping**.

### APP.VUE

```

<template>
  <div id="app">
    <ejs-chart id="container" :title='title'>
      <e-series-collection>
        <e-series :dataSource='seriesData' type='MultiColoredLine'
xName='x' yName='y' name='London' width=2
        :marker='marker' pointColorMapping= 'color'> </e-series>
      </e-series-collection>
    </ejs-chart>
  </div>
</template>
<script>
import Vue from "vue";
import { ChartPlugin, MultiColoredLineSeries } from "@syncfusion/ej2-vue-
charts";
Vue.use(ChartPlugin);
export default {
  data() {
    return {
      seriesData:[
        { x: 2005, y: 28 , color: 'red'}, { x: 2006, y: 25, color:'green'},
        { x: 2007, y: 26, color: '#ff0097' }, { x: 2008, y: 27, color:
'crimson' },

```



```

        { x: 2009, y: 32, color: 'blue' }, { x: 2010, y: 35, color:
'darkorange' }
    ],
    marker: { visible: true, width: 10, height: 10 },
    title: "Climate Graph-2012"
  };
},
provide: {
  chart: [MultiColoredLineSeries]
},
};
</script>
<style>
  #container {
    height: 350px;
  }
</style>

```

{% previewsample "page.domainurl/code-snippet/chart/series/line-cs7" %}

#### Series customization

The following properties can be used to customize the **line** series.

- [fill](#) – Specifies the color of the series.
- [opacity](#) – Specifies the opacity of [fill](#).
- [dashArray](#) – Specifies the dashes for series.
- [width](#) – Specifies the width for series.

#### APP.VUE

```

<template>
  <div id="app">
    <ejs-chart id="container" :title='title'>
      <e-series-collection>
        <e-series :dataSource='seriesData' type='Line' xName='x'
yName='y' name='India'
          fill='green' width=3 dashArray='5,5' :marker='marker'> </e-
series>
      </e-series-collection>
    </ejs-chart>
  </div>
</template>
<script>
import Vue from "vue";
import { ChartPlugin, LineSeries } from "@syncfusion/ej2-vue-charts";
Vue.use(ChartPlugin);
export default {
  data() {
    return {
      seriesData:[
        { x: 2005, y: 28 }, { x: 2006, y: 25 }, { x: 2007, y: 26 }, { x:
2008, y: 27 },
        { x: 2009, y: 32 }, { x: 2010, y: 35 }, { x: 2011, y: 30 }
      ],
      title: "Efficiency of oil-fired power production",

```

```

        marker: { visible: true },
    };
},
provide: {
    chart: [LineSeries]
},
};
</script>
<style>
#container {
    height: 350px;
}
</style>

```

{% previewsample "page.domainurl/code-snippet/chart/series/line-cs8" %}

[See Also](#)

- [Data label](#)
- [Tooltip](#)

Step line in Vue Chart component

[Step line](#)

To render a step line series, use series [type](#) as `StepLine` and inject `StepLineSeries` into the `provide`.

#### APP.VUE

```

<template>
  <div id="app">
    <ejs-chart id="container" :title='title'>
      <e-series-collection>
        <e-series :dataSource='seriesData' type='StepLine' xName='x'
yName='y' name='USA' width=2> </e-series>
      </e-series-collection>
    </ejs-chart>
  </div>
</template>
<script>
import Vue from "vue";
import { ChartPlugin, StepLineSeries } from "@syncfusion/ej2-vue-charts";
Vue.use(ChartPlugin);
export default {
  data() {
    return {
      seriesData:[
        { x: 2006, y: 378 }, { x: 2007, y: 416 },
        { x: 2008, y: 404 }, { x: 2009, y: 390 },
        { x: 2010, y: 376 }, { x: 2011, y: 365 }
      ],
      title: "CO2 - Intensity Analysis"
    };
  },
  provide: {

```

```

    chart: [StepLineSeries]
  },
};
</script>
<style>
  #container {
    height: 350px;
  }
</style>

```

{% previewsample "page.domainurl/code-snippet/chart/series/line-cs2" %}

#### Series customization

The following properties can be used to customize the **step line** series.

- [fill](#) – Specifies the color of the series.
- [opacity](#) – Specifies the opacity of [fill](#).
- [dashArray](#) – Specifies the dashes for series.
- [width](#) – Specifies the width for series.
- [step](#) – Specifies the position of the step for the series.

#### APP.VUE

```

<template>
  <div id="app">
    <ejs-chart id="container" :title='title'>
      <e-series-collection>
        <e-series :dataSource='seriesData' type='StepLine'
dashArray='2.5' :marker="marker" xName='x' yName='y' name='USA' width=2
fill="blue" :border='border' opacity=0.5 step="Left"> </e-series>
      </e-series-collection>
    </ejs-chart>
  </div>
</template>
<script>
import Vue from "vue";
import { ChartPlugin, StepLineSeries } from "@syncfusion/ej2-vue-charts";
Vue.use(ChartPlugin);
export default {
  data() {
    return {
      seriesData: [
        { x: 2006, y: 378 }, { x: 2007, y: 416 },
        { x: 2008, y: 404 }, { x: 2009, y: 390 },
        { x: 2010, y: 376 }, { x: 2011, y: 365 }
      ],
      border: { width: 1.5, color: 'brown' },
      title: "CO2 - Intensity Analysis",
      marker: {
        visible: true
      }
    };
  },
  provide: {
    chart: [StepLineSeries]
  }
};

```

```

    },
  };
</script>
<style>
  #container {
    height: 350px;
  }
</style>

```

{% previewsample "page.domainurl/code-snippet/chart/series/stepline-cs" %}

[See also](#)

- [Data label](#)
- [Tooltip](#)

## Stacked Line in Chart Vue Chart Component

### Stacked Line

To render a Stacked Line series, use series [type](#) as `StackingLine` and inject `StackingLineSeries` into the `provide`.

### APP.VUE

```

<template>
  <div id="app">
    <ejs-chart id="chartcontainer"
      :title="title"
      :primaryXAxis="primaryXAxis"
      :primaryYAxis="primaryYAxis"
      :tooltip="tooltip"
      :chartArea="chartArea">
      <e-series-collection>
        <e-series :dataSource="seriesData" type="StackingLine"
xName="x"
yName="y" name="John" width="2" dashArray="5,1"
:marker="marker" ></e-series>
        <e-series :dataSource="seriesData" type="StackingLine" xName="x"
yName="y1"
name="Peter" width="2" dashArray="5,1"
:marker="marker" ></e-series>
        <e-series :dataSource="seriesData" type="StackingLine" xName="x"
yName="y2"
name="Steve" width="2" dashArray="5,1" :marker="marker" ></e-
series>
        <e-series :dataSource="seriesData" type="StackingLine" xName="x"
yName="y3"
name="Charle" width="2" dashArray="5,1" :marker="marker"
></e-series>
      </e-series-collection>
    </ejs-chart>
  </div>
</template>
<script>
import Vue from "vue";
import { ChartPlugin, StackingLineSeries, Legend, Tooltip, Category } from
"@syncfusion/ej2-vue-charts";

```

```

Vue.use(ChartPlugin);
export default {
  data() {
    return {
      seriesData: [
        { x: "Food", y: 90, y1: 40, y2: 70, y3: 120 },
        { x: "Transport", y: 80, y1: 90, y2: 110, y3: 70 },
        { x: "Medical", y: 50, y1: 80, y2: 120, y3: 50 },
        { x: "Clothes", y: 70, y1: 30, y2: 60, y3: 180 },
        { x: "Personal Care", y: 30, y1: 80, y2: 80, y3: 30 },
        { x: "Books", y: 10, y1: 40, y2: 30, y3: 270 },
        { x: "Fitness", y: 100, y1: 30, y2: 70, y3: 40 },
        { x: "Electricity", y: 55, y1: 95, y2: 55, y3: 75 },
        { x: "Tax", y: 20, y1: 50, y2: 40, y3: 65 },
        { x: "Pet Care", y: 40, y1: 20, y2: 80, y3: 95 },
        { x: "Education", y: 45, y1: 15, y2: 45, y3: 195 },
        { x: "Entertainment", y: 75, y1: 45, y2: 65, y3: 115 }
      ],
      primaryXAxis: {
        majorGridLines: { width: 0 },
        minorGridLines: { width: 0 },
        majorTickLines: { width: 0 },
        minorTickLines: { width: 0 },
        interval: 1,
        lineStyle: { width: 0 },
        valueType: "Category"
      },
      primaryYAxis: {
        title: "Expense",
        lineStyle: { width: 0 },
        interval: 50,
        minorTickLines: { width: 0 },
        majorTickLines: { width: 0 },
        majorGridLines: { width: 1 },
        minorGridLines: { width: 1 }
      },
      chartArea: {
        border: {
          width: 0
        }
      },
      marker: {
        visible: true
      },
      tooltip: {
        enable: true,
        format: "${point.x} : <b>${point.y} (${point.percentage}%)</b>"
      },
      title: "Family Expense for Month"
    };
  },
  provide: {
    chart: [StackingLineSeries, Legend, Tooltip, Category]
  }
};
</script>
<style>
#container {

```

```

    height: 350px;
  }
</style>

```

{% previewsample "page.domainurl/code-snippet/chart/series/stack-line-cs1" %}

### Series customization

The following properties can be used to customize the **stacked line** series.

- [fill](#) – Specifies the color of the series.
- [opacity](#) – Specifies the opacity of [fill](#).
- [dashArray](#) – Specifies the dashes for series.
- [width](#) – Specifies the width for series.

### APP.VUE

```

<template>
  <div id="app">
    <ejs-chart id="chartcontainer"
      :title="title"
      :primaryXAxis="primaryXAxis"
      :primaryYAxis="primaryYAxis"
      :tooltip="tooltip"
      :chartArea="chartArea">
      <e-series-collection>
        <e-series :dataSource="seriesData" type="StackingLine"
          xName="x" yName="y" name="John" width="2" dashArray="5,1" :marker="marker"
          fill="red" ></e-series>
        <e-series :dataSource="seriesData" type="StackingLine"
          xName="x" yName="y1" name="Peter" width="2" dashArray="5,1" :marker="marker"
          fill="yellow" ></e-series>
        <e-series :dataSource="seriesData" type="StackingLine"
          xName="x" yName="y2" name="Steve" width="2" dashArray="5,1" :marker="marker"
          fill="green" ></e-series>
        <e-series :dataSource="seriesData" type="StackingLine"
          xName="x" yName="y3" name="Charle" width="2" dashArray="5,1"
          :marker="marker" fill="blue"></e-series>
      </e-series-collection>
    </ejs-chart>
  </div>
</template>
<script>
import Vue from "vue";
import { ChartPlugin, StackingLineSeries, Legend, Tooltip, Category } from
"@syncfusion/ej2-vue-charts";
Vue.use(ChartPlugin);
export default {
  data() {
    return {
      seriesData: [
        { x: "Food", y: 90, y1: 40, y2: 70, y3: 120 },
        { x: "Transport", y: 80, y1: 90, y2: 110, y3: 70 },
        { x: "Medical", y: 50, y1: 80, y2: 120, y3: 50 },
        { x: "Clothes", y: 70, y1: 30, y2: 60, y3: 180 },
        { x: "Personal Care", y: 30, y1: 80, y2: 80, y3: 30 },
      ]
    }
  }
}

```

```

        { x: "Books", y: 10, y1: 40, y2: 30, y3: 270 },
        { x: "Fitness", y: 100, y1: 30, y2: 70, y3: 40 },
        { x: "Electricity", y: 55, y1: 95, y2: 55, y3: 75 },
        { x: "Tax", y: 20, y1: 50, y2: 40, y3: 65 },
        { x: "Pet Care", y: 40, y1: 20, y2: 80, y3: 95 },
        { x: "Education", y: 45, y1: 15, y2: 45, y3: 195 },
        { x: "Entertainment", y: 75, y1: 45, y2: 65, y3: 115 }
    ],
    primaryXAxis: {
        majorGridLines: { width: 0 },
        minorGridLines: { width: 0 },
        majorTickLines: { width: 0 },
        minorTickLines: { width: 0 },
        interval: 1,
        lineStyle: { width: 0 },
        valueType: "Category"
    }, primaryYAxis: {
        title: "Expense",
        lineStyle: { width: 0 },
        interval: 50,
        minorTickLines: { width: 0 },
        majorTickLines: { width: 0 },
        majorGridLines: { width: 1 },
        minorGridLines: { width: 1 }
    },
    chartArea: {
        border: {
            width: 0
        }
    },
    marker: {
        visible: true
    },
    tooltip: {
        enable: true,
        format: "${point.x} : <b>${point.y} (${point.percentage}%)</b>"
    },
    title: "Family Expense for Month"
};
};
provide: {
    chart: [StackingLineSeries, Legend, Tooltip, Category]
}
};
</script>
<style>
    #container {
        height: 350px;
    }
</style>

```

{% previewsample "page.domainurl/code-snippet/chart/series/stack-line-cs2" %}

[See Also](#)

- [Data label](#)

- [Tooltip](#)

## 100% Stacked Line in Chart Vue Chart Component

### 100% Stacked line

To render a 100% Stacked Line series, use series [type](#) as `StackingLine100` and inject `StackingLineSeries` into the `provide`.

### APP.VUE

```
<template>
  <div id="app">
    <ejs-chart id="chartcontainer"
      :title="title"
      :primaryXAxis="primaryXAxis"
      :primaryYAxis="primaryYAxis"
      :tooltip="tooltip"
      :chartArea="chartArea">
      <e-series-collection>
        <e-series :dataSource="seriesData" type="StackingLine100"
xName="x" yName="y" name="John" width="2" dashArray="5,1"
:marker="marker" ></e-series>
        <e-series :dataSource="seriesData" type="StackingLine100"
xName="x" yName="y1" name="Peter" width="2" dashArray="5,1"
:marker="marker" ></e-series>
        <e-series :dataSource="seriesData" type="StackingLine100"
xName="x" yName="y2" name="Steve" width="2" dashArray="5,1" :marker="marker" ></e-
series>
        <e-series :dataSource="seriesData" type="StackingLine100"
xName="x" yName="y3" name="Charle" width="2" dashArray="5,1" :marker="marker"
></e-series>
      </e-series-collection>
    </ejs-chart>
  </div>
</template>
<script>
import Vue from "vue";
import { ChartPlugin, StackingLineSeries, Legend, Tooltip, Category } from
"@syncfusion/ej2-vue-charts";
Vue.use(ChartPlugin);
export default {
  data() {
    return {
      seriesData: [
        { x: "Food", y: 90, y1: 40, y2: 70, y3: 120 },
        { x: "Transport", y: 80, y1: 90, y2: 110, y3: 70 },
        { x: "Medical", y: 50, y1: 80, y2: 120, y3: 50 },
        { x: "Clothes", y: 70, y1: 30, y2: 60, y3: 180 },
        { x: "Personal Care", y: 30, y1: 80, y2: 80, y3: 30 },
        { x: "Books", y: 10, y1: 40, y2: 30, y3: 270 },
        { x: "Fitness", y: 100, y1: 30, y2: 70, y3: 40 },
        { x: "Electricity", y: 55, y1: 95, y2: 55, y3: 75 },
        { x: "Tax", y: 20, y1: 50, y2: 40, y3: 65 },
        { x: "Pet Care", y: 40, y1: 20, y2: 80, y3: 95 },
      ]
    }
  }
}
```



```

        { x: "Education", y: 45, y1: 15, y2: 45, y3: 195 },
        { x: "Entertainment", y: 75, y1: 45, y2: 65, y3: 115 }
    ],
    primaryXAxis: {
        majorGridLines: { width: 0 },
        minorGridLines: { width: 0 },
        majorTickLines: { width: 0 },
        minorTickLines: { width: 0 },
        interval: 1,
        lineStyle: { width: 0 },
        valueType: "Category"
    }, primaryYAxis: {
        title: "Expense",
        lineStyle: { width: 0 },
        interval: 20,
        minorTickLines: { width: 0 },
        majorTickLines: { width: 0 },
        majorGridLines: { width: 1 },
        minorGridLines: { width: 1 }
    },
    chartArea: {
        border: {
            width: 0
        }
    },
    marker: {
        visible: true
    },
    tooltip: {
        enable: true,
        format: "${point.x} : <b>${point.y} (${point.percentage}%)</b>"
    },
    title: "Family Expense for Month"
};
},
provide: {
    chart: [StackingLineSeries, Legend, Tooltip, Category]
}
};
</script>
<style>
    #container {
        height: 350px;
    }
</style>

```

{% previewsample "page.domainurl/code-snippet/chart/series/line-cs4" %}

#### Series customization

The following properties can be used to customize the 100% stacked line series.

- [fill](#) – Specifies the color of the series.
- [opacity](#) – Specifies the opacity of [fill](#).
- [dashArray](#) – Specifies the dashes for series.
- [width](#) – Specifies the width for series.

**APP.VUE**

```

<template>
  <div id="app">
    <ejs-chart id="chartcontainer"
      :title="title"
      :primaryXAxis="primaryXAxis"
      :primaryYAxis="primaryYAxis"
      :tooltip="tooltip"
      :chartArea="chartArea">
      <e-series-collection>
        <e-series :dataSource="seriesData" type="StackingLine100"
xName="x" yName="y" name="John" width="2" dashArray="5,1" :marker="marker"
fill="red" ></e-series>
        <e-series :dataSource="seriesData" type="StackingLine100"
xName="x" yName="y1" name="Peter" width="2" dashArray="5,1" :marker="marker"
fill="yellow" ></e-series>
        <e-series :dataSource="seriesData" type="StackingLine100"
xName="x" yName="y2" name="Steve" width="2" dashArray="5,1" :marker="marker"
fill="green" ></e-series>
        <e-series :dataSource="seriesData" type="StackingLine100"
xName="x" yName="y3" name="Charles" width="2" dashArray="5,1"
:marker="marker" fill="blue" ></e-series>
      </e-series-collection>
    </ejs-chart>
  </div>
</template>
<script>
import Vue from "vue";
import { ChartPlugin, StackingLineSeries, Legend, Tooltip, Category } from
"@syncfusion/ej2-vue-charts";
Vue.use(ChartPlugin);
export default {
  data() {
    return {
      seriesData: [
        { x: "Food", y: 90, y1: 40, y2: 70, y3: 120 },
        { x: "Transport", y: 80, y1: 90, y2: 110, y3: 70 },
        { x: "Medical", y: 50, y1: 80, y2: 120, y3: 50 },
        { x: "Clothes", y: 70, y1: 30, y2: 60, y3: 180 },
        { x: "Personal Care", y: 30, y1: 80, y2: 80, y3: 30 },
        { x: "Books", y: 10, y1: 40, y2: 30, y3: 270 },
        { x: "Fitness", y: 100, y1: 30, y2: 70, y3: 40 },
        { x: "Electricity", y: 55, y1: 95, y2: 55, y3: 75 },
        { x: "Tax", y: 20, y1: 50, y2: 40, y3: 65 },
        { x: "Pet Care", y: 40, y1: 20, y2: 80, y3: 95 },
        { x: "Education", y: 45, y1: 15, y2: 45, y3: 195 },
        { x: "Entertainment", y: 75, y1: 45, y2: 65, y3: 115 }
      ],
      primaryXAxis: {
        majorGridLines: { width: 0 },
        minorGridLines: { width: 0 },
        majorTickLines: { width: 0 },
        minorTickLines: { width: 0 },
        interval: 1,
        lineStyle: { width: 0 },
        valueType: "Category"
      }
    }
  }
}

```

```

    }, primaryYAxis: {
      title: "Expense",
      lineStyle: { width: 0 },
      interval: 20,
      minorTickLines: { width: 0 },
      majorTickLines: { width: 0 },
      majorGridLines: { width: 1 },
      minorGridLines: { width: 1 }
    },
    chartArea: {
      border: {
        width: 0
      }
    },
    marker: {
      visible: true
    },
    tooltip: {
      enable: true,
      format: "${point.x} : <b>${point.y} (${point.percentage}%)</b>"
    },
    title: "Family Expense for Month"
  };
},
provide: {
  chart: [StackingLineSeries, Legend, Tooltip, Category]
}
};
</script>
<style>
  #container {
    height: 350px;
  }
</style>

```

{% previewsample "page.domainurl/code-snippet/chart/series/stacked-line-cs" %}

[See Also](#)

- [Data label](#)
- [Tooltip](#)

### Spline in Vue Chart Component

#### [Spline](#)

To render a spline series, use series [type](#) as `Spline` and inject `SplineSeries` into the `provide`.

#### **APP.VUE**

```

<template>
  <div id="app">
    <ejs-chart id="container" :title='title'
    :primaryXAxis='primaryXAxis'>
      <e-series-collection>
        <e-series :dataSource='seriesData' type='Spline' xName='x'
        yName='y' name='London' width=2

```

```

        :marker='marker'> </e-series>
      </e-series-collection>
    </ejs-chart>
  </div>
</template>
<script>
import Vue from "vue";
import { ChartPlugin, SplineSeries, Category } from "@syncfusion/ej2-vue-
charts";
Vue.use(ChartPlugin);
export default {
  data() {
    return {
      seriesData:[
        { x: 'Jan', y: -1 }, { x: 'Feb', y: -1 }, { x: 'Mar', y: 2 },
        { x: 'Apr', y: 8 }, { x: 'May', y: 13 }, { x: 'Jun', y: 18 },
        { x: 'Jul', y: 21 }, { x: 'Aug', y: 20 }, { x: 'Sep', y: 16 },
        { x: 'Oct', y: 10 }, { x: 'Nov', y: 4 }, { x: 'Dec', y: 0 }
      ],
      primaryXAxis: {
        title: 'Month',
        valueType: 'Category'
      },
      marker: { visible: true, width: 10, height: 10 },
      title: "Climate Graph-2012"
    };
  },
  provide: {
    chart: [SplineSeries, Category]
  }
};
</script>
<style>
#container {
  height: 350px;
}
</style>

```

{% previewsample "page.domainurl/code-snippet/chart/series/line-cs5" %}

### Series customization

The following properties can be used to customize the **spline** series.

- [fill](#) – Specifies the color of the series.
- [opacity](#) – Specifies the opacity of [fill](#).
- [dashArray](#) – Specifies the dashes for series.
- [width](#) – Specifies the width for series.

### APP.VUE

```

<template>
  <div id="app">
    <ejs-chart id="container" :title='title'
    :primaryXAxis='primaryXAxis'>
      <e-series-collection>

```

```

        <e-series :dataSource='seriesData' type='Spline' xName='x'
yName='y' name='London' width=2
        :marker='marker' fill="red" :border="border"
dashArray='2.5'> </e-series>
    </e-series-collection>
</ejs-chart>
</div>
</template>
<script>
import Vue from "vue";
import { ChartPlugin, SplineSeries, Category } from "@syncfusion/ej2-vue-
charts";
Vue.use(ChartPlugin);
export default {
    data() {
        return {
            seriesData:[
                { x: 'Jan', y: -1 }, { x: 'Feb', y: -1 }, { x: 'Mar', y: 2 },
                { x: 'Apr', y: 8 }, { x: 'May', y: 13 }, { x: 'Jun', y: 18 },
                { x: 'Jul', y: 21 }, { x: 'Aug', y: 20 }, { x: 'Sep', y: 16 },
                { x: 'Oct', y: 10 }, { x: 'Nov', y: 4 }, { x: 'Dec', y: 0 }
            ],
            border:{
                width: 2, color:'yellow'
            },
            primaryXAxis: {
                title: 'Month',
                valueType: 'Category'
            },
            marker: { visible: true, width: 10, height: 10 },
            title: "Climate Graph-2012"
        };
    },
    provide: {
        chart: [SplineSeries, Category]
    }
};
</script>
<style>
#container {
    height: 350px;
}
</style>

```

{% previewsample "page.domainurl/code-snippet/chart/series/spline-cs" %}

[See Also](#)

- [Data label](#)
- [Tooltip](#)

## Area in Vue Chart Component

### Area

To render a area series, use series [type](#) as `Area` and inject `AreaSeries` into the `provide`.

**APP.VUE**

```

<template>
  <div id="app">
    <ejs-chart>
      <e-series-collection>
        <e-series :dataSource='seriesData' type='Area' xName='x'
yName='y' name='India'> </e-series>
      </e-series-collection>
    </ejs-chart>
  </div>
</template>
<script>
import Vue from "vue";
import { ChartPlugin, AreaSeries } from "@syncfusion/ej2-vue-charts";
Vue.use(ChartPlugin);
export default {
  data() {
    return {
      seriesData:[
        { x: 1900, y: 4 }, { x: 1920, y: 3.0 }, { x: 1940, y: 3.8 },
        { x: 1960, y: 3.4 }, { x: 1980, y: 3.2 }, { x: 2000, y: 3.9 }
      ]
    };
  },
  provide: {
    chart: [AreaSeries]
  },
};
</script>
<style>
  #container {
    height: 350px;
  }
</style>

```

{% previewsample "page.domainurl/code-snippet/chart/series/area-cs1" %}

*Multicolored area*

To render a multicolored area series, use the series type as **MultiColoredArea**, and inject the **MultiColoredAreaSeries** into the **provide**. The required **segments** of the series can be customized using the **value**, **color**, and **dashArray**.

**APP.VUE**

```

<template>
  <div id="app">
    <ejs-chart :title='title'>
      <e-series-collection>
        <e-series :dataSource='seriesData' type='MultiColoredArea'
xName='x' yName='y' name='England'
          segmentAxis='X' :segments='segments'> </e-series>
      </e-series-collection>
    </ejs-chart>
  </div>
</template>

```

```

<script>
import Vue from "vue";
import { ChartPlugin, MultiColoredAreaSeries } from "@syncfusion/ej2-vue-charts";
Vue.use(ChartPlugin);
export default {
  data() {
    return {
      seriesData:[
        { x: 2005, y: 28 }, { x: 2006, y: 25},
        { x: 2007, y: 26 }, { x: 2008, y: 27 },
        { x: 2009, y: 32}, { x: 2010, y: 35 },
        { x: 2011, y: 25 }
      ],
      title: 'England - Run Rate',
      segments: [{
        value: 2007,
        color: 'blue'
      }, {
        value: 2009,
        color: 'lightgreen'
      }, {
        color: 'orange'
      }
    ],
  };
},
provide: {
  chart: [MultiColoredAreaSeries]
}
};
</script>
<style>
#container {
  height: 350px;
}
</style>

```

{% previewsample "page.domainurl/code-snippet/chart/series/area-cs8" %}

### Series customization

The following properties can be used to customize the **area** series.

- [fill](#) – Specifies the color of the series.
- [opacity](#) – Specifies the opacity of [fill](#).
- [dashArray](#) – Specifies the dashes for series.
- [width](#) – Specifies the width for series.

### APP.VUE

```

<template>
  <div id="app">
    <ejs-chart id="container" :title='title'>
      <e-series-collection>
        <e-series :dataSource='seriesData' type='Area' xName='x'
yName='y' name='Product A'

```

```

        fill='green' width=2 dashArray='5,5' :border='border'
opacity=0.6> </e-series>
    </e-series-collection>
</ejs-chart>
</div>
</template>
<script>
import Vue from "vue";
import { ChartPlugin, AreaSeries } from "@syncfusion/ej2-vue-charts";
Vue.use(ChartPlugin);
export default {
    data() {
        return {
            seriesData:[
                { x: 1900, y: 4 }, { x: 1920, y: 3.0 }, { x: 1940, y: 3.8 },
                { x: 1960, y: 3.4 }, { x: 1980, y: 3.2 }, { x: 2000, y: 3.9 }
            ],
            border:{width:2, color:'Red'},
            title: "Average Sales Comparison"
        };
    },
    provide: {
        chart: [AreaSeries]
    },
};
</script>
<style>
#container {
    height: 350px;
}
</style>

```

{% previewsample "page.domainurl/code-snippet/chart/series/area-cs9" %}

#### Area border

The following properties in the **border** can be used to customize the border of the Area Chart.

- [width](#) - Specifies the width for the border of the Area Chart.
- [color](#) - Specifies the Color for the border of the Area Chart.

#### APP.VUE

```

<template>
    <div id="app">
        <ejs-chart id="container" :title='title'>
            <e-series-collection>
                <e-series :dataSource='seriesData' type='Area' xName='x'
yName='y' name='Product A'
                fill='green' width=2 :border='border' opacity=0.4> </e-
series>
            </e-series-collection>
        </ejs-chart>
    </div>
</template>
<script>

```



```

import Vue from "vue";
import { ChartPlugin, AreaSeries } from "@syncfusion/ej2-vue-charts";
Vue.use(ChartPlugin);
export default {
  data() {
    return {
      seriesData:[
        { x: 1900, y: 4 }, { x: 1920, y: 3.0 }, { x: 1940, y: 3.8 },
        { x: 1960, y: 3.4 }, { x: 1980, y: 3.2 }, { x: 2000, y: 3.9 }
      ],
      border:{width: 1.5 },
      title: "Average Sales Comparison"
    };
  },
  provide: {
    chart: [AreaSeries]
  },
};
</script>
<style>
#container {
  height: 350px;
}
</style>

```

{% previewsample "page.domainurl/code-snippet/chart/series/area-cs10" %}

[See Also](#)

- [Data label](#)
- [Tooltip](#)

## Range Area in Vue Chart Component

### [Range Area](#)

To render a range area series, use series [type](#) as `RangeArea` and inject `RangeAreaSeries` into the `provide`.

### **APP.VUE**

```

<template>
  <div id="app">
    <ejs-chart id="container" :title='title'
    :primaryXAxis='primaryXAxis' :primaryYAxis='primaryYAxis'>
      <e-series-collection>
        <e-series :dataSource='seriesData' type='RangeArea'
        xName='x' high='high' low='low'> </e-series>
      </e-series-collection>
    </ejs-chart>
  </div>
</template>
<script>
import Vue from "vue";
import { ChartPlugin, RangeAreaSeries, Category } from "@syncfusion/ej2-vue-
charts";
Vue.use(ChartPlugin);

```

```

export default {
  data() {
    return {
      seriesData:[
        { x: 'Jan', low: 0.7, high: 6.1 }, { x: 'Feb', low: 1.3, high:
6.3 },
        { x: 'Mar', low: 1.9, high: 8.5 }, { x: 'Apr', low: 3.1, high:
10.8 },
        { x: 'May', low: 5.7, high: 14.4 }, { x: 'June', low: 8.4, high:
16.9 },
        { x: 'July', low: 10.6, high: 19.2 }, { x: 'Aug', low: 10.5,
high: 18.9 },
        { x: 'Sep', low: 8.5, high: 16.1 }, { x: 'Oct', low: 6.0, high:
12.5 },
        { x: 'Nov', low: 1.5, high: 6.9 }, { x: 'Dec', low: 5.1, high:
12.1 }
      ],
      primaryXAxis: {
        title: 'Month', valueType: 'Category',
        edgeLabelPlacement: 'Shift'
      },
      primaryYAxis: {
        title: 'Temperature(Celsius)',
        minimum: 0, maximum: 20
      },
      title: "Maximum and Minimum Temperature"
    };
  },
  provide: {
    chart: [RangeAreaSeries, Category]
  },
};
</script>
<style>
  #container {
    height: 350px;
  }
</style>

```

{% previewsample "page.domainurl/code-snippet/chart/series/area-cs2" %}

### Series customization

The following properties can be used to customize the **range area** series.

- [fill](#) – Specifies the color of the series.
- [opacity](#) – Specifies the opacity of [fill](#).
- [dashArray](#) – Specifies the dashes for series.
- [width](#) – Specifies the width for series.

### APP.VUE

```

<template>
  <div id="app">
    <ejs-chart id="container" :title='title'>
      <e-series-collection>

```

```

        <e-series :dataSource='seriesData' type='RangeArea' xName='x'
high='high' low="low" name='Product A' fill='green'
        width=2 dashArray='5,5' :border='border' :marker='marker'
opacity=0.4> </e-series>
    </e-series-collection>
</ejs-chart>
</div>
</template>
<script>
import Vue from "vue";
import { ChartPlugin, RangeAreaSeries } from "@syncfusion/ej2-vue-charts";
Vue.use(ChartPlugin);
export default {
    data() {
        return {
            seriesData: [
                { x: 2000, low: 0.7, high: 6.1 }, { x: 2001, low: 1.3, high: 6.3 },
                { x: 2002, low: 1.9, high: 8.5 }, { x: 2003, low: 3.1, high: 10.8 },
                { x: 2004, low: 5.7, high: 14.4 }, { x: 2005, low: 8.4, high: 16.9 },
            ],
            { x: 2006, low: 10.6, high: 19.2 }, { x: 2007, low: 10.5, high: 18.9 },
            { x: 2008, low: 8.5, high: 16.1 }, { x: 2009, low: 6.0, high: 12.5 },
            { x: 2010, low: 1.5, high: 6.9 }, { x: 2011, low: 5.1, high: 12.1 }
        ],
            border: { width: 2, color: 'Red' },
            title: "Average Sales Comparison",
            marker: {
                visible: true
            }
        };
    },
    provide: {
        chart: [RangeAreaSeries]
    },
};
</script>
<style>
#container {
    height: 350px;
}
</style>

```

{% previewsample "page.domainurl/code-snippet/chart/series/rangearea-cs" %}

[See Also](#)

- [Data label](#)
- [Tooltip](#)

## Range step area in Vue Chart component

*Range step area*

To render a range step area series, use series [type](#) as `RangeStepArea` and inject `RangeStepAreaSeries` into the `provide`.

**APP.VUE**

```
<template>
  <div id="app">
    <ejs-chart id="container" :title='title'
:primaryXAxis='primaryXAxis' :primaryYAxis='primaryYAxis'>
      <e-series-collection>
        <e-series :dataSource='seriesData' type='RangeStepArea'
xName='x' high='high' low='low' name='England'> </e-series>
        <e-series :dataSource='seriesData1' type='RangeStepArea'
xName='x' high='high' low='low' name='India'> </e-series>
      </e-series-collection>
    </ejs-chart>
  </div>
</template>
<script>
import Vue from "vue";
import { ChartPlugin, RangeStepAreaSeries, Category } from "@syncfusion/ej2-
vue-charts";
Vue.use(ChartPlugin);
export default {
  data() {
    return {
      seriesData:[
        { x: 'Jan', high: 14, low: 4 },
        { x: 'Feb', high: 17, low: 7 },
        { x: 'Mar', high: 20, low: 10 },
        { x: 'Apr', high: 22, low: 12 },
        { x: 'May', high: 20, low: 10 },
        { x: 'Jun', high: 17, low: 7 },
        { x: 'Jul', high: 15, low: 5 },
        { x: 'Aug', high: 17, low: 7 },
        { x: 'Sep', high: 20, low: 10 },
        { x: 'Oct', high: 22, low: 12 },
        { x: 'Nov', high: 20, low: 10 },
        { x: 'Dec', high: 17, low: 7 }
      ],
      seriesData1:[
        { x: 'Jan', high: 29, low: 19 },
        { x: 'Feb', high: 32, low: 22 },
        { x: 'Mar', high: 35, low: 25 },
        { x: 'Apr', high: 37, low: 27 },
        { x: 'May', high: 35, low: 25 },
        { x: 'Jun', high: 32, low: 22 },
        { x: 'Jul', high: 30, low: 20 },
        { x: 'Aug', high: 32, low: 22 },
        { x: 'Sep', high: 35, low: 25 },
        { x: 'Oct', high: 37, low: 27 },
        { x: 'Nov', high: 35, low: 25 },
        { x: 'Dec', high: 32, low: 22 }
      ]
    }
  }
}
```

```

    primaryXAxis: {
      valueType: 'Category',
      edgeLabelPlacement: 'Shift',
      majorGridLines: { width: 0 }
    },
    primaryYAxis: {
      labelFormat: '{value}°C',
      lineStyle: { width: 0 },
      minimum: 0,
      maximum: 40,
      majorTickLines: { width: 0 }
    },
    title: 'Monthly Temperature Range'
  };
},
provide: {
  chart: [RangeStepAreaSeries, Category]
},
};
</script>
<style>
  #container {
    height: 350px;
  }
</style>

```

{% previewsample "page.domainurl/code-snippet/chart/series/area-cs11" %}

#### *Series customization*

The following properties can be used to customize the **range step area** series.

- [fill](#) – Specifies the color of the series.
- [opacity](#) – Specifies the opacity of [fill](#).
- [dashArray](#) – Specifies the dashes for series.
- [width](#) – Specifies the width for series.
- [step](#) – Specifies the position of the step for the series.

#### **APP.VUE**

```

<template>
  <div id="app">
    <ejs-chart id="container" :title='title'
    :primaryXAxis='primaryXAxis' :primaryYAxis='primaryYAxis'>
      <e-series-collection>
        <e-series :dataSource='seriesData' type='RangeStepArea'
        xName='x' high='high' low='low' name='England' :border="border" fill="red"
        opacity=0.4 width=2 dashArray="5.5" step="Center"></e-series>
        <e-series :dataSource='seriesData1' type='RangeStepArea'
        xName='x' high='high' low='low' name='India' :border="border" fill="blue"
        opacity=0.4 width=2 dashArray="5.5" step="Center"></e-series>
      </e-series-collection>
    </ejs-chart>
  </div>
</template>
<script>

```

```

import Vue from "vue";
import { ChartPlugin, RangeStepAreaSeries, Category } from "@syncfusion/ej2-vue-charts";
Vue.use(ChartPlugin);
export default {
  data() {
    return {
      seriesData:[
        { x: 'Jan', high: 14, low: 4 },
        { x: 'Feb', high: 17, low: 7 },
        { x: 'Mar', high: 20, low: 10 },
        { x: 'Apr', high: 22, low: 12 },
        { x: 'May', high: 20, low: 10 },
        { x: 'Jun', high: 17, low: 7 },
        { x: 'Jul', high: 15, low: 5 },
        { x: 'Aug', high: 17, low: 7 },
        { x: 'Sep', high: 20, low: 10 },
        { x: 'Oct', high: 22, low: 12 },
        { x: 'Nov', high: 20, low: 10 },
        { x: 'Dec', high: 17, low: 7 }
      ],
      seriesData1:[
        { x: 'Jan', high: 29, low: 19 },
        { x: 'Feb', high: 32, low: 22 },
        { x: 'Mar', high: 35, low: 25 },
        { x: 'Apr', high: 37, low: 27 },
        { x: 'May', high: 35, low: 25 },
        { x: 'Jun', high: 32, low: 22 },
        { x: 'Jul', high: 30, low: 20 },
        { x: 'Aug', high: 32, low: 22 },
        { x: 'Sep', high: 35, low: 25 },
        { x: 'Oct', high: 37, low: 27 },
        { x: 'Nov', high: 35, low: 25 },
        { x: 'Dec', high: 32, low: 22 }
      ],
      primaryXAxis: {
        valueType: 'Category',
        edgeLabelPlacement: 'Shift',
        majorGridLines: { width: 0 }
      },
      primaryYAxis: {
        labelFormat: '{value}°C',
        lineStyle: { width: 0 },
        minimum: 0,
        maximum: 40,
        majorTickLines: { width: 0 }
      },
      title: 'Monthly Temperature Range',
      border: {
        width: 2,
        color: 'Yellow'
      }
    };
  },
  provide: {
    chart: [RangeStepAreaSeries, Category]
  },

```

```
};
</script>
<style>
  #container {
    height: 350px;
  }
</style>
```

{% previewsample "page.domainurl/code-snippet/chart/series/area-cs12" %}

[See also](#)

- [Data label](#)
- [Tooltip](#)

## Spline Range Area in Vue Chart Component

### *Spline Range Area*

The Spline Range Area Chart is used to display continuous data points as a set of splines that vary between high and low values over intervals of time and across different categories.

To render a spline range area series, use series [type](#) as `SplineRangeArea` and inject `SplineRangeAreaSeries` into the `provide`.

### **APP.VUE**

```
<template>
  <div id="app">
    <ejs-chart id="container" :title='title'
:primaryXAxis='primaryXAxis' :primaryYAxis='primaryYAxis'>
      <e-series-collection>
        <e-series :dataSource='seriesData' type='SplineRangeArea'
xName='x' high='high' low='low' name='England'> </e-series>
        <e-series :dataSource='seriesData1' type='SplineRangeArea'
xName='x' high='high' low='low' name='India'> </e-series>
      </e-series-collection>
    </ejs-chart>
  </div>
</template>
<script>
import Vue from "vue";
import { ChartPlugin, SplineRangeAreaSeries, Category } from
"@syncfusion/ej2-vue-charts";
Vue.use(ChartPlugin);
export default {
  data() {
    return {
      seriesData:[
        { x: 'Jan', high: 14, low: 4 },
        { x: 'Feb', high: 17, low: 7 },
        { x: 'Mar', high: 20, low: 10 },
        { x: 'Apr', high: 22, low: 12 },
        { x: 'May', high: 20, low: 10 },
        { x: 'Jun', high: 17, low: 7 },
        { x: 'Jul', high: 15, low: 5 },
        { x: 'Aug', high: 17, low: 7 },
      ]
    }
  }
}
```

```

        { x: 'Sep', high: 20, low: 10 },
        { x: 'Oct', high: 22, low: 12 },
        { x: 'Nov', high: 20, low: 10 },
        { x: 'Dec', high: 17, low: 7 }
    ],
    seriesData1:[
        { x: 'Jan', high: 29, low: 19 },
        { x: 'Feb', high: 32, low: 22 },
        { x: 'Mar', high: 35, low: 25 },
        { x: 'Apr', high: 37, low: 27 },
        { x: 'May', high: 35, low: 25 },
        { x: 'Jun', high: 32, low: 22 },
        { x: 'Jul', high: 30, low: 20 },
        { x: 'Aug', high: 32, low: 22 },
        { x: 'Sep', high: 35, low: 25 },
        { x: 'Oct', high: 37, low: 27 },
        { x: 'Nov', high: 35, low: 25 },
        { x: 'Dec', high: 32, low: 22 }
    ],
    primaryXAxis: {
        valueType: 'Category',
        edgeLabelPlacement: 'Shift',
        majorGridLines: { width: 0 }
    },
    primaryYAxis: {
        labelFormat: '{value}°C',
        lineStyle: { width: 0 },
        minimum: 0,
        maximum: 40,
        majorTickLines: { width: 0 }
    },
    title: 'Monthly Temperature Range'
};
},
provide: {
    chart: [SplineRangeAreaSeries, Category]
},
};
</script>
<style>
    #container {
        height: 350px;
    }
</style>

```

{% previewsample "page.domainurl/code-snippet/chart/series/area-cs3" %}

#### Series customization

The following properties can be used to customize the **spline range area** series.

- [fill](#) – Specifies the color of the series.
- [opacity](#) – Specifies the opacity of [fill](#).
- [dashArray](#) – Specifies the dashes for series.
- [width](#) – Specifies the width for series.



**APP.VUE**

```

<template>
  <div id="app">
    <ejs-chart id="container" :title='title'
:primaryXAxis='primaryXAxis' :primaryYAxis='primaryYAxis'>
      <e-series-collection>
        <e-series :dataSource='seriesData' type='SplineRangeArea'
xName='x' high='high' low='low' name='England' fill="yellow"
:border="border" opacity=0.6 dashArray="5"> </e-series>
        <e-series :dataSource='seriesData1' type='SplineRangeArea'
xName='x' high='high' low='low' name='India' fill="red" :border="border"
opacity=0.6 dashArray="5"> </e-series>
      </e-series-collection>
    </ejs-chart>
  </div>
</template>
<script>
import Vue from "vue";
import { ChartPlugin, SplineRangeAreaSeries, Category } from
"@syncfusion/ej2-vue-charts";
Vue.use(ChartPlugin);
export default {
  data() {
    return {
      seriesData:[
        { x: 'Jan', high: 14, low: 4 },
        { x: 'Feb', high: 17, low: 7 },
        { x: 'Mar', high: 20, low: 10 },
        { x: 'Apr', high: 22, low: 12 },
        { x: 'May', high: 20, low: 10 },
        { x: 'Jun', high: 17, low: 7 },
        { x: 'Jul', high: 15, low: 5 },
        { x: 'Aug', high: 17, low: 7 },
        { x: 'Sep', high: 20, low: 10 },
        { x: 'Oct', high: 22, low: 12 },
        { x: 'Nov', high: 20, low: 10 },
        { x: 'Dec', high: 17, low: 7 }
      ],
      seriesData1:[
        { x: 'Jan', high: 29, low: 19 },
        { x: 'Feb', high: 32, low: 22 },
        { x: 'Mar', high: 35, low: 25 },
        { x: 'Apr', high: 37, low: 27 },
        { x: 'May', high: 35, low: 25 },
        { x: 'Jun', high: 32, low: 22 },
        { x: 'Jul', high: 30, low: 20 },
        { x: 'Aug', high: 32, low: 22 },
        { x: 'Sep', high: 35, low: 25 },
        { x: 'Oct', high: 37, low: 27 },
        { x: 'Nov', high: 35, low: 25 },
        { x: 'Dec', high: 32, low: 22 }
      ],
      primaryXAxis: {
        valueType: 'Category',
        edgeLabelPlacement: 'Shift',
        majorGridLines: { width: 0 }
      }
    }
  }
}

```

```

    },
    primaryYAxis: {
      labelFormat: '{value}°C',
      lineStyle: { width: 0 },
      minimum: 0,
      maximum: 40,
      majorTickLines: { width: 0 }
    },
    border:{
      width: 2, color: 'brown'
    },
    title: 'Monthly Temperature Range'
  };
},
provide: {
  chart: [SplineRangeAreaSeries, Category]
},
};
</script>
<style>
  #container {
    height: 350px;
  }
</style>

```

{% previewsample "page.domainurl/code-snippet/chart/series/splinerangearea-cs" %}

[See Also](#)

- [Data label](#)
- [Tooltip](#)

## Stacked Area in Vue Chart Component

### [Stacked Area](#)

To render a stacked area series, use series [type](#) as `StackingArea` and inject `StackingAreaSeries` into the `provide`.

### **APP.VUE**

```

<template>
  <div id="app">
    <ejs-chart :title='title' :primaryXAxis='primaryXAxis'>
      <e-series-collection>
        <e-series :dataSource='seriesData' type='StackingArea'
xName='x' yName='y' name='Organic'> </e-series>
        <e-series :dataSource='seriesData' type='StackingArea'
xName='x' yName='y1' name='Fair-trade'> </e-series>
        <e-series :dataSource='seriesData' type='StackingArea'
xName='x' yName='y2' name='Veg Alternatives'> </e-series>
        <e-series :dataSource='seriesData' type='StackingArea'
xName='x' yName='y3' name='Others'> </e-series>
      </e-series-collection>
    </ejs-chart>
  </div>
</template>

```

```

<script>
import Vue from "vue";
import { ChartPlugin, StackingAreaSeries, DateTime } from "@syncfusion/ej2-vue-charts";
Vue.use(ChartPlugin);
export default {
  data() {
    return {
      seriesData:[
        { x: new Date(2000, 0, 1), y: 0.61, y1: 0.03, y2: 0.48, y3: 0.23 },
        { x: new Date(2001, 0, 1), y: 0.81, y1: 0.05, y2: 0.53, y3: 0.17 },
        { x: new Date(2002, 0, 1), y: 0.91, y1: 0.06, y2: 0.57, y3: 0.17 },
        { x: new Date(2003, 0, 1), y: 1, y1: 0.09, y2: 0.61, y3: 0.20 },
        { x: new Date(2004, 0, 1), y: 1.19, y1: 0.14, y2: 0.63, y3: 0.23 },
        { x: new Date(2005, 0, 1), y: 1.47, y1: 0.20, y2: 0.64, y3: 0.36 },
        { x: new Date(2006, 0, 1), y: 1.74, y1: 0.29, y2: 0.66, y3: 0.43 },
        { x: new Date(2007, 0, 1), y: 1.98, y1: 0.46, y2: 0.76, y3: 0.52 },
        { x: new Date(2008, 0, 1), y: 1.99, y1: 0.64, y2: 0.77, y3: 0.72 },
        { x: new Date(2009, 0, 1), y: 1.70, y1: 0.75, y2: 0.55, y3: 1.29 },
        { x: new Date(2010, 0, 1), y: 1.48, y1: 1.06, y2: 0.54, y3: 1.38 },
        { x: new Date(2011, 0, 1), y: 1.38, y1: 1.25, y2: 0.57, y3: 1.82 },
        { x: new Date(2012, 0, 1), y: 1.66, y1: 1.55, y2: 0.61, y3: 2.16 },
        { x: new Date(2013, 0, 1), y: 1.66, y1: 1.55, y2: 0.67, y3: 2.51 },
        { x: new Date(2014, 0, 1), y: 1.67, y1: 1.65, y2: 0.67, y3: 2.61 }
      ],
      primaryXAxis: {
        valueType: 'DateTime'
      },
      title: "Trend in Sales of Ethical Produce"
    };
  },
  provide: {
    chart: [StackingAreaSeries, DateTime]
  },
};
</script>
<style>
#container {
  height: 350px;
}
</style>

```

{% previewsample "page.domainurl/code-snippet/chart/series/area-cs4" %}

### Series customization

The following properties can be used to customize the **stacked area** series.

- [fill](#) – Specifies the color of the series.
- [opacity](#) – Specifies the opacity of [fill](#).
- [dashArray](#) – Specifies the dashes for series.
- [width](#) – Specifies the width for series.

### APP.VUE

```
<template>
  <div id="app">
    <ejs-chart :title='title' :primaryXAxis='primaryXAxis'>
      <e-series-collection>
        <e-series :dataSource='seriesData' type='StackingArea'
xName='x' yName='y' name='Organic' fill="red" :border="border" dashArray="5"
opacity=0.7> </e-series>
        <e-series :dataSource='seriesData' type='StackingArea'
xName='x' yName='y1' name='Fair-trade' fill="yellow" :border="border"
dashArray="5" opacity=0.7> </e-series>
        <e-series :dataSource='seriesData' type='StackingArea'
xName='x' yName='y2' name='Veg Alternatives' fill="green" :border="border"
dashArray="5" opacity=0.7> </e-series>
        <e-series :dataSource='seriesData' type='StackingArea'
xName='x' yName='y3' name='Others' fill="blue" :border="border"
dashArray="5" opacity=0.7> </e-series>
      </e-series-collection>
    </ejs-chart>
  </div>
</template>
<script>
import Vue from "vue";
import { ChartPlugin, StackingAreaSeries, DateTime } from "@syncfusion/ej2-
vue-charts";
Vue.use(ChartPlugin);
export default {
  data() {
    return {
      seriesData:[
        { x: new Date(2000, 0, 1), y: 0.61, y1: 0.03, y2: 0.48, y3:
0.23 },
        { x: new Date(2001, 0, 1), y: 0.81, y1: 0.05, y2: 0.53, y3:
0.17 },
        { x: new Date(2002, 0, 1), y: 0.91, y1: 0.06, y2: 0.57, y3:
0.17 },
        { x: new Date(2003, 0, 1), y: 1, y1: 0.09, y2: 0.61, y3: 0.20
},
        { x: new Date(2004, 0, 1), y: 1.19, y1: 0.14, y2: 0.63, y3:
0.23 },
        { x: new Date(2005, 0, 1), y: 1.47, y1: 0.20, y2: 0.64, y3:
0.36 },
      ]
    }
  }
}
```

```

0.43 },
0.52 },
0.72 },
1.29 },
1.38 },
1.82 },
2.16 },
2.51 },
2.61 }
    ],
    primaryXAxis: {
        valueType: 'DateTime'
    },
    border:{
        width: 2, color: 'white'
    },
    title: "Trend in Sales of Ethical Produce"
};
},
provide: {
    chart: [StackingAreaSeries, DateTime]
},
};
</script>
<style>
#container {
    height: 350px;
}
</style>

```

{% previewsample "page.domainurl/code-snippet/chart/series/stackarea-cs" %}

[See Also](#)

- [Data label](#)
- [Tooltip](#)

## 100% Stacked Area in Vue Chart Component

### 100% Stacked Area

To render a 100% stacked area series, use series [type](#) as `StackingArea100` and inject `StackingAreaSeries` into the `provide`.

### APP.VUE

```

<template>
  <div id="app">

```

```

    <ejs-chart id="container" :title='title'
:primaryXAxis='primaryXAxis'>
      <e-series-collection>
        <e-series :dataSource='seriesData' type='StackingArea100'
xName='x' yName='y' name='USA'> </e-series>
        <e-series :dataSource='seriesData' type='StackingArea100'
xName='x' yName='y1' name='UK'> </e-series>
        <e-series :dataSource='seriesData' type='StackingArea100'
xName='x' yName='y2' name='Canada'> </e-series>
        <e-series :dataSource='seriesData' type='StackingArea100'
xName='x' yName='y3' name='China'> </e-series>
      </e-series-collection>
    </ejs-chart>
  </div>
</template>
<script>
import Vue from "vue";
import { ChartPlugin, StackingAreaSeries, DateTime } from "@syncfusion/ej2-
vue-charts";
Vue.use(ChartPlugin);
export default {
  data() {
    return {
      seriesData:[
        { x: new Date(2006, 0, 1), y: 34, y1: 51, y2: 14, y3: 37 },
        { x: new Date(2007, 0, 1), y: 20, y1: 26, y2: 34, y3: 15 },
        { x: new Date(2008, 0, 1), y: 40, y1: 37, y2: 73, y3: 53 },
        { x: new Date(2009, 0, 1), y: 51, y1: 51, y2: 51, y3: 51 },
        { x: new Date(2010, 0, 1), y: 26, y1: 26, y2: 26, y3: 26 },
        { x: new Date(2011, 0, 1), y: 37, y1: 37, y2: 37, y3: 37 },
        { x: new Date(2012, 0, 1), y: 54, y1: 43, y2: 12, y3: 54 },
        { x: new Date(2013, 0, 1), y: 44, y1: 23, y2: 16, y3: 44 },
        { x: new Date(2014, 0, 1), y: 48, y1: 55, y2: 34, y3: 23 }
      ],
      primaryXAxis: {
        valueType: 'DateTime'
      },
      title: "Annual Temperature Comparisone"
    };
  },
  provide: {
    chart: [StackingAreaSeries, DateTime]
  },
};
</script>
<style>
#container {
  height: 350px;
}
</style>

```

{% previewsample "page.domainurl/code-snippet/chart/series/area-cs5" %}

#### Series customization

The following properties can be used to customize the 100% stacked area series.

- [fill](#) – Specifies the color of the series.
- [opacity](#) – Specifies the opacity of [fill](#).
- [dashArray](#) – Specifies the dashes for series.
- [width](#) – Specifies the width for series.

**APP.VUE**

```

<template>
  <div id="app">
    <ejs-chart id="container" :title='title'
:primaryXAxis='primaryXAxis'>
      <e-series-collection>
        <e-series :dataSource='seriesData' type='StackingArea100'
xName='x' yName='y' name='USA' opacity=0.7 fill="red" :border="border"
dashArray="5"> </e-series>
        <e-series :dataSource='seriesData' type='StackingArea100'
xName='x' yName='y1' name='UK' opacity=0.7 fill="yellow" :border="border"
dashArray="5"> </e-series>
        <e-series :dataSource='seriesData' type='StackingArea100'
xName='x' yName='y2' name='Canada' opacity=0.7 fill="green" :border="border"
dashArray="5"> </e-series>
        <e-series :dataSource='seriesData' type='StackingArea100'
xName='x' yName='y3' name='China' opacity=0.7 fill="blue" :border="border"
dashArray="5"> </e-series>
      </e-series-collection>
    </ejs-chart>
  </div>
</template>
<script>
import Vue from "vue";
import { ChartPlugin, StackingAreaSeries, DateTime } from "@syncfusion/ej2-
vue-charts";
Vue.use(ChartPlugin);
export default {
  data() {
    return {
      seriesData:[
        { x: new Date(2006, 0, 1), y: 34, y1: 51, y2: 14, y3: 37 },
        { x: new Date(2007, 0, 1), y: 20, y1: 26, y2: 34, y3: 15 },
        { x: new Date(2008, 0, 1), y: 40, y1: 37, y2: 73, y3: 53 },
        { x: new Date(2009, 0, 1), y: 51, y1: 51, y2: 51, y3: 51 },
        { x: new Date(2010, 0, 1), y: 26, y1: 26, y2: 26, y3: 26 },
        { x: new Date(2011, 0, 1), y: 37, y1: 37, y2: 37, y3: 37 },
        { x: new Date(2012, 0, 1), y: 54, y1: 43, y2: 12, y3: 54 },
        { x: new Date(2013, 0, 1), y: 44, y1: 23, y2: 16, y3: 44 },
        { x: new Date(2014, 0, 1), y: 48, y1: 55, y2: 34, y3: 23 }
      ],
      primaryXAxis: {
        valueType: 'DateTime'
      },
      title: "Annual Temperature Comparisone",
      border:{
        width: 2,
        color: 'white'
      }
    }
  };

```

```

    },
    provide: {
      chart: [StackingAreaSeries, DateTime]
    },
  };
</script>
<style>
  #container {
    height: 350px;
  }
</style>

```

{% previewsample "page.domainurl/code-snippet/chart/series/stackedarea-cs" %}

[See Also](#)

- [Data label](#)
- [Tooltip](#)

Stacked step area in Vue Chart component

[Stacked step area](#)

To render a stacked step area series, use series [type](#) as `StackingStepArea` and inject `StackingStepAreaSeries` into the `provide`.

#### APP.VUE

```

<template>
  <div id="app">
    <ejs-chart :title='title' :primaryXAxis='primaryXAxis'>
      <e-series-collection>
        <e-series :dataSource='seriesData' type='StackingStepArea'
xName='x' yName='y' name='Organic'> </e-series>
        <e-series :dataSource='seriesData' type='StackingStepArea'
xName='x' yName='y1' name='Fair-trade'> </e-series>
        <e-series :dataSource='seriesData' type='StackingStepArea'
xName='x' yName='y2' name='Veg Alternatives'> </e-series>
        <e-series :dataSource='seriesData' type='StackingStepArea'
xName='x' yName='y3' name='Others'> </e-series>
      </e-series-collection>
    </ejs-chart>
  </div>
</template>
<script>
import Vue from "vue";
import { ChartPlugin, StackingStepAreaSeries, DateTime } from
"@syncfusion/ej2-vue-charts";
Vue.use(ChartPlugin);
export default {
  data() {
    return {
      seriesData:[
        { x: new Date(2000, 0, 1), y: 0.61, y1: 0.03, y2: 0.48, y3: 0.23
},
        { x: new Date(2001, 0, 1), y: 0.81, y1: 0.05, y2: 0.53, y3: 0.17
},

```



```

    { x: new Date(2002, 0, 1), y: 0.91, y1: 0.06, y2: 0.57, y3: 0.17
  },
    { x: new Date(2003, 0, 1), y: 1, y1: 0.09, y2: 0.61, y3: 0.20 },
    { x: new Date(2004, 0, 1), y: 1.19, y1: 0.14, y2: 0.63, y3: 0.23
  },
    { x: new Date(2005, 0, 1), y: 1.47, y1: 0.20, y2: 0.64, y3: 0.36
  },
    { x: new Date(2006, 0, 1), y: 1.74, y1: 0.29, y2: 0.66, y3: 0.43
  },
    { x: new Date(2007, 0, 1), y: 1.98, y1: 0.46, y2: 0.76, y3: 0.52
  },
    { x: new Date(2008, 0, 1), y: 1.99, y1: 0.64, y2: 0.77, y3: 0.72
  },
    { x: new Date(2009, 0, 1), y: 1.70, y1: 0.75, y2: 0.55, y3: 1.29
  },
    { x: new Date(2010, 0, 1), y: 1.48, y1: 1.06, y2: 0.54, y3: 1.38
  },
    { x: new Date(2011, 0, 1), y: 1.38, y1: 1.25, y2: 0.57, y3: 1.82
  },
    { x: new Date(2012, 0, 1), y: 1.66, y1: 1.55, y2: 0.61, y3: 2.16
  },
    { x: new Date(2013, 0, 1), y: 1.66, y1: 1.55, y2: 0.67, y3: 2.51
  },
    { x: new Date(2014, 0, 1), y: 1.67, y1: 1.65, y2: 0.67, y3: 2.61
  }
  ],
  primaryXAxis: {
    valueType: 'DateTime'
  },
  title: "Trend in Sales of Ethical Produce"
};
},
provide: {
  chart: [StackingStepAreaSeries, DateTime]
},
};
</script>
<style>
#container {
  height: 350px;
}
</style>

```

{% previewsample "page.domainurl/code-snippet/chart/series/stackedsteparea" %}

#### Series customization

The following properties can be used to customize the **stacked step area** series.

- [fill](#) – Specifies the color of the series.
- [opacity](#) – Specifies the opacity of [fill](#).
- [dashArray](#) – Specifies the dashes for series.
- [width](#) – Specifies the width for series.
- [step](#) – Specifies the position of the step for the series.

**APP.VUE**

```

<template>
  <div id="app">
    <ejs-chart :title='title' :primaryXAxis='primaryXAxis'>
      <e-series-collection>
        <e-series :dataSource='seriesData' type='StackingStepArea'
xName='x' yName='y' name='Organic' fill="red" :border="border"
dashArray="5.5" opacity=0.6 width=2 step="Center"> </e-series>
        <e-series :dataSource='seriesData' type='StackingStepArea'
xName='x' yName='y1' name='Fair-trade' fill="yellow" :border="border"
dashArray="5.5" opacity=0.6 width=2 step="Center"> </e-series>
        <e-series :dataSource='seriesData' type='StackingStepArea'
xName='x' yName='y2' name='Veg Alternatives' fill="green" :border="border"
dashArray="5.5" opacity=0.6 width=2 step="Center"> </e-series>
        <e-series :dataSource='seriesData' type='StackingStepArea'
xName='x' yName='y3' name='Others' fill="blue" :border="border"
dashArray="5.5" opacity=0.6 width=2 step="Center"> </e-series>
      </e-series-collection>
    </ejs-chart>
  </div>
</template>
<script>
import Vue from "vue";
import { ChartPlugin, StackingStepAreaSeries, DateTime } from
"@syncfusion/ej2-vue-charts";
Vue.use(ChartPlugin);
export default {
  data() {
    return {
      seriesData:[
        { x: new Date(2000, 0, 1), y: 0.61, y1: 0.03, y2: 0.48, y3: 0.23
},
        { x: new Date(2001, 0, 1), y: 0.81, y1: 0.05, y2: 0.53, y3: 0.17
},
        { x: new Date(2002, 0, 1), y: 0.91, y1: 0.06, y2: 0.57, y3: 0.17
},
        { x: new Date(2003, 0, 1), y: 1, y1: 0.09, y2: 0.61, y3: 0.20 },
        { x: new Date(2004, 0, 1), y: 1.19, y1: 0.14, y2: 0.63, y3: 0.23
},
        { x: new Date(2005, 0, 1), y: 1.47, y1: 0.20, y2: 0.64, y3: 0.36
},
        { x: new Date(2006, 0, 1), y: 1.74, y1: 0.29, y2: 0.66, y3: 0.43
},
        { x: new Date(2007, 0, 1), y: 1.98, y1: 0.46, y2: 0.76, y3: 0.52
},
        { x: new Date(2008, 0, 1), y: 1.99, y1: 0.64, y2: 0.77, y3: 0.72
},
        { x: new Date(2009, 0, 1), y: 1.70, y1: 0.75, y2: 0.55, y3: 1.29
},
        { x: new Date(2010, 0, 1), y: 1.48, y1: 1.06, y2: 0.54, y3: 1.38
},
        { x: new Date(2011, 0, 1), y: 1.38, y1: 1.25, y2: 0.57, y3: 1.82
},
        { x: new Date(2012, 0, 1), y: 1.66, y1: 1.55, y2: 0.61, y3: 2.16
},
      ],
    }
  }
}

```

```

        { x: new Date(2013, 0, 1), y: 1.66, y1: 1.55, y2: 0.67, y3: 2.51
    },
        { x: new Date(2014, 0, 1), y: 1.67, y1: 1.65, y2: 0.67, y3: 2.61 }
    ],
    primaryXAxis: {
        valueType: 'DateTime'
    },
    title: "Trend in Sales of Ethical Produce",
    border:{
        width: 2,
        color: 'black'
    }
};
},
provide: {
    chart: [StackingStepAreaSeries, DateTime]
},
};
</script>
<style>
#container {
    height: 350px;
}
</style>

```

{% previewsample "page.domainurl/code-snippet/chart/series/stackedsteparea-cs" %}

[See also](#)

- [Data label](#)
- [Tooltip](#)

Step area in Vue Chart component

[Step area](#)

To render a step area series, use series [type](#) as `StepArea` and inject `StepAreaSeries` into the `provide`.

#### APP.VUE

```

<template>
    <div id="app">
        <ejs-chart id="container" :title='title'
:primaryXAxis='primaryXAxis' :primaryYAxis='primaryYAxis'>
            <e-series-collection>
                <e-series :dataSource='seriesData' type='StepArea' xName='x'
yName='y' name='India'> </e-series>
            </e-series-collection>
        </ejs-chart>
    </div>
</template>
<script>
import Vue from "vue";
import { ChartPlugin, StepAreaSeries } from "@syncfusion/ej2-vue-charts";
Vue.use(ChartPlugin);
export default {
    data() {

```

```

return {
  seriesData:[
    { x: 1, y: 7 }, { x: 2, y: 1 }, { x: 3, y: 1 },
    { x: 4, y: 14 }, { x: 5, y: 1 }, { x: 6, y: 10 },
    { x: 7, y: 8 }, { x: 8, y: 6 }, { x: 9, y: 10 },
    { x: 10, y: 10 }, { x: 11, y: 16 }, { x: 12, y: 6 },
    { x: 13, y: 14 }, { x: 14, y: 7 }, { x: 15, y: 5 },
    { x: 16, y: 2 }, { x: 17, y: 14 }, { x: 18, y: 7 },
    { x: 19, y: 7 }, { x: 20, y: 10 }
  ],
  primaryXAxis: {
    valueType: 'Double',
    title: 'Overs'
  },
  primaryYAxis: {
    title: 'Runs'
  },
  title: 'England - Run Rate',
};
},
provide: {
  chart: [StepAreaSeries]
}
};
</script>
<style>
#container {
  height: 350px;
}
</style>

```

{% previewsample "page.domainurl/code-snippet/chart/series/area-cs6" %}

### Series customization

The following properties can be used to customize the **step area** series.

- [fill](#) – Specifies the color of the series.
- [opacity](#) – Specifies the opacity of [fill](#).
- [dashArray](#) – Specifies the dashes for series.
- [width](#) – Specifies the width for series.
- [step](#) – Specifies the position of the step for the series.

### APP.VUE

```

<template>
  <div id="app">
    <ejs-chart id="container" :title='title'
:primaryXAxis='primaryXAxis' :primaryYAxis='primaryYAxis'>
      <e-series-collection>
        <e-series :dataSource='seriesData' type='StepArea' xName='x'
yName='y' name='India' fill="blue" :border="border" dashArray="5"
opacity=0.6 :marker='marker' width=2 step="Right"> </e-series>
      </e-series-collection>
    </ejs-chart>
  </div>

```

```

</template>
<script>
import Vue from "vue";
import { ChartPlugin, StepAreaSeries } from "@syncfusion/ej2-vue-charts";
Vue.use(ChartPlugin);
export default {
  data() {
    return {
      seriesData: [
        { x: 1, y: 7 }, { x: 2, y: 1 }, { x: 3, y: 1 },
        { x: 4, y: 14 }, { x: 5, y: 1 }, { x: 6, y: 10 },
        { x: 7, y: 8 }, { x: 8, y: 6 }, { x: 9, y: 10 },
        { x: 10, y: 10 }, { x: 11, y: 16 }, { x: 12, y: 6 },
        { x: 13, y: 14 }, { x: 14, y: 7 }, { x: 15, y: 5 },
        { x: 16, y: 2 }, { x: 17, y: 14 }, { x: 18, y: 7 },
        { x: 19, y: 7 }, { x: 20, y: 10 }
      ],
      primaryXAxis: {
        valueType: 'Double',
        title: 'Overs'
      },
      primaryYAxis: {
        title: 'Runs'
      },
      title: 'England - Run Rate',
      border: {
        width: 2,
        color: 'black'
      },
      marker: { visible: true }
    };
  },
  provide: {
    chart: [StepAreaSeries]
  }
};
</script>
<style>
  #container {
    height: 350px;
  }
</style>

```

{% previewsample "page.domainurl/code-snippet/chart/series/steparea-cs" %}

[See also](#)

- [Data label](#)
- [Tooltip](#)

## Spline Area in Vue Chart Component

### [Spline Area](#)

To render a step area series, use series [type](#) as `SplineArea` and inject `SplineAreaSeries` into the `provide`.

**APP.VUE**

```

<template>
  <div id="app">
    <ejs-chart id="container" :title='title'
:primaryXAxis='primaryXAxis' :primaryYAxis='primaryYAxis'>
      <e-series-collection>
        <e-series :dataSource='seriesData' type='SplineArea'
xName='x' yName='y' name='India'> </e-series>
      </e-series-collection>
    </ejs-chart>
  </div>
</template>
<script>
import Vue from "vue";
import { ChartPlugin, SplineAreaSeries } from "@syncfusion/ej2-vue-charts";
Vue.use(ChartPlugin);
export default {
  data() {
    return {
      seriesData:[
        { x: 1, y: 7 }, { x: 2, y: 1 }, { x: 3, y: 1 },
        { x: 4, y: 14 }, { x: 5, y: 1 }, { x: 6, y: 10 },
        { x: 7, y: 8 }, { x: 8, y: 6 }, { x: 9, y: 10 },
        { x: 10, y: 10 }, { x: 11, y: 16 }, { x: 12, y: 6 },
        { x: 13, y: 14 }, { x: 14, y: 7 }, { x: 15, y: 5 },
        { x: 16, y: 2 }, { x: 17, y: 14 }, { x: 18, y: 7 },
        { x: 19, y: 7 }, { x: 20, y: 10 }
      ],
      primaryXAxis: {
        valueType: 'Double',
        title: 'Overs'
      },
      primaryYAxis: {
        title: 'Runs'
      },
      title: 'England - Run Rate'
    };
  },
  provide: {
    chart: [SplineAreaSeries]
  }
};
</script>
<style>
  #container {
    height: 350px;
  }
</style>

```

{% previewsample "page.domainurl/code-snippet/chart/series/splinearea" %}

*Series customization*

The following properties can be used to customize the **spline area** series.

- [fill](#) – Specifies the color of the series.

- [opacity](#) – Specifies the opacity of [fill](#).
- [dashArray](#) – Specifies the dashes for series.
- [width](#) – Specifies the width for series.

### APP.VUE

```

<template>
  <div id="app">
    <ejs-chart id="container" :title='title' :primaryXAxis='primaryXAxis'
:primaryYAxis='primaryYAxis'>
      <e-series-collection>
        <e-series :dataSource='seriesData' type='SplineArea' xName='x'
yName='y' name='India' fill="red" dashArray="5" :border="border"
:marker='marker' opacity='0.6'> </e-series>
      </e-series-collection>
    </ejs-chart>
  </div>
</template>
<script>
import Vue from "vue";
import { ChartPlugin, SplineAreaSeries } from "@syncfusion/ej2-vue-charts";
Vue.use(ChartPlugin);
export default {
  data() {
    return {
      seriesData:[
        { x: 1, y: 7 }, { x: 2, y: 1 }, { x: 3, y: 1 },
        { x: 4, y: 14 }, { x: 5, y: 1 }, { x: 6, y: 10 },
        { x: 7, y: 8 }, { x: 8, y: 6 }, { x: 9, y: 10 },
        { x: 10, y: 10 }, { x: 11, y: 16 }, { x: 12, y: 6 },
        { x: 13, y: 14 }, { x: 14, y: 7 }, { x: 15, y: 5 },
        { x: 16, y: 2 }, { x: 17, y: 14 }, { x: 18, y: 7 },
        { x: 19, y: 7 }, { x: 20, y: 10 }
      ],
      primaryXAxis: {
        valueType: 'Double',
        title: 'Overs'
      },
      primaryYAxis: {
        title: 'Runs'
      },
      title: 'England - Run Rate',
      border:{
        width: 3,
        color: 'yellow'
      },
      marker: {
        visible: true
      }
    };
  },
  provide: {
    chart: [SplineAreaSeries]
  }
};
</script>

```

```
<style>
#container {
  height: 350px;
}
</style>
```

{% previewsample "page.domainurl/code-snippet/chart/series/splinearea-cs" %}

[See Also](#)

- [Data label](#)
- [Tooltip](#)

## Column Chart in Vue Chart Component

### Column

To render a column series, use series [type](#) as `Column` and inject `ColumnSeries` into the `provide`.

### APP.VUE

```
<template>
  <div id="app">
    <ejs-chart id="container" :title='title'
:primaryXAxis='primaryXAxis'>
      <e-series-collection>
        <e-series :dataSource='seriesData' type='Column'
xName='country' yName='gold' name='Gold'> </e-series>
      </e-series-collection>
    </ejs-chart>
  </div>
</template>
<script>
import Vue from "vue";
import { ChartPlugin, ColumnSeries, Category } from "@syncfusion/ej2-vue-
charts";
Vue.use(ChartPlugin);
export default {
  data: function() {
    return {
      seriesData: [
        { country: "USA", gold: 50 },
        { country: "China", gold: 40 },
        { country: "Japan", gold: 70 },
        { country: "Australia", gold: 60 },
        { country: "France", gold: 50 },
        { country: "Germany", gold: 40 },
        { country: "Italy", gold: 40 },
        { country: "Sweden", gold: 30 }
      ],
      primaryXAxis: {
        valueType: 'Category',
        title: 'Countries'
      },
      title: "Olympic Medals"
    };
  },
};
```



```

    provide: {
      chart: [ColumnSeries, Category]
    }
  };
</script>
<style>
  #container {
    height: 350px;
  }
</style>

```

{% previewsample "page.domainurl/code-snippet/chart/series/column-cs13" %}

Column space and width

The [columnSpacing](#) and [columnWidth](#) properties are used to customize the space between columns.

### APP.VUE

```

<template>
  <div id="app">
    <ejs-chart id="container" :title='title'
:primaryXAxis='primaryXAxis'>
      <e-series-collection>
        <e-series :dataSource='seriesData' type='Column'
xName='country' yName='gold' name='Gold'> </e-series>
        <e-series :dataSource='seriesData' type='Column'
xName='country' yName='silver' name='Gold' columnSpacing="0.25"
columnWidth="0.25"> </e-series>
      </e-series-collection>
    </ejs-chart>
  </div>
</template>
<script>
import Vue from "vue";
import { ChartPlugin, ColumnSeries, Category } from "@syncfusion/ej2-vue-
charts";
Vue.use(ChartPlugin);
export default {
  data: function() {
    return {
      seriesData: [
        { country: "USA", gold: 50, silver: 20 },
        { country: "China", gold: 40, silver: 30 },
        { country: "Japan", gold: 70, silver: 10 },
        { country: "Australia", gold: 60, silver: 50 },
        { country: "France", gold: 50, silver: 20 },
        { country: "Germany", gold: 40, silver: 15 },
        { country: "Italy", gold: 40, silver: 25 },
        { country: "Sweden", gold: 30, silver: 35 }
      ],
      primaryXAxis: {
        valueType: 'Category',
        title: 'Countries'
      },
      title: "Olympic Medals"
    };
  }
};

```

```

    },
    provide: {
      chart: [ColumnSeries, Category]
    }
  };
</script>
<style>
  #container {
    height: 350px;
  }
</style>

```

{% previewsample "page.domainurl/code-snippet/chart/series/column-cs23" %}

### Grouped column

You can use the [groupName](#) property to group the data points in the column type charts. Data points with same group name are grouped together.

### APP.VUE

```

<template>
  <ejs-chart style="display: block" align="center" id="container"
  :title="title" :primaryXAxis="primaryXAxis"
  :primaryYAxis="primaryYAxis" :chartArea="chartArea" :tooltip="tooltip">
    <e-series-collection>
      <e-series :dataSource="seriesData" type="Column" xName="x" yName="y"
      name="USA Total" width="2" :marker="marker"
      groupName="USA" columnWidth="0.7" columnSpacing="0.1">
      </e-series>
      <e-series :dataSource="seriesData1" type="Column" xName="x" yName="y"
      name="USA Gold" width="2" :marker="marker"
      groupName="USA" columnWidth="0.5" columnSpacing="0.1">
      </e-series>
      <e-series :dataSource="seriesData2" type="Column" xName="x" yName="y"
      name="UK Total" width="2" :marker="marker"
      groupName="UK" columnWidth="0.7" columnSpacing="0.1">
      </e-series>
      <e-series :dataSource="seriesData3" type="Column" xName="x" yName="y"
      name="UK Gold" width="2" :marker="marker"
      groupName="UK" columnWidth="0.5" columnSpacing="0.1">
      </e-series>
      <e-series :dataSource="seriesData4" type="Column" xName="x" yName="y"
      name="China Total" width="2" :marker="marker"
      groupName="China" columnWidth="0.7" columnSpacing="0.1">
      </e-series>
      <e-series :dataSource="seriesData5" type="Column" xName="x" yName="y"
      name="China Gold" width="2" :marker="marker"
      groupName="China" columnWidth="0.5" columnSpacing="0.1">
      </e-series>
    </e-series-collection>
  </ejs-chart>
</template>
<script>
import Vue from "vue";
import { Browser } from "@syncfusion/ej2-base";

```

```

import { ChartPlugin, ColumnSeries, Category, DataLabel, Tooltip, Legend }
from "@syncfusion/ej2-vue-charts";
Vue.use(ChartPlugin);
export default Vue.extend({
  data: function () {
    return {
      seriesData: [
        { x: "2012", y: 104 },
        { x: "2016", y: 121 },
        { x: "2020", y: 113 },
      ],
      seriesData1: [
        { x: "2012", y: 46 },
        { x: "2016", y: 46 },
        { x: "2020", y: 39 },
      ],
      seriesData2: [
        { x: "2012", y: 65 },
        { x: "2016", y: 67 },
        { x: "2020", y: 65 },
      ],
      seriesData3: [
        { x: "2012", y: 29 },
        { x: "2016", y: 27 },
        { x: "2020", y: 22 },
      ],
      seriesData4: [
        { x: "2012", y: 91 },
        { x: "2016", y: 70 },
        { x: "2020", y: 88 },
      ],
      seriesData5: [
        { x: "2012", y: 38 },
        { x: "2016", y: 26 },
        { x: "2020", y: 38 },
      ],
      //Initializing Primary X Axis
      primaryXAxis: {
        valueType: "Category",
        interval: 1,
        majorGridLines: { width: 0 },
      },
      chartArea: { border: { width: 0 } },
      //Initializing Primary Y Axis
      primaryYAxis: {
        majorGridLines: { width: 0 },
        majorTickLines: { width: 0 },
        lineStyle: { width: 0 },
        labelStyle: { color: "transparent" },
      },
      tooltip: {
        enable: true,
      },
      title: "Olympics Medal Tally",
    };
  },
  provide: {

```

```

        chart: [ColumnSeries, Legend, DataLabel, Category, Tooltip],
    },
});
</script>

```

{% previewsample "page.domainurl/code-snippet/chart/series/group-column-cs1" %}

### Cylindrical column chart

To render a cylindrical column chart, set the [columnFacet](#) property to **Cylinder** in the chart series.

### APP.VUE

```

<template>
  <div id="app">
    <ejs-chart id="container" :title='title'
:primaryXAxis='primaryXAxis' :primaryYAxis='primaryYAxis'
:tooltip='tooltip'>
      <e-series-collection>
        <e-series :dataSource='seriesData' type='Column'
columnFacet='Cylinder' xName='country' yName='gold'
tooltipMappingName='tooltipMappingName'> </e-series>
      </e-series-collection>
    </ejs-chart>
  </div>
</template>
<script>
import Vue from "vue";
import { ChartPlugin, ColumnSeries, Category } from "@syncfusion/ej2-vue-
charts";
Vue.use(ChartPlugin);
export default {
  data: function() {
    return {
      seriesData: [
        { country: "USA", gold: 50, tooltipMappingName: 'USA' },
        { country: "Japan", gold: 70, tooltipMappingName: 'Japan' },
        { country: "Australia", gold: 60, tooltipMappingName: 'Australia' },
        { country: "France", gold: 50, tooltipMappingName: 'France' },
        { country: "Italy", gold: 40, tooltipMappingName: 'Italy' },
        { country: "Sweden", gold: 55, tooltipMappingName: 'Sweden' }
      ],
      primaryXAxis: {
        valueType: 'Category',
        interval: 1
      },
      primaryYAxis: {
        {
          minimum: 0,
          maximum: 80,
          interval: 10,
          title: 'Medal Count'
        },
        tooltip: { enable: true, header: "<b>${point.tooltip}</b>", format:
"Gold Medal: <b>${point.y}</b>" },
        title: "Olympic Gold Medal Counts - RIO"
      }
    }
  }
}

```

```

    };
  },
  provide: {
    chart: [ColumnSeries, Category]
  }
};
</script>
<style>
  #container {
    height: 350px;
  }
</style>

```

{% previewsample "page.domainurl/code-snippet/chart/series/column-cs24" %}

#### *Series customization*

The following properties can be used to customize the **column** series.

- [fill](#) – Specifies the color of the series.
- [opacity](#) – Specifies the opacity of [fill](#).
- [dashArray](#) – Specifies the dashes of series.
- [border](#) – Specifies the [color](#) and [width](#) of series border.

#### **APP.VUE**

```

<template>
  <div id="app">
    <ejs-chart id="container" :title='title'
:primaryXAxis='primaryXAxis'>
      <e-series-collection>
        <e-series :dataSource='seriesData' columnWidth=0.5
columnSpacing=0.5 type='Column' xName='country' yName='gold' name='Gold'
fill='red' dashArray="4" opacity="0.4" :border='border'> </e-series>
      </e-series-collection>
    </ejs-chart>
  </div>
</template>
<script>
import Vue from "vue";
import { ChartPlugin, ColumnSeries, Category } from "@syncfusion/ej2-vue-
charts";
Vue.use(ChartPlugin);
export default {
  data() {
    return {
      seriesData: [
        { country: "USA", gold: 50 },
        { country: "China", gold: 40 },
        { country: "Japan", gold: 70 },
        { country: "Australia", gold: 60 },
        { country: "France", gold: 50 },
        { country: "Germany", gold: 40 },
        { country: "Italy", gold: 40 },
        { country: "Sweden", gold: 30 }
      ],
    };
  }
};

```

```

        border: {
            color: 'black',
            width: 2
        },
        primaryXAxis: {
            valueType: 'Category',
            title: 'Countries'
        },
        title: "Olympic Medals"
    };
},
provide: {
    chart: [ColumnSeries, Category]
},
};
</script>
<style>
    #container {
        height: 350px;
    }
</style>

```

{% previewsample "page.domainurl/code-snippet/chart/series/column-cs18" %}

[See also](#)

- [Data label](#)
- [Tooltip](#)

## Range Column in Vue Chart Component

### [Range Column](#)

To render a range column series, use series [type](#) as `RangeColumn` and inject `RangeColumnSeries` into the `provide`.

### **APP.VUE**

```

<template>
    <div id="app">
        <ejs-chart id="container" :title='title'
        :primaryXAxis='primaryXAxis'>
            <e-series-collection>
                <e-series :dataSource='seriesData1' type='RangeColumn'
                xName='x' low='low' high='high'></e-series>
                <e-series :dataSource='seriesData2' type='RangeColumn'
                xName='x' low='low' high='high'></e-series>
            </e-series-collection>
        </ejs-chart>
    </div>
</template>
<script>
import Vue from "vue";
import { ChartPlugin, RangeColumnSeries, Category } from "@syncfusion/ej2-
vue-charts";
Vue.use(ChartPlugin);
export default {

```

```

data() {
  return {
    seriesData1: [
      { x: 'Jan', low: 0.7, high: 6.1 }, { x: 'Feb', low: 1.3, high: 6.3 }, { x: 'Mar', low: 1.9, high: 8.5 },
      { x: 'Apr', low: 3.1, high: 10.8 }, { x: 'May', low: 5.7, high: 14.40 }, { x: 'Jun', low: 8.4, high: 16.90 },
      { x: 'Jul', low: 10.6, high: 19.20 }, { x: 'Aug', low: 10.5, high: 18.9 }, { x: 'Sep', low: 8.5, high: 16.1 },
      { x: 'Oct', low: 6.0, high: 12.5 }, { x: 'Nov', low: 1.5, high: 6.9 }, { x: 'Dec', low: 5.1, high: 12.1 }
    ],
    seriesData2: [
      { x: 'Jan', low: 1.7, high: 7.1 }, { x: 'Feb', low: 1.9, high: 7.7 }, { x: 'Mar', low: 1.2, high: 7.5 },
      { x: 'Apr', low: 2.5, high: 9.8 }, { x: 'May', low: 4.7, high: 11.4 }, { x: 'Jun', low: 6.4, high: 14.4 },
      { x: 'Jul', low: 9.6, high: 17.2 }, { x: 'Aug', low: 10.7, high: 17.9 }, { x: 'Sep', low: 7.5, high: 15.1 },
      { x: 'Oct', low: 3.0, high: 10.5 }, { x: 'Nov', low: 1.2, high: 7.9 }, { x: 'Dec', low: 4.1, high: 9.1 }
    ],
    primaryXAxis: {
      title: 'month',
      valueType: 'Category'
    },
    title: "Maximum and minimum Temperature"
  };
},
provide: {
  chart: [RangeColumnSeries, Category]
},
};
</script>
<style>
  #container {
    height: 350px;
  }
</style>

```

{% previewsample "page.domainurl/code-snippet/chart/series/column-cs14" %}

### Series customization

The following properties can be used to customize the range column series.

- [fill](#) – Specifies the color of the series.
- [opacity](#) – Specifies the opacity of [fill](#).
- [dashArray](#) – Specifies the dashes of series.
- [border](#) – Specifies the [color](#) and [width](#) of series border.

### APP.VUE

```

<template>
  <div id="app">

```

```

    <ejs-chart id="container" :title='title'
:primaryXAxis='primaryXAxis'>
      <e-series-collection>
        <e-series :dataSource='seriesData1' type='RangeColumn'
xName='x' low='low' high='high' fill="blue" :border="border" dashArray='2.5'
opacity='0.8'></e-series>
        <e-series :dataSource='seriesData2' type='RangeColumn'
xName='x' low='low' high='high' fill="red" :border="border" dashArray='2.5'
opacity='0.8'></e-series>
      </e-series-collection>
    </ejs-chart>
  </div>
</template>
<script>
import Vue from "vue";
import { ChartPlugin, RangeColumnSeries, Category } from "@syncfusion/ej2-
vue-charts";
Vue.use(ChartPlugin);
export default {
  data() {
    return {
      seriesData1: [
        { x: 'Jan', low: 0.7, high: 6.1 }, { x: 'Feb', low: 1.3, high:
6.3 }, { x: 'Mar', low: 1.9, high: 8.5 },
        { x: 'Apr', low: 3.1, high: 10.8 }, { x: 'May', low: 5.7, high:
14.40 }, { x: 'Jun', low: 8.4, high: 16.90 },
        { x: 'Jul', low: 10.6, high: 19.20 }, { x: 'Aug', low:
10.5, high: 18.9 }, { x: 'Sep', low: 8.5, high: 16.1 },
        { x: 'Oct', low: 6.0, high: 12.5 }, { x: 'Nov', low: 1.5, high:
6.9 }, { x: 'Dec', low: 5.1, high: 12.1 }
      ],
      seriesData2: [
        { x: 'Jan', low: 1.7, high: 7.1 }, { x: 'Feb', low: 1.9, high:
7.7 }, { x: 'Mar', low: 1.2, high: 7.5 },
        { x: 'Apr', low: 2.5, high: 9.8 }, { x: 'May', low: 4.7, high:
11.4 }, { x: 'Jun', low: 6.4, high: 14.4 },
        { x: 'Jul', low: 9.6, high: 17.2 }, { x: 'Aug', low: 10.7,
high: 17.9 }, { x: 'Sep', low: 7.5, high: 15.1 },
        { x: 'Oct', low: 3.0, high: 10.5 }, { x: 'Nov', low: 1.2, high:
7.9 }, { x: 'Dec', low: 4.1, high: 9.1 }
      ],
      primaryXAxis: {
        title: 'month',
        valueType: 'Category'
      },
      title: "Maximum and minimum Temperature",
      border:{
        width: 3,
        color: 'black'
      }
    };
  },
  provide: {
    chart: [RangeColumnSeries, Category]
  },
};
</script>

```



```
<style>
  #container {
    height: 350px;
  }
</style>
```

{% previewsample "page.domainurl/code-snippet/chart/series/rangecolumn-cs" %}

[See Also](#)

- [Data label](#)
- [Tooltip](#)

## Stacked Column in Vue Chart Component

### *Stacked column*

To render a stacked column series, use series [type](#) as `StackingColumn` and inject `StackingColumnSeries` into the `provide`.

### **APP.VUE**

```
<template>
  <div id="app">
    <ejs-chart id="container" :title='title'
:primaryXAxis='primaryXAxis'>
      <e-series-collection>
        <e-series :dataSource='seriesData' type='StackingColumn'
xName='x' yName='y' name='USA'> </e-series>
        <e-series :dataSource='seriesData' type='StackingColumn'
xName='x' yName='y1' name='UK'> </e-series>
        <e-series :dataSource='seriesData' type='StackingColumn'
xName='x' yName='y2' name='Canada'> </e-series>
        <e-series :dataSource='seriesData' type='StackingColumn'
xName='x' yName='y3' name='China'> </e-series>
      </e-series-collection>
    </ejs-chart>
  </div>
</template>
<script>
import Vue from "vue";
import { ChartPlugin, StackingColumnSeries, Category } from
"@syncfusion/ej2-vue-charts";
Vue.use(ChartPlugin);
export default {
  data() {
    return {
      seriesData:[
        { x: '2014', y: 111.1, y1: 76.9, y2: 66.1, y3: 34.1 },
        { x: '2015', y: 127.3, y1: 99.5, y2: 79.3, y3: 38.2 },
        { x: '2016', y: 143.4, y1: 121.7, y2: 91.3, y3: 44.0 },
        { x: '2017', y: 159.9, y1: 142.5, y2: 102.4, y3: 51.6 },
        { x: '2018', y: 175.4, y1: 166.7, y2: 112.9, y3: 61.9 },
        { x: '2019', y: 189.0, y1: 182.9, y2: 122.4, y3: 71.5 },
        { x: '2020', y: 202.7, y1: 197.3, y2: 120.9, y3: 82.0 }
      ],
      primaryXAxis: {
```

```

        title: 'Years',
        interval: 1,
        valueType: 'Category'
    },
    title: "Mobile Game Market by Country"
};
},
provide: {
    chart: [StackingColumnSeries, Category]
},
};
</script>
<style>
    #container {
        height: 350px;
    }
</style>

```

{% previewsample "page.domainurl/code-snippet/chart/series/column-cs15" %}

### *Cylindrical stacked column chart*

To render a cylindrical stacked column chart, set the [columnFacet](#) property to **Cylinder** in the chart series.

### **APP.VUE**

```

<template>
    <div id="app">
        <ejs-chart id="container" :primaryXAxis='primaryXAxis'
:primaryYAxis='primaryYAxis'>
            <e-series-collection>
                <e-series :dataSource='seriesData' type='StackingColumn'
columnFacet='Cylinder' xName='x' yName='y' name='UK'> </e-series>
                <e-series :dataSource='seriesData' type='StackingColumn'
columnFacet='Cylinder' xName='x' yName='y1' name='Germany'> </e-series>
                <e-series :dataSource='seriesData' type='StackingColumn'
columnFacet='Cylinder' xName='x' yName='y2' name='France'> </e-series>
                <e-series :dataSource='seriesData' type='StackingColumn'
columnFacet='Cylinder' xName='x' yName='y3' name='Italy'> </e-series>
            </e-series-collection>
        </ejs-chart>
    </div>
</template>
<script>
import Vue from "vue";
import { ChartPlugin, StackingColumnSeries, Category } from
"@syncfusion/ej2-vue-charts";
Vue.use(ChartPlugin);
export default {
    data() {
        return {
            seriesData:[
                { x: '2014', y: 111.1, y1: 76.9, y2: 66.1, y3: 34.1 },
                { x: '2015', y: 127.3, y1: 99.5, y2: 79.3, y3: 38.2 },
                { x: '2016', y: 143.4, y1: 121.7, y2: 91.3, y3: 44.0 },
                { x: '2017', y: 159.9, y1: 142.5, y2: 102.4, y3: 51.6 },
            ]
        }
    }
}

```

```

        { x: '2018', y: 175.4, y1: 166.7, y2: 112.9, y3: 61.9 },
        { x: '2019', y: 189.0, y1: 182.9, y2: 122.4, y3: 71.5 },
        { x: '2020', y: 202.7, y1: 197.3, y2: 120.9, y3: 82.0 }
    ],
    primaryXAxis: {
        title: 'Years',
        interval: 1,
        valueType: 'Category'
    },
    primaryYAxis: {
        title: 'Sales in Billions',
        minimum: 0,
        maximum: 700,
        interval: 100,
        labelFormat: '{value}B'
    },
    };
},
provide: {
    chart: [StackingColumnSeries, Category]
},
};
</script>
<style>
    #container {
        height: 350px;
    }
</style>

```

{% previewsample "page.domainurl/code-snippet/chart/series/column-cs25" %}

#### Series customization

The following properties can be used to customize the **stacked column** series.

- [fill](#) – Specifies the color of the series.
- [opacity](#) – Specifies the opacity of [fill](#).
- [dashArray](#) – Specifies the dashes of series.
- [border](#) – Specifies the [color](#) and [width](#) of series border.

#### APP.VUE

```

<template>
    <div id="app">
        <ejs-chart id="container" :title='title'
        :primaryXAxis='primaryXAxis'>
            <e-series-collection>
                <e-series :dataSource='seriesData' type='StackingColumn'
                xName='x' yName='y' name='USA' fill="red" dashArray="5" :border="border">
            </e-series>
                <e-series :dataSource='seriesData' type='StackingColumn'
                xName='x' yName='y1' name='UK' fill="yellow" dashArray="5" :border="border">
            </e-series>
                <e-series :dataSource='seriesData' type='StackingColumn'
                xName='x' yName='y2' name='Canada' fill="green" dashArray="5"
                :border="border"> </e-series>
            </e-series-collection>
        </ejs-chart>
    </div>
</template>

```

```

        <e-series :dataSource='seriesData' type='StackingColumn'
xName='x' yName='y3' name='China' fill="blue" dashArray="5"
:border="border"> </e-series>
    </e-series-collection>
</ejs-chart>
</div>
</template>
<script>
import Vue from "vue";
import { ChartPlugin, StackingColumnSeries, Category } from
"@syncfusion/ej2-vue-charts";
Vue.use(ChartPlugin);
export default {
  data() {
    return {
      seriesData:[
        { x: '2014', y: 111.1, y1: 76.9, y2: 66.1, y3: 34.1 },
        { x: '2015', y: 127.3, y1: 99.5, y2: 79.3, y3: 38.2 },
        { x: '2016', y: 143.4, y1: 121.7, y2: 91.3, y3: 44.0 },
        { x: '2017', y: 159.9, y1: 142.5, y2: 102.4, y3: 51.6 },
        { x: '2018', y: 175.4, y1: 166.7, y2: 112.9, y3: 61.9 },
        { x: '2019', y: 189.0, y1: 182.9, y2: 122.4, y3: 71.5 },
        { x: '2020', y: 202.7, y1: 197.3, y2: 120.9, y3: 82.0 }
      ],
      primaryXAxis: {
        title: 'Years',
        interval: 1,
        valueType: 'Category'
      },
      title: "Mobile Game Market by Country",
      border:{
        width: 1,
        color: 'black'
      }
    };
  },
  provide: {
    chart: [StackingColumnSeries, Category]
  },
};
</script>
<style>
#container {
  height: 350px;
}
</style>

```

{% previewsample "page.domainurl/code-snippet/chart/series/stackcolumn-cs" %}

[See also](#)

- [Data label](#)
- [Tooltip](#)

## 100% Stacked Column in Vue Chart Component

*100% Stacked column*

To render a 100% stacked column series, use series [type](#) as `StackingColumn100` and inject `StackingColumnSeries` into the `provide`.

**APP.VUE**

```

<template>
  <div id="app">
    <ejs-chart id="container" :title='title'
    :primaryXAxis='primaryXAxis'>
      <e-series-collection>
        <e-series :dataSource='seriesData' type='StackingColumn100'
        xName='x' yName='y' name='USA'> </e-series>
        <e-series :dataSource='seriesData' type='StackingColumn100'
        xName='x' yName='y1' name='UK'> </e-series>
        <e-series :dataSource='seriesData' type='StackingColumn100'
        xName='x' yName='y2' name='Canada'> </e-series>
        <e-series :dataSource='seriesData' type='StackingColumn100'
        xName='x' yName='y3' name='China'> </e-series>
      </e-series-collection>
    </ejs-chart>
  </div>
</template>
<script>
import Vue from "vue";
import { ChartPlugin, StackingColumnSeries, Category } from
"@syncfusion/ej2-vue-charts";
Vue.use(ChartPlugin);
export default {
  data() {
    return {
      seriesData:[
        { x: '2006', y: 900, y1: 190, y2: 250, y3: 150 },
        { x: '2007', y: 544, y1: 226, y2: 145, y3: 120 },
        { x: '2008', y: 880, y1: 194, y2: 190, y3: 115 },
        { x: '2009', y: 675, y1: 250, y2: 220, y3: 125 },
        { x: '2010', y: 765, y1: 222, y2: 225, y3: 132 },
        { x: '2011', y: 679, y1: 181, y2: 135, y3: 137 },
        { x: '2012', y: 770, y1: 128, y2: 152, y3: 110 },
      ],
      primaryXAxis: {
        title: 'Years',
        interval: 1,
        valueType: 'Category'
      },
      title: "Gross Domestic Product Growth"
    };
  },
  provide: {
    chart: [StackingColumnSeries, Category]
  },
};
</script>
<style>
#container {

```

```

    height: 350px;
  }
</style>

```

{% previewsample "page.domainurl/code-snippet/chart/series/column-cs16" %}

### 100% Cylindrical stacked column chart

To render a 100% cylindrical stacked column chart, set the [columnFacet](#) property to **Cylinder** in the chart series.

### APP.VUE

```

<template>
  <div id="app">
    <ejs-chart id="container" :title='title'
:primaryXAxis='primaryXAxis' :primaryYAxis='primaryYAxis'>
      <e-series-collection>
        <e-series :dataSource='seriesData' type='StackingColumn100'
columnFacet='Cylinder' xName='x' yName='y' name='UK'> </e-series>
        <e-series :dataSource='seriesData' type='StackingColumn100'
columnFacet='Cylinder' xName='x' yName='y1' name='Germany'> </e-series>
        <e-series :dataSource='seriesData' type='StackingColumn100'
columnFacet='Cylinder' xName='x' yName='y2' name='France'> </e-series>
        <e-series :dataSource='seriesData' type='StackingColumn100'
columnFacet='Cylinder' xName='x' yName='y3' name='Italy'> </e-series>
      </e-series-collection>
    </ejs-chart>
  </div>
</template>
<script>
import Vue from "vue";
import { ChartPlugin, StackingColumnSeries, DateTime } from
"@syncfusion/ej2-vue-charts";
Vue.use(ChartPlugin);
export default {
  data() {
    return {
      seriesData:[
        { x: new Date(2006, 0, 1), y: 900, y1: 190, y2: 250, y3: 150 },
        { x: new Date(2007, 0, 1), y: 544, y1: 226, y2: 145, y3: 120 },
        { x: new Date(2008, 0, 1), y: 880, y1: 194, y2: 190, y3: 115 },
        { x: new Date(2009, 0, 1), y: 675, y1: 250, y2: 220, y3: 125 },
        { x: new Date(2010, 0, 1), y: 765, y1: 222, y2: 225, y3: 132 },
        { x: new Date(2011, 0, 1), y: 679, y1: 181, y2: 135, y3: 137 },
        { x: new Date(2012, 0, 1), y: 770, y1: 128, y2: 152, y3: 110 }
      ],
      primaryXAxis: {
        title: 'Years',
        interval: 1,
        valueType: 'DateTime',
        labelPlacement: 'BetweenTicks',
        labelFormat: 'y'
      },
      primaryYAxis: {
        title: 'GDP (%) Per Annum',
        rangePadding: 'None',

```

```

        labelFormat: '{value}%'
      },
      title: "Gross Domestic Product Growth"
    };
  },
  provide: {
    chart: [StackingColumnSeries, DateTime]
  },
};
</script>
<style>
  #container {
    height: 350px;
  }
</style>

```

{% previewsample "page.domainurl/code-snippet/chart/series/column-cs26" %}

### Series customization

The following properties can be used to customize the 100% stacked column series.

- [fill](#) – Specifies the color of the series.
- [opacity](#) – Specifies the opacity of [fill](#).
- [dashArray](#) – Specifies the dashes of series.
- [border](#) – Specifies the [color](#) and [width](#) of series border.

### APP.VUE

```

<template>
  <div id="app">
    <ejs-chart id="container" :title='title'
:primaryXAxis='primaryXAxis'>
      <e-series-collection>
        <e-series :dataSource='seriesData' type='StackingColumn100'
xName='x' yName='y' name='USA' fill="red" dashArray="5,5" :border="border">
        </e-series>
        <e-series :dataSource='seriesData' type='StackingColumn100'
xName='x' yName='y1' name='UK' fill="yellow" dashArray="5,5"
:border="border"> </e-series>
        <e-series :dataSource='seriesData' type='StackingColumn100'
xName='x' yName='y2' name='Canada' fill="green" dashArray="5,5"
:border="border"> </e-series>
        <e-series :dataSource='seriesData' type='StackingColumn100'
xName='x' yName='y3' name='China' fill="blue" dashArray="5,5"
:border="border"> </e-series>
      </e-series-collection>
    </ejs-chart>
  </div>
</template>
<script>
import Vue from "vue";
import { ChartPlugin, StackingColumnSeries, Category } from
"@syncfusion/ej2-vue-charts";
Vue.use(ChartPlugin);
export default {

```

```

data() {
  return {
    seriesData:[
      { x: '2006', y: 900, y1: 190, y2: 250, y3: 150 },
      { x: '2007', y: 544, y1: 226, y2: 145, y3: 120 },
      { x: '2008', y: 880, y1: 194, y2: 190, y3: 115 },
      { x: '2009', y: 675, y1: 250, y2: 220, y3: 125 },
      { x: '2010', y: 765, y1: 222, y2: 225, y3: 132 },
      { x: '2011', y: 679, y1: 181, y2: 135, y3: 137 },
      { x: '2012', y: 770, y1: 128, y2: 152, y3: 110 },
    ],
    primaryXAxis: {
      title: 'Years',
      interval: 1,
      valueType: 'Category'
    },
    title: "Gross Domestic Product Growth",
    border:{
      width: 1.5,
      color: 'black'
    }
  };
},
provide: {
  chart: [StackingColumnSeries, Category]
},
};
</script>
<style>
#container {
  height: 350px;
}
</style>

```

{% previewsample "page.domainurl/code-snippet/chart/series/stackedcolumn-cs" %}

[See also](#)

- [Data label](#)
- [Tooltip](#)

## Bar Chart in Vue Chart Component

*Bar*

To render a bar series, use series [type](#) as `Bar` and inject `BarSeries` into the `provide`.

### APP.VUE

```

<template>
  <div id="app">
    <ejs-chart id="container" :title='title'>
      <e-series-collection>
        <e-series :dataSource='seriesData' type='Bar' xName='x'
yName='y' name='India'> </e-series>
      </e-series-collection>
    </ejs-chart>
  </div>

```



```

    </div>
</template>
<script>
import Vue from "vue";
import { ChartPlugin, BarSeries } from "@syncfusion/ej2-vue-charts";
Vue.use(ChartPlugin);
export default {
  data() {
    return {
      seriesData: [
        { x: 2006, y: 7.8 }, { x: 2007, y: 7.2},
        { x: 2008, y: 6.8 }, { x: 2009, y: 10.7 },
        { x: 2010, y: 10.8}, { x: 2011, y: 9.8 }
      ],
      title: "Unemployment rate (%)"
    };
  },
  provide: {
    chart: [BarSeries]
  }
};
</script>
<style>
#container {
  height: 350px;
}
</style>

```

{% previewsample "page.domainurl/code-snippet/chart/series/bar-cs1" %}

#### Bar space and width

The [columnSpacing](#) and [columnWidth](#) properties are used to customize the space between bars.

#### APP.VUE

```

<template>
  <div id="app">
    <ejs-chart id="container" :title='title'>
      <e-series-collection>
        <e-series :dataSource='seriesData' type='Bar' xName='x'
yName='y' name='India'> </e-series>
        <e-series :dataSource='seriesData' type='Bar' xName='x'
yName='y1' name='India' columnSpacing="1.5" columnWidth="1.25"> </e-series>
      </e-series-collection>
    </ejs-chart>
  </div>
</template>
<script>
import Vue from "vue";
import { ChartPlugin, BarSeries } from "@syncfusion/ej2-vue-charts";
Vue.use(ChartPlugin);
export default {
  data() {
    return {
      seriesData: [
        { x: 2006, y: 7.8, y1: 7 }, { x: 2007, y: 7.2, y1: 6.8},

```

```

        { x: 2008, y: 6.8, y1: 6 }, { x: 2009, y: 10.7, y1: 4.5 },
        { x: 2010, y: 10.8, y1: 3}, { x: 2011, y: 9.8, y1: 4 }
    ],
    title: "Unemployment rate (%)"
};
},
provide: {
    chart: [BarSeries]
}
};
</script>
<style>
    #container {
        height: 350px;
    }
</style>

```

{% previewsample "page.domainurl/code-snippet/chart/series/bar-cs" %}

### Grouped bar

You can use the [groupName](#) property to group the data points in the bar type charts. Data points with same group name are grouped together.

### APP.VUE

```

<template>
  <ejs-chart
    style="display: block"
    align="center"
    id="container"
    :title="title"
    :primaryXAxis="primaryXAxis"
    :primaryYAxis="primaryYAxis"
    :chartArea="chartArea"
    :tooltip="tooltip"
  >
    <e-series-collection>
      <e-series
        :dataSource="seriesData"
        type="Bar"
        xName="x"
        yName="y"
        name="USA Total"
        width="2"
        :marker="marker"
        groupName="USA"
        columnWidth="0.7"
        columnSpacing="0.1"
      >
      </e-series>
      <e-series
        :dataSource="seriesData1"
        type="Bar"
        xName="x"
        yName="y"
        name="USA Gold"
      >
      </e-series>
    </e-series-collection>
  </ejs-chart>

```

```
        width="2"
        :marker="marker"
        groupName="USA"
        columnWidth="0.5"
        columnSpacing="0.1"
    >
</e-series>
<e-series
    :dataSource="seriesData2"
    type="Bar"
    xName="x"
    yName="y"
    name="UK Total"
    width="2"
    :marker="marker"
    groupName="UK"
    columnWidth="0.7"
    columnSpacing="0.1"
>
</e-series>
<e-series
    :dataSource="seriesData3"
    type="Bar"
    xName="x"
    yName="y"
    name="UK Gold"
    width="2"
    :marker="marker"
    groupName="UK"
    columnWidth="0.5"
    columnSpacing="0.1"
>
</e-series>
<e-series
    :dataSource="seriesData4"
    type="Bar"
    xName="x"
    yName="y"
    name="China Total"
    width="2"
    :marker="marker"
    groupName="China"
    columnWidth="0.7"
    columnSpacing="0.1"
>
</e-series>
<e-series
    :dataSource="seriesData5"
    type="Bar"
    xName="x"
    yName="y"
    name="China Gold"
    width="2"
    :marker="marker"
    groupName="China"
    columnWidth="0.5"
    columnSpacing="0.1"
```

```

    >
    </e-series>
  </e-series-collection>
</ejs-chart>
</template>
<script>
import Vue from "vue";
import { Browser } from "@syncfusion/ej2-base";
import {
  ChartPlugin,
  BarSeries,
  Category,
  DataLabel,
  Tooltip,
  Legend,
} from "@syncfusion/ej2-vue-charts";
Vue.use(ChartPlugin);
export default Vue.extend({
  data: function () {
    return {
      seriesData: [
        { x: "2012", y: 104 },
        { x: "2016", y: 121 },
        { x: "2020", y: 113 },
      ],
      seriesData1: [
        { x: "2012", y: 46 },
        { x: "2016", y: 46 },
        { x: "2020", y: 39 },
      ],
      seriesData2: [
        { x: "2012", y: 65 },
        { x: "2016", y: 67 },
        { x: "2020", y: 65 },
      ],
      seriesData3: [
        { x: "2012", y: 29 },
        { x: "2016", y: 27 },
        { x: "2020", y: 22 },
      ],
      seriesData4: [
        { x: "2012", y: 91 },
        { x: "2016", y: 70 },
        { x: "2020", y: 88 },
      ],
      seriesData5: [
        { x: "2012", y: 38 },
        { x: "2016", y: 26 },
        { x: "2020", y: 38 },
      ],
      //Initializing Primary X Axis
      primaryXAxis: {
        valueType: "Category",
        interval: 1,
        majorGridLines: { width: 0 },
      },
      chartArea: { border: { width: 0 } },
    };
  },
});

```

```
//Initializing Primary Y Axis
primaryYAxis: {
  majorGridLines: { width: 0 },
  majorTickLines: { width: 0 },
  lineStyle: { width: 0 },
  labelStyle: { color: "transparent" },
},
marker: {
  dataLabel: {
    visible: true,
    position: "Top",
    font: { fontWeight: "600", color: "#ffffff" },
  },
},
tooltip: {
  enable: true,
},
title: "Olympics Medal Tally",
};
},
provide: {
  chart: [BarSeries, Legend, DataLabel, Category, Tooltip],
},
});
</script>
```

{% previewsample "page.domainurl/code-snippet/chart/series/group-bar-cs1" %}

#### Cylindrical bar chart

To render a cylindrical bar chart, set the [columnFacet](#) property to **Cylinder** in the chart series.

#### APP.VUE

```
<template>
  <div id="app">
    <ejs-chart id="container" :title='title'
:primaryXAxis='primaryXAxis' :primaryYAxis='primaryYAxis'>
      <e-series-collection>
        <e-series :dataSource='seriesData' type='Bar'
columnFacet='Cylinder' xName='x' yName='y' name='India'> </e-series>
      </e-series-collection>
    </ejs-chart>
  </div>
</template>
<script>
import Vue from "vue";
import { ChartPlugin, BarSeries } from "@syncfusion/ej2-vue-charts";
Vue.use(ChartPlugin);
export default {
  data() {
    return {
      seriesData: [
        { x: 2006, y: 9 },
        { x: 2007, y: 7.8 },
        { x: 2008, y: 10.5 },
        { x: 2009, y: 8.4 },
      ],
    };
  },
};
```

```

        { x: 2010, y: 6 },
        { x: 2011, y: 11 }
    ],
    title: "Unemployment rate in percentage",
    primaryXAxis: {
        minimum: 2005,
        maximum: 2012,
        interval: 1,
        title: 'Year'
    },
    primaryYAxis:
    {
        minimum: 3,
        maximum: 12,
        interval: 1,
        title: 'Percentage',
        labelFormat: '{value}%'
    }
    };
},
provide: {
    chart: [BarSeries]
}
};
</script>
<style>
    #container {
        height: 350px;
    }
</style>

```

{% previewsample "page.domainurl/code-snippet/chart/series/bar-cs6" %}

### [Series customization](#)

The following properties can be used to customize the **bar** series.

- [fill](#) – Specifies the color of the series.
- [opacity](#) – Specifies the opacity of [fill](#).
- [dashArray](#) – Specifies the dashes of series.
- [border](#) – Specifies the [color](#) and [width](#) of series border.

### **APP.VUE**

```

<template>
    <div id="app">
        <ejs-chart id="container" :title='title'>
            <e-series-collection>
                <e-series :dataSource='seriesData' columnSpacing=0.25
columnWidth=0.75 type='Bar' xName='x' yName='y' name='India' fill='green'
opacity='0.8' dashArray='4' :border='border'> </e-series>
            </e-series-collection>
        </ejs-chart>
    </div>
</template>
<script>

```

```
import Vue from "vue";
import { ChartPlugin, BarSeries } from "@syncfusion/ej2-vue-charts";
Vue.use(ChartPlugin);
export default {
  data() {
    return {
      seriesData: [
        { x: 2006, y: 7.8 }, { x: 2007, y: 7.2 },
        { x: 2008, y: 6.8 }, { x: 2009, y: 10.7 },
        { x: 2010, y: 10.8 }, { x: 2011, y: 9.8 }
      ],
      border: {
        width: 2,
        color: 'red'
      },
      title: "Unemployment rate (%)"
    };
  },
  provide: {
    chart: [BarSeries]
  }
};
</script>
<style>
  #container {
    height: 350px;
  }
</style>
```

{% previewsample "page.domainurl/code-snippet/chart/series/bar-cs5" %}

[See also](#)

- [Data label](#)
- [Tooltip](#)

## Stacked Bar in Vue Chart Component

[Stacked bar](#)

To render a stacked bar series, use series [type](#) as `StackingBar` and inject `StackingBarSeries` into the `provide`.

### APP.VUE

```
<template>
  <div id="app">
    <ejs-chart id="container" :title='title'
:primaryXAxis='primaryXAxis'>
      <e-series-collection>
        <e-series :dataSource='seriesData' type='StackingBar'
xName='x' yName='y' name='Apple'> </e-series>
        <e-series :dataSource='seriesData' type='StackingBar'
xName='x' yName='y1' name='Orange'> </e-series>
        <e-series :dataSource='seriesData' type='StackingBar'
xName='x' yName='y2' name='Wastage'> </e-series>
      </e-series-collection>
    </div>
  </template>
```

```

        </ejs-chart>
    </div>
</template>
<script>
import Vue from "vue";
import { ChartPlugin, StackingBarSeries, Category } from "@syncfusion/ej2-vue-charts";
Vue.use(ChartPlugin);
export default {
    data: function() {
        return {
            seriesData:[
                { x: 'Jan', y: 6, y1: 6, y2: -1 }, { x: 'Feb', y: 8, y1: 8, y2: -1.5 },
                { x: 'Mar', y: 12, y1: 11, y2: -2 }, { x: 'Apr', y: 15, y1: 16, y2: -2.5 },
                { x: 'May', y: 20, y1: 21, y2: -3 }, { x: 'Jun', y: 24, y1: 25, y2: -3.5 },
                { x: 'Jul', y: 28, y1: 27, y2: -4 }, { x: 'Aug', y: 32, y1: 31, y2: -4.5 },
                { x: 'Sep', y: 33, y1: 34, y2: -5 }, { x: 'Oct', y: 35, y1: 34, y2: -5.5 },
                { x: 'Nov', y: 40, y1: 41, y2: -6 }, { x: 'Dec', y: 42, y1: 42, y2: -6.5 }
            ],
            primaryXAxis: {
                valueType: 'Category',
                title: 'Months'
            },
            title: "Sales Comparison"
        };
    },
    provide: {
        chart: [StackingBarSeries, Category]
    },
};
</script>
<style>
    #container {
        height: 350px;
    }
</style>

```

{% previewsample "page.domainurl/code-snippet/chart/series/bar-cs2" %}

### *Cylindrical stacked bar chart*

To render a cylindrical stacked bar chart, set the [columnFacet](#) property to **Cylinder** in the chart series.

### **APP.VUE**

```

<template>
    <div id="app">
        <ejs-chart id="container">
            <e-series-collection>
                <e-series :dataSource='seriesData' type='StackingBar'
                columnFacet='Cylinder' xName='x' yName='y'> </e-series>
            </e-series-collection>
        </ejs-chart>
    </div>
</template>

```



```

        <e-series :dataSource='seriesData' type='StackingBar'
columnFacet='Cylinder' xName='x' yName='y1'> </e-series>
        <e-series :dataSource='seriesData' type='StackingBar'
columnFacet='Cylinder' xName='x' yName='y2'> </e-series>
    </e-series-collection>
</ejs-chart>
</div>
</template>
<script>
import Vue from "vue";
import { ChartPlugin, StackingBarSeries, Category } from "@syncfusion/ej2-
vue-charts";
Vue.use(ChartPlugin);
export default {
    data: function() {
        return {
            seriesData:[
                { x: 2000, y: 0.61, y1: 0.03, y2: 0.48 }, { x: 2001, y: 0.81, y1:
0.05, y2: 0.53 },
                { x: 2002, y: 0.91, y1: 0.06, y2: 0.57 }, { x: 2003, y: 1, y1:
0.09, y2: 0.61 },
                { x: 2004, y: 1.19, y1: 0.14, y2: 0.63 }, { x: 2005, y: 1.47, y1:
0.20, y2: 0.64 },
                { x: 2006, y: 1.74, y1: 0.29, y2: 0.66 }, { x: 2007, y: 1.98, y1:
0.46, y2: 0.76 },
                { x: 2008, y: 1.99, y1: 0.64, y2: 0.77 }, { x: 2009, y: 1.70, y1:
0.75, y2: 0.55 }
            ],
        };
    },
    provide: {
        chart: [StackingBarSeries, Category]
    },
};
</script>
<style>
#container {
    height: 350px;
}
</style>

```

{% previewsample "page.domainurl/code-snippet/chart/series/bar-cs7" %}

### Series customization

The following properties can be used to customize the **stacked bar** series.

- [fill](#) – Specifies the color of the series.
- [opacity](#) – Specifies the opacity of [fill](#).
- [dashArray](#) – Specifies the dashes of series.
- [border](#) – Specifies the [color](#) and [width](#) of series border.

### APP.VUE

```

<template>
  <div id="app">

```

```

    <ejs-chart id="container" :title='title'
:primaryXAxis='primaryXAxis'>
      <e-series-collection>
        <e-series :dataSource='seriesData' type='StackingBar'
xName='x' yName='y' name='Apple' fill="red" :border="border" dashArray="5">
</e-series>
        <e-series :dataSource='seriesData' type='StackingBar'
xName='x' yName='y1' name='Orange' fill="yellow" :border="border"
dashArray="5"> </e-series>
        <e-series :dataSource='seriesData' type='StackingBar'
xName='x' yName='y2' name='Wastage' fill="blue" :border="border"
dashArray="5"> </e-series>
      </e-series-collection>
    </ejs-chart>
  </div>
</template>
<script>
import Vue from "vue";
import { ChartPlugin, StackingBarSeries, Category } from "@syncfusion/ej2-
vue-charts";
Vue.use(ChartPlugin);
export default {
  data: function() {
    return {
      seriesData:[
        { x: 'May', y: 20, y1: 21, y2: -3 }, { x: 'Jun', y: 24, y1:
25, y2: -3.5 },
        { x: 'Jul', y: 28, y1: 27, y2: -4 }, { x: 'Aug', y: 32, y1:
31, y2: -4.5 },
        { x: 'Sep', y: 33, y1: 34, y2: -5 }, { x: 'Oct', y: 35, y1:
34, y2: -5.5 },
        { x: 'Nov', y: 40, y1: 41, y2: -6 }, { x: 'Dec', y: 42, y1:
42, y2: -6.5 }
      ],
      primaryXAxis: {
        valueType: 'Category',
        title: 'Months'
      },
      title: "Sales Comparison",
      border: {
        width: 2,
        color: 'black'
      }
    };
  },
  provide: {
    chart: [StackingBarSeries, Category]
  },
};
</script>
<style>
#container {
  height: 350px;
}
</style>

```

```
{% previewsample "page.domainurl/code-snippet/chart/series/stackbar-cs" %}
```

*See also*

- [Data label](#)
- [Tooltip](#)

## 100% Stacked Bar in Vue Chart Component

### 100% Stacked bar

To render a 100% stacked bar series, use series [type](#) as `StackingBar100` and inject `StackingBarSeries` into the `provide`.

### APP.VUE

```
<template>
  <div id="app">
    <ejs-chart id="container" :title='title'
:primaryXAxis='primaryXAxis' :primaryYAxis='primaryYAxis'>
      <e-series-collection>
        <e-series :dataSource='seriesData' type='StackingBar100'
xName='x' yName='y' name='Apple'> </e-series>
        <e-series :dataSource='seriesData' type='StackingBar100'
xName='x' yName='y1' name='Orange'> </e-series>
        <e-series :dataSource='seriesData' type='StackingBar100'
xName='x' yName='y2' name='Wastage'> </e-series>
      </e-series-collection>
    </ejs-chart>
  </div>
</template>
<script>
import Vue from "vue";
import { ChartPlugin, StackingBarSeries, Category } from "@syncfusion/ej2-
vue-charts";
Vue.use(ChartPlugin);
export default {
  data: function() {
    return {
      seriesData:[
        { x: 'May', y: 20, y1: 21, y2: -3 }, { x: 'Jun', y: 24, y1:
25, y2: -3.5 },
        { x: 'Jul', y: 28, y1: 27, y2: -4 }, { x: 'Aug', y: 32, y1:
31, y2: -4.5 },
        { x: 'Sep', y: 33, y1: 34, y2: -5 }, { x: 'Oct', y: 35, y1:
34, y2: -5.5 },
        { x: 'Nov', y: 40, y1: 41, y2: -6 }, { x: 'Dec', y: 42, y1:
42, y2: -6.5 }
      ],
      primaryXAxis: {
        valueType: 'Category',
        title: 'Months'
      },
      primaryYAxis:{
        minimum: 0, maximum:100
      },
      title: "Sales Comparison"
    }
  }
}
```

```

    };
  },
  provide: {
    chart: [StackingBarSeries, Category]
  },
};
</script>
<style>
  #container {
    height: 350px;
  }
</style>

```

{% previewsample "page.domainurl/code-snippet/chart/series/bar-cs3" %}

### 100% Cylindrical stacked bar chart

To render a 100% cylindrical stacked bar chart, set the [columnFacet](#) property to `Cylinder` in the chart series.

### APP.VUE

```

<template>
  <div id="app">
    <ejs-chart id="container">
      <e-series-collection>
        <e-series :dataSource='seriesData' type='StackingBar100'
columnFacet='Cylinder' xName='x' yName='y'> </e-series>
        <e-series :dataSource='seriesData' type='StackingBar100'
columnFacet='Cylinder' xName='x' yName='y1'> </e-series>
        <e-series :dataSource='seriesData' type='StackingBar100'
columnFacet='Cylinder' xName='x' yName='y2'> </e-series>
      </e-series-collection>
    </ejs-chart>
  </div>
</template>
<script>
import Vue from "vue";
import { ChartPlugin, StackingBarSeries, Category } from "@syncfusion/ej2-
vue-charts";
Vue.use(ChartPlugin);
export default {
  data: function() {
    return {
      seriesData:[
        { x: 2000, y: 0.61, y1: 0.03, y2: 0.48 }, { x: 2001, y: 0.81, y1:
0.05, y2: 0.53 },
        { x: 2002, y: 0.91, y1: 0.06, y2: 0.57 }, { x: 2003, y: 1, y1:
0.09, y2: 0.61 },
        { x: 2004, y: 1.19, y1: 0.14, y2: 0.63 }, { x: 2005, y: 1.47, y1:
0.20, y2: 0.64 },
        { x: 2006, y: 1.74, y1: 0.29, y2: 0.66 }, { x: 2007, y: 1.98, y1:
0.46, y2: 0.76 },
        { x: 2008, y: 1.99, y1: 0.64, y2: 0.77 }, { x: 2009, y: 1.70, y1:
0.75, y2: 0.55 }
      ]
    };
  }
};

```

```

    },
    provide: {
      chart: [StackingBarSeries, Category]
    },
  };
</script>
<style>
  #container {
    height: 350px;
  }
</style>

```

{% previewsample "page.domainurl/code-snippet/chart/series/bar-cs8" %}

#### Series customization

The following properties can be used to customize the 100% stacked bar series.

- [fill](#) – Specifies the color of the series.
- [opacity](#) – Specifies the opacity of [Fill](#).
- [dashArray](#) – Specifies the dashes of series.
- [border](#) – Specifies the [Color](#) and [Width](#) of series border.

#### APP.VUE

```

<template>
  <div id="app">
    <ejs-chart id="container" :title='title'
:primaryXAxis='primaryXAxis' :primaryYAxis='primaryYAxis'>
      <e-series-collection>
        <e-series :dataSource='seriesData' type='StackingBar100'
xName='x' yName='y' name='Apple' fill="red" dashArray="5" :border="border"
opacity='0.8'> </e-series>
        <e-series :dataSource='seriesData' type='StackingBar100'
xName='x' yName='y1' name='Orange' fill="green" dashArray="5"
:border="border" opacity='0.8'> </e-series>
        <e-series :dataSource='seriesData' type='StackingBar100'
xName='x' yName='y2' name='Wastage' fill="blue" dashArray="5"
:border="border" opacity='0.8'> </e-series>
      </e-series-collection>
    </ejs-chart>
  </div>
</template>
<script>
import Vue from "vue";
import { ChartPlugin, StackingBarSeries, Category } from "@syncfusion/ej2-
vue-charts";
Vue.use(ChartPlugin);
export default {
  data: function() {
    return {
      seriesData:[
        { x: 'May', y: 20, y1: 21, y2: -3 }, { x: 'Jun', y: 24, y1:
25, y2: -3.5 },
        { x: 'Jul', y: 28, y1: 27, y2: -4 }, { x: 'Aug', y: 32, y1:
31, y2: -4.5 },

```

```

        { x: 'Sep', y: 33, y1: 34, y2: -5 }, { x: 'Oct', y: 35, y1:
34, y2: -5.5 },
        { x: 'Nov', y: 40, y1: 41, y2: -6 }, { x: 'Dec', y: 42, y1:
42, y2: -6.5 }
    ],
    primaryXAxis: {
        valueType: 'Category',
        title: 'Months'
    },
    primaryYAxis:{
        minimum: 0
    },
    title: "Sales Comparison",
    border:{
        width: 2,
        color: 'black'
    }
    };
},
provide: {
    chart: [StackingBarSeries, Category]
},
};
</script>
<style>
    #container {
        height: 350px;
    }
</style>

```

{% previewsample "page.domainurl/code-snippet/chart/series/stackedbar-cs" %}

[See also](#)

- [Data label](#)
- [Tooltip](#)

## Scatter in Vue Chart Component

[Scatter](#)

To render a scatter series, use series [type](#) as `Scatter` and inject `ScatterSeries` into the `provide`.

### APP.VUE

```

<template>
    <div id="app">
        <ejs-chart id="container">
            <e-series-collection>
                <e-series :dataSource='seriesData1' type='Scatter' xName='x'
yName='y' name='Male' opacity=0.7> </e-series>
                <e-series :dataSource='seriesData2' type='Scatter' xName='x'
yName='y' name='Female' opacity=0.7> </e-series>
            </e-series-collection>
        </ejs-chart>
    </div>
</template>

```

```

<script>
import Vue from "vue";
import { ChartPlugin, ScatterSeries } from "@syncfusion/ej2-vue-charts";
Vue.use(ChartPlugin);
let series1: Object[] = [];
let series2: Object[] = [];
let point1: Object;
let value: number = 80;
let value1: number = 70;
let i: number;
for (i = 1; i < 50; i++) {
  if (Math.random() > 0.5) {
    value += Math.random();
  } else {
    value -= Math.random();
  }
  value = value < 60 ? 60 : value > 90 ? 90 : value;
  point1 = { x: 120 + (i / 2), y: value.toFixed(1) };
  series1.push(point1);
}
for (i = 1; i < 50; i++) {
  if (Math.random() > 0.5) {
    value1 += Math.random();
  } else {
    value1 -= Math.random();
  }
  value1 = value1 < 60 ? 60 : value1 > 90 ? 90 : value1;
  point1 = { x: 120 + (i / 2), y: value1.toFixed(1) };
  series2.push(point1);
}
export default {
  data() {
    return {
      seriesData1: series1,
      seriesData2: series2
    };
  },
  provide: {
    chart: [ScatterSeries]
  },
};
</script>
<style>
#container {
  height: 350px;
}
</style>

```

{% previewsample "page.domainurl/code-snippet/chart/series/scatter-cs1" %}

#### Series customization

The following properties can be used to customize the `scatter` series.

- [fill](#) – Specifies the color of the series.
- [opacity](#) – Specifies the opacity of [fill](#).

- [shape](#) - Specifies the shape of the scatter series.

**APP.VUE**

```

<template>
  <div id="app">
    <ejs-chart id="container">
      <e-series-collection>
        <e-series :dataSource='seriesData1' type='Scatter' xName='x'
yName='y' name='Male' opacity=0.7 fill="red" :marker="marker"> </e-series>
        <e-series :dataSource='seriesData2' type='Scatter' xName='x'
yName='y' name='Female' opacity=0.7 fill="green" :marker="marker"> </e-
series>
      </e-series-collection>
    </ejs-chart>
  </div>
</template>
<script>
import Vue from "vue";
import { ChartPlugin, ScatterSeries } from "@syncfusion/ej2-vue-charts";
Vue.use(ChartPlugin);
let series1 = [];
let series2 = [];
let point1;
let value = 80;
let value1 = 70;
let i;
for (i = 1; i < 50; i++) {
  if (Math.random() > 0.5) {
    value += Math.random();
  } else {
    value -= Math.random();
  }
  value = value < 60 ? 60 : value > 90 ? 90 : value;
  point1 = { x: 120 + (i / 2), y: value.toFixed(1) };
  series1.push(point1);
}
for (i = 1; i < 50; i++) {
  if (Math.random() > 0.5) {
    value1 += Math.random();
  } else {
    value1 -= Math.random();
  }
  value1 = value1 < 60 ? 60 : value1 > 90 ? 90 : value1;
  point1 = { x: 120 + (i / 2), y: value1.toFixed(1) };
  series2.push(point1);
}
export default {
  data() {
    return {
      seriesData1: series1,
      seriesData2: series2,
      marker:{
        shape: 'Triangle',
        height: 7,
        width: 7
      }
    }
  }
}

```



```

    }
  };
},
provide: {
  chart: [ScatterSeries]
},
};
</script>
<style>
#container {
  height: 350px;
}
</style>

```

{% previewsample "page.domainurl/code-snippet/chart/series/scatter-cs" %}

[See Also](#)

- [Data label](#)
- [Tooltip](#)

## Bubble in Vue Chart Component

### Bubble

To render a bubble series, use series [type](#) as **Bubble** and inject **BubbleSeries** into the **provide**.

### APP.VUE

```

<template>
  <div id="app">
    <ejs-chart id="container">
      <e-series-collection>
        <e-series :dataSource='seriesData1' type='Bubble'
size='size' xName='x' yName='y' name='pound'> </e-series>
      </e-series-collection>
    </ejs-chart>
  </div>
</template>
<script>
import Vue from "vue";
import { ChartPlugin, BubbleSeries } from "@syncfusion/ej2-vue-charts";
Vue.use(ChartPlugin);
export default {
  data() {
    return {
      seriesData1: [
        { x: 92.2, y: 7.8, size: 1.347, text: 'China' },
        { x: 74, y: 6.5, size: 1.241, text: 'India' },
        { x: 90.4, y: 6.0, size: 0.238, text: 'Indonesia' },
        { x: 99.4, y: 2.2, size: 0.312, text: 'US' },
        { x: 88.6, y: 1.3, size: 0.197, text: 'Brazil' },
        { x: 99, y: 0.7, size: 0.0818, text: 'Germany' },
        { x: 72, y: 2.0, size: 0.0826, text: 'Egypt' },
        { x: 99.6, y: 3.4, size: 0.143, text: 'Russia' },
        { x: 99, y: 0.2, size: 0.128, text: 'Japan' },
        { x: 86.1, y: 4.0, size: 0.115, text: 'Mexico' },
      ]
    }
  }
}

```

```

        { x: 92.6, y: 6.6, size: 0.096, text: 'Philippines' },
        { x: 61.3, y: 14.5, size: 0.162, text: 'Nigeria' }]
    };
},
provide: {
    chart: [BubbleSeries]
},
};
</script>
<style>
    #container {
        height: 350px;
    }
</style>

```

{% previewsample "page.domainurl/code-snippet/chart/series/bubble-cs1" %}

### [Size mapping](#)

size property can be used to map the size value specified in data source.

### **APP.VUE**

```

<template>
    <div id="app">
        <ejs-chart id="container">
            <e-series-collection>
                <e-series :dataSource='seriesData1' type='Bubble'
size='size' xName='x' yName='y' name='pound'> </e-series>
            </e-series-collection>
        </ejs-chart>
    </div>
</template>
<script>
import Vue from "vue";
import { ChartPlugin, BubbleSeries } from "@syncfusion/ej2-vue-charts";
Vue.use(ChartPlugin);
export default {
    data() {
        return {
            seriesData1: [
                { x: 92.2, y: 7.8, size: 1.347, text: 'China' },
                { x: 74, y: 6.5, size: 1.241, text: 'India' },
                { x: 90.4, y: 6.0, size: 0.238, text: 'Indonesia' },
                { x: 99.4, y: 2.2, size: 0.312, text: 'US' },
                { x: 88.6, y: 1.3, size: 0.197, text: 'Brazil' },
                { x: 99, y: 0.7, size: 0.0818, text: 'Germany' },
                { x: 72, y: 2.0, size: 0.0826, text: 'Egypt' },
                { x: 99.6, y: 3.4, size: 0.143, text: 'Russia' },
                { x: 99, y: 0.2, size: 0.128, text: 'Japan' },
                { x: 86.1, y: 4.0, size: 0.115, text: 'Mexico' },
                { x: 92.6, y: 6.6, size: 0.096, text: 'Philippines' },
                { x: 61.3, y: 14.5, size: 0.162, text: 'Nigeria' }]
            };
        },
        provide: {
            chart: [BubbleSeries]

```

```

    },
  };
</script>
<style>
  #container {
    height: 350px;
  }
</style>

```

{% previewsample "page.domainurl/code-snippet/chart/series/bubble-cs2" %}

#### Series customization

The following properties can be used to customize the **bubble** series.

- [fill](#) – Specifies the color of the series.
- [opacity](#) – Specifies the opacity of [fill](#).

#### APP.VUE

```

<template>
  <div id="app">
    <ejs-chart id="container">
      <e-series-collection>
        <e-series :dataSource='seriesData1' type='Bubble'
size='size' xName='x' yName='y' name='pound' fill="blue" :border="border">
      </e-series>
    </e-series-collection>
  </ejs-chart>
</div>
</template>
<script>
import Vue from "vue";
import { ChartPlugin, BubbleSeries } from "@syncfusion/ej2-vue-charts";
Vue.use(ChartPlugin);
export default {
  data() {
    return {
      seriesData1: [
        { x: 92.2, y: 7.8, size: 1.347, text: 'China' },
        { x: 74, y: 6.5, size: 1.241, text: 'India' },
        { x: 90.4, y: 6.0, size: 0.238, text: 'Indonesia' },
        { x: 99.4, y: 2.2, size: 0.312, text: 'US' },
        { x: 88.6, y: 1.3, size: 0.197, text: 'Brazil' },
        { x: 99, y: 0.7, size: 0.0818, text: 'Germany' },
        { x: 72, y: 2.0, size: 0.0826, text: 'Egypt' },
        { x: 99.6, y: 3.4, size: 0.143, text: 'Russia' },
        { x: 99, y: 0.2, size: 0.128, text: 'Japan' },
        { x: 86.1, y: 4.0, size: 0.115, text: 'Mexico' },
        { x: 92.6, y: 6.6, size: 0.096, text: 'Philippines' },
        { x: 61.3, y: 14.5, size: 0.162, text: 'Nigeria' }],
      border:{
        width: 2,
        color : 'black'
      }
    };
  }
};

```

```

    },
    provide: {
      chart: [BubbleSeries]
    },
  };
</script>
<style>
  #container {
    height: 350px;
  }
</style>

```

{% previewsample "page.domainurl/code-snippet/chart/series/bubble-cs" %}

[See Also](#)

- [Data label](#)
- [Tooltip](#)

## Polar in Vue Chart Component

### [Polar Chart](#)

To render a polar series, use series [type](#) as **Polar** and inject **PolarSeries** into the **provide**.

### [Draw Types](#)

Polar drawType property is used to change the series plotting type to line, column, area, range column, spline,

scatter, stacking area and stacking column. The default value of drawType is **Line**.

### [Line](#)

To render a line draw type, use series [drawType](#) as **Line** and inject

**LineSeries** inject **LineSeries** into the **provide**.

[isClosed](#) property specifies whether to join start and end point of

a line series used in polar chart to form a closed path. Default value of isClosed is true.

## **APP.VUE**

```

<template>
  <div id="app">
    <ejs-chart id="container" :title='title'
    :primaryXAxis='primaryXAxis' :primaryYAxis='primaryYAxis'>
      <e-series-collection>
        <e-series :dataSource='seriesData' type='Polar' xName='x'
        yName='y' drawType='Line'> </e-series>
      </e-series-collection>
    </ejs-chart>
  </div>
</template>
<script>
import Vue from "vue";
import { ChartPlugin, Tooltip, Legend, PolarSeries, Category, LineSeries,
RadarSeries } from "@syncfusion/ej2-vue-charts";
Vue.use(ChartPlugin);

```

```

export default {
  data() {
    return {
      seriesData:[
        { x: 2005, y: 28 }, { x: 2006, y: 25 }, { x: 2007, y: 26 },
        { x: 2008, y: 27 }, { x: 2009, y: 32 }, { x: 2010, y: 35 },
        { x: 2011, y: 30 }],
      primaryXAxis: {
        title: 'Year',
        minimum: 2004, maximum: 2012, interval: 1
      },
      primaryYAxis: {
        minimum: 20, maximum: 40, interval: 5,
        title: 'Efficiency',
        labelFormat: '{value}%'
      },
      title: "Efficiency of oil-fired power production"
    };
  },
  provide: {
    chart: [Tooltip, Legend, PolarSeries, Category, LineSeries, RadarSeries]
  },
};
</script>
<style>
  #container {
    height: 350px;
  }
</style>

```

{% previewsample "page.domainurl/code-snippet/chart/series/polar-cs3" %}

### Spline

To render a spline line draw type, use series [drawType](#) as **Spline**

and inject **SplineSeries** inject **SplineSeries** into the **provide**.

### APP.VUE

```

<template>
  <div id="app">
    <ejs-chart id="container" :title='title'
    :primaryXAxis='primaryXAxis' :primaryYAxis='primaryYAxis'>
      <e-series-collection>
        <e-series :dataSource='seriesData' type='Polar' xName='x'
        yName='y' drawType='Spline'> </e-series>
      </e-series-collection>
    </ejs-chart>
  </div>
</template>
<script>
import Vue from "vue";
import { Tooltip, Legend, PolarSeries, Category, SplineSeries, RadarSeries,
ChartPlugin } from "@syncfusion/ej2-vue-charts";
Vue.use(CharPlugin);
export default {
  data() {

```

```

return {
  seriesData:[
    { x: 'Jan', y: -1 }, { x: 'Feb', y: -1 }, { x: 'Mar', y: 2 },
    { x: 'Apr', y: 8 }, { x: 'May', y: 13 }, { x: 'Jun', y: 18 },
    { x: 'Jul', y: 21 }, { x: 'Aug', y: 20 }, { x: 'Sep', y: 16 },
    { x: 'Oct', y: 10 }, { x: 'Nov', y: 4 }, { x: 'Dec', y: 0 }
  ],
  primaryXAxis: {
    valueType: 'Category'
  },
  primaryYAxis: {
    minimum: -5, maximum: 35, interval: 10,
    title: 'Temperature in Celsius',
    labelFormat: '{value}C'
  },
  title: "Climate Graph-2012"
};
},
provide: {
  chart: [Tooltip, Legend, PolarSeries, Category, SplineSeries,
  RadarSeries]
},
};
</script>
<style>
#container {
  height: 350px;
}
</style>

```

{% previewsample "page.domainurl/code-snippet/chart/series/polar-cs4" %}

### Area

To render a area line draw type, use series [drawType](#) as `Area` and inject `AreaSeries` into the `provide`.

### APP.VUE

```

<template>
  <div id="app">
    <ejs-chart id="container" :title='title'
    :primaryXAxis='primaryXAxis' :primaryYAxis='primaryYAxis'>
      <e-series-collection>
        <e-series :dataSource='seriesData' type='Polar' xName='x'
        yName='y' drawType='Area'> </e-series>
      </e-series-collection>
    </ejs-chart>
  </div>
</template>
<script>
import Vue from "vue";
import { ChartPlugin, PolarSeries, Category, AreaSeries } from
"@syncfusion/ej2-vue-charts";
Vue.use(ChartPlugin);
export default {
  data() {

```

```

return {
  seriesData:[
    { x: 'Jan', y: -1 }, { x: 'Feb', y: -1 }, { x: 'Mar', y: 2 },
    { x: 'Apr', y: 8 }, { x: 'May', y: 13 }, { x: 'Jun', y: 18 },
    { x: 'Jul', y: 21 }, { x: 'Aug', y: 20 }, { x: 'Sep', y: 16 },
    { x: 'Oct', y: 10 }, { x: 'Nov', y: 4 }, { x: 'Dec', y: 0 }
  ],
  primaryXAxis: {
    valueType: 'Category'
  },
  primaryYAxis: {
    minimum: -5, maximum: 35, interval: 10,
    title: 'Temperature in Celsius',
    labelFormat: '{value}C'
  },
  title: "Climate Graph-2012"
};
},
provide: {
  chart: [PolarSeries, Category, AreaSeries]
}
};
</script>
<style>
  #container {
    height: 350px;
  }
</style>

```

{% previewsample "page.domainurl/code-snippet/chart/series/polar-cs5" %}

### Stacked Area

To render a stacked area draw type, use series [drawType](#) as `StackingArea` and inject `StackingAreaSeries`

### APP.VUE

```

<template>
  <div id="app">
    <ejs-chart id="container" :title='title'
:primaryXAxis='primaryXAxis'>
      <e-series-collection>
        <e-series :dataSource='seriesData' type='Polar' xName='x'
yName='y' drawType='StackingArea' name='Organic'></e-series>
        <e-series :dataSource='seriesData' type='Polar' xName='x'
yName='y1' drawType='StackingArea' name='Fair-trade'></e-series>
        <e-series :dataSource='seriesData' type='Polar' xName='x'
yName='y' drawType='StackingArea' name='veg-Alternatives'></e-series>
      </e-series-collection>
    </ejs-chart>
  </div>
</template>
<script>
import Vue from "vue";
import { ChartPlugin, PolarSeries, Category, StackingAreaSeries } from
"@syncfusion/ej2-vue-charts";

```

```

Vue.use(ChartPlugin);
export default {
  data() {
    return {
      seriesData:[
        { x: '2000', y: 0.61, y1: 0.03, y2: 0.48},
        { x: '2001', y: 0.81, y1: 0.05, y2: 0.53 },
        { x: '2002', y: 0.91, y1: 0.06, y2: 0.57 },
        { x: '2003', y: 1, y1: 0.09, y2: 0.61 },
        { x: '2004', y: 1.19, y1: 0.14, y2: 0.63 },
        { x: '2005', y: 1.47, y1: 0.20, y2: 0.64 },
        { x: '2006', y: 1.74, y1: 0.29, y2: 0.66 },
        { x: '2007', y: 1.98, y1: 0.46, y2: 0.76 },
        { x: '2008', y: 1.99, y1: 0.64, y2: 0.77 },
        { x: '2009', y: 1.70, y1: 0.75, y2: 0.55 }
      ],
      primaryXAxis: {
        valueType: 'Category'
      },
      title: "Trend in Sales of Ethical Produce"
    };
  },
  provide: {
    chart: [PolarSeries, Category, StackingAreaSeries]
  }
};
</script>
<style>
#container {
  height: 350px;
}
</style>

```

{% previewsample "page.domainurl/code-snippet/chart/series/polar-cs6" %}

### Column

To render a column draw type, use series [drawType](#) as `Column` and inject `ColumnSeries` into the `provide`.

### APP.VUE

```

<template>
  <div id="app">
    <ejs-chart id="container" :title='title'
:primaryXAxis='primaryXAxis'>
      <e-series-collection>
        <e-series :dataSource='seriesData' type='Polar'
xName='country' yName='gold' drawType='Column' name='Gold'> </e-series>
      </e-series-collection>
    </ejs-chart>
  </div>
</template>
<script>
import Vue from "vue";
import { ChartPlugin, PolarSeries, ColumnSeries, Category } from
"@syncfusion/ej2-vue-charts";

```



```

Vue.use(ChartPlugin);
export default {
  data() {
    return {
      seriesData: [
        { country: "USA", gold: 50 },
        { country: "China", gold: 40 },
        { country: "Japan", gold: 70 },
        { country: "Australia", gold: 60 },
        { country: "France", gold: 50 },
        { country: "Germany", gold: 40 },
        { country: "Italy", gold: 40 },
        { country: "Sweden", gold: 30 }
      ],
      primaryXAxis: {
        valueType: 'Category',
        title: 'Countries'
      },
      title: "Olympic Medals"
    };
  },
  provide: {
    chart: [ColumnSeries, Category, PolarSeries]
  }
};
</script>
<style>
  #container {
    height: 350px;
  }
</style>

```

{% previewsample "page.domainurl/code-snippet/chart/series/polar-cs7" %}

### Stacked Column

To render a stacked column draw type, use series [drawType](#) as `StackingColumn` and inject `StackingColumnSeries` into the `provide`.

### APP.VUE

```

<template>
  <div id="app">
    <ejs-chart id="container" :title='title'
    :primaryXAxis='primaryXAxis'>
      <e-series-collection>
        <e-series :dataSource='seriesData' type='Polar' xName='x'
        yName='y' drawType='StackingColumn' name='Organic'></e-series>
        <e-series :dataSource='seriesData' type='Polar' xName='x'
        yName='y1' drawType='StackingColumn' name='Fair-trade'></e-series>
        <e-series :dataSource='seriesData' type='Polar' xName='x'
        yName='y' drawType='StackingColumn' name='veg-Alternatives'></e-series>
      </e-series-collection>
    </ejs-chart>
  </div>
</template>
<script>

```

```

import Vue from "vue";
import { ChartPlugin, PolarSeries, Category, StackingColumnSeries } from
"@syncfusion/ej2-vue-charts";
Vue.use(ChartPlugin);
export default {
  data() {
    return {
      seriesData:[
        { x: '2014', y: 111.1, y1: 76.9, y2: 66.1, y3: 34.1 },
        { x: '2015', y: 127.3, y1: 99.5, y2: 79.3, y3: 38.2 },
        { x: '2016', y: 143.4, y1: 121.7, y2: 91.3, y3: 44.0 },
        { x: '2017', y: 159.9, y1: 142.5, y2: 102.4, y3: 51.6 },
        { x: '2018', y: 175.4, y1: 166.7, y2: 112.9, y3: 61.9 },
        { x: '2019', y: 189.0, y1: 182.9, y2: 122.4, y3: 71.5 },
        { x: '2020', y: 202.7, y1: 197.3, y2: 120.9, y3: 82.0 }
      ],
      primaryXAxis: {
        valueType: 'Category'
      },
      title: "Trend in Sales of Ethical Produce"
    };
  },
  provide: {
    chart: [PolarSeries, Category, StackingColumnSeries]
  }
};
</script>
<style>
  #container {
    height: 350px;
  }
</style>

```

{% previewsample "page.domainurl/code-snippet/chart/series/polar-cs8" %}

### Range Column

To render a range column draw type, use series [drawType](#) as `RangeColumn` and inject `RangeColumnSeries` into the `provide`.

### APP.VUE

```

<template>
  <div id="app">
    <ejs-chart id="container" :title='title'
:primaryXAxis='primaryXAxis'>
      <e-series-collection>
        <e-series :dataSource='seriesData' type='Polar' xName='x'
high='high' low='low' drawType='RangeColumn' name='Gold'> </e-series>
      </e-series-collection>
    </ejs-chart>
  </div>
</template>
<script>
import Vue from "vue";
import { ChartPlugin, PolarSeries, RangeColumnSeries, Category } from
"@syncfusion/ej2-vue-charts";

```

```

Vue.use(ChartPlugin);
export default {
  data() {
    return {
      seriesData: [
        { x: 'Jan', low: 0.7, high: 6.1 }, { x: 'Feb', low: 1.3, high: 6.3 },
        { x: 'Mar', low: 1.9, high: 8.5 },
        { x: 'Apr', low: 3.1, high: 10.8 }, { x: 'May', low: 5.7, high: 14.40 },
        { x: 'Jun', low: 8.4, high: 16.90 },
        { x: 'Jul', low: 10.6, high: 19.20 }, { x: 'Aug', low: 10.5, high: 18.9 },
        { x: 'Sep', low: 8.5, high: 16.1 },
        { x: 'Oct', low: 6.0, high: 12.5 }, { x: 'Nov', low: 1.5, high: 6.9 },
        { x: 'Dec', low: 5.1, high: 12.1 }
      ],
      primaryXAxis: {
        valueType: 'Category',
        title: 'Months'
      },
      title: "Maximum and Minimum Temperature"
    };
  },
  provide: {
    chart: [RangeColumnSeries, Category, PolarSeries]
  }
};
</script>
<style>
#container {
  height: 350px;
}
</style>

```

{% previewsample "page.domainurl/code-snippet/chart/series/polar-cs9" %}

### Scatter

To render a scatter draw type, use series [drawType](#) as **Scatter** and inject **ScatterSeries** into the **provide**.

### APP.VUE

```

<template>
  <div id="app">
    <ejs-chart id="container" :title='title'
    :primaryXAxis='primaryXAxis' :primaryYAxis='primaryYAxis'>
      <e-series-collection>
        <e-series :dataSource='seriesData' type='Polar' xName='x'
        yName='y' drawType='Scatter' name='London'> </e-series>
      </e-series-collection>
    </ejs-chart>
  </div>
</template>
<script>
import Vue from "vue";
import { ChartPlugin, PolarSeries, Category, ScatterSeries } from
"@syncfusion/ej2-vue-charts";
Vue.use(ChartPlugin);

```

```

export default {
  data() {
    return {
      seriesData:[
        { x: 'Jan', y: -1 }, { x: 'Feb', y: -1 }, { x: 'Mar', y: 2 },
        { x: 'Apr', y: 8 }, { x: 'May', y: 13 }, { x: 'Jun', y: 18 },
        { x: 'Jul', y: 21 }, { x: 'Aug', y: 20 }, { x: 'Sep', y: 16 },
        { x: 'Oct', y: 10 }, { x: 'Nov', y: 4 }, { x: 'Dec', y: 0 }
      ],
      primaryXAxis: {
        valueType: 'Category'
      },
      primaryYAxis: {
        minimum: -5, maximum: 35, interval: 10,
        title: 'Temperature in Celsius',
        labelFormat: '{value}C'
      },
      title: "Climate Graph-2012"
    };
  },
  provide: {
    chart: [PolarSeries, Category, ScatterSeries]
  }
};
</script>
<style>
  #container {
    height: 350px;
  }
</style>

```

{% previewsample "page.domainurl/code-snippet/chart/series/polar-cs10" %}

### [Series customization](#)

#### [Start Angle](#)

You can customize the start angle of the polar series using [startAngle](#) property. By default, `startAngle` is 0 degree.

### **APP.VUE**

```

<template>
  <div id="app">
    <ejs-chart id="container" :title='title'
    :primaryXAxis='primaryXAxis' :primaryYAxis='primaryYAxis'>
      <e-series-collection>
        <e-series :dataSource='seriesData' type='Polar' xName='x'
        yName='y' drawType='Line'> </e-series>
      </e-series-collection>
    </ejs-chart>
  </div>
</template>
<script>
import Vue from "vue";
import { ChartPlugin, PolarSeries, LineSeries } from "@syncfusion/ej2-vue-
charts";
Vue.use(ChartPlugin);

```

```

export default {
  data() {
    return {
      seriesData:[
        { x: 2005, y: 28 }, { x: 2006, y: 25 }, { x: 2007, y: 26 },
        { x: 2008, y: 27 }, { x: 2009, y: 32 }, { x: 2010, y: 35 },
        { x: 2011, y: 30 }],
      primaryXAxis: {
        title: 'Year', startAngle: 90,
        minimum: 2004, maximum: 2012, interval: 1, coefficient: 40
      },
      primaryYAxis: {
        minimum: 20, maximum: 40, interval: 5,
        title: 'Efficiency',
        labelFormat: '{value}%'
      },
      title: "Efficiency of oil-fired power production"
    };
  },
  provide: {
    chart: [PolarSeries, LineSeries]
  },
};
</script>
<style>
  #container {
    height: 350px;
  }
</style>

```

{% previewsample "page.domainurl/code-snippet/chart/series/polar-cs0" %}

#### Coefficient in axis

You can customize the radius of the polar series using [coefficient](#) property. By default, `coefficient` is 100.

#### APP.VUE

```

<template>
  <div id="app">
    <ejs-chart id="container" :title='title'
    :primaryXAxis='primaryXAxis' :primaryYAxis='primaryYAxis'>
      <e-series-collection>
        <e-series :dataSource='seriesData' type='Polar' xName='x'
        yName='y' drawType='Line'> </e-series>
      </e-series-collection>
    </ejs-chart>
  </div>
</template>
<script>
import Vue from "vue";
import { ChartPlugin, PolarSeries, LineSeries } from "@syncfusion/ej2-vue-
charts";
Vue.use(ChartPlugin);
export default {
  data() {

```

```

return {
  seriesData:[
    { x: 2005, y: 28 }, { x: 2006, y: 25 }, { x: 2007, y: 26 },
    { x: 2008, y: 27 }, { x: 2009, y: 32 }, { x: 2010, y: 35 },
    { x: 2011, y: 30 }],
    primaryXAxis: {
      title: 'Year', coefficient: 50,
      minimum: 2004, maximum: 2012, interval: 1, coefficient: 40
    },
    primaryYAxis: {
      minimum: 20, maximum: 40, interval: 5,
      title: 'Efficiency',
      labelFormat: '{value}%'
    },
    title: "Efficiency of oil-fired power production"
  ];
},
provide: {
  chart: [PolarSeries, LineSeries]
},
};
</script>
<style>
  #container {
    height: 350px;
  }
</style>

```

{% previewsample "page.domainurl/code-snippet/chart/series/polar-cs01" %}

[See Also](#)

- [Data label](#)
- [Tooltip](#)

## Radar in Vue Chart Component

### [Radar Chart](#)

To render a radar series, use series [type](#) as **Radar** and to render a line draw type, use series [drawType](#) as **Line** and inject

**LineSeries** inject **LineSeries** into the **provide**. [isClosed](#) property specifies whether to join start and end point of a line series used in polar chart to form a closed path. Default value of isClosed is true.

### **APP.VUE**

```

<template>
  <div id="app">
    <ejs-chart id="container" :title='title'
    :primaryXAxis='primaryXAxis' :primaryYAxis='primaryYAxis'>
      <e-series-collection>
        <e-series :dataSource='seriesData' type='Radar' xName='x'
        yName='y' drawType='Line'> </e-series>
      </e-series-collection>
    </ejs-chart>
  </div>

```

```

</template>
<script>
import Vue from "vue";
import { ChartPlugin, RadarSeries, LineSeries } from "@syncfusion/ej2-vue-charts";
Vue.use(ChartPlugin);
export default {
  data() {
    return {
      seriesData:[
        { x: 2005, y: 28 }, { x: 2006, y: 25 }, { x: 2007, y: 26 },
        { x: 2008, y: 27 }, { x: 2009, y: 32 }, { x: 2010, y: 35 },
        { x: 2011, y: 30 }],
      primaryXAxis: {
        title: 'Year',
        minimum: 2004, maximum: 2012, interval: 1
      },
      primaryYAxis: {
        minimum: 20, maximum: 40, interval: 5,
        title: 'Efficiency',
        labelFormat: '{value}%'
      },
      title: "Efficiency of oil-fired power production"
    };
  },
  provide: {
    chart: [RadarSeries, LineSeries]
  },
};
</script>
<style>
#container {
  height: 350px;
}
</style>

```

{% previewsample "page.domainurl/code-snippet/chart/series/polar-cs11" %}

### Series customization

#### Start Angle

You can customize the start angle of the radar series using [startAngle](#) property. By default, `startAngle` is 0 degree.

### APP.VUE

```

<template>
  <div id="app">
    <ejs-chart id="container" :title='title'
    :primaryXAxis='primaryXAxis' :primaryYAxis='primaryYAxis'>
      <e-series-collection>
        <e-series :dataSource='seriesData' type='Radar' xName='x'
        yName='y' drawType='Line'> </e-series>
      </e-series-collection>
    </ejs-chart>
  </div>
</template>

```

```

<script>
import Vue from "vue";
import { ChartPlugin, RadarSeries, LineSeries } from "@syncfusion/ej2-vue-charts";
Vue.use(ChartPlugin);
export default {
  data() {
    return {
      seriesData: [
        { x: 2005, y: 28 }, { x: 2006, y: 25 }, { x: 2007, y: 26 },
        { x: 2008, y: 27 }, { x: 2009, y: 32 }, { x: 2010, y: 35 },
        { x: 2011, y: 30 }],
      primaryXAxis: {
        title: 'Year', startAngle: 90,
        minimum: 2004, maximum: 2012, interval: 1
      },
      primaryYAxis: {
        minimum: 20, maximum: 40, interval: 5,
        title: 'Efficiency',
        labelFormat: '{value}%'
      },
      title: "Efficiency of oil-fired power production"
    };
  },
  provide: {
    chart: [RadarSeries, LineSeries]
  },
};
</script>
<style>
#container {
  height: 350px;
}
</style>

```

{% previewsample "page.domainurl/code-snippet/chart/series/polar-cs12" %}

### Coefficient in axis

You can customize the radius of the radar series using [coefficient](#) property. By default, `coefficient` is 100.

### APP.VUE

```

<template>
  <div id="app">
    <ejs-chart id="container" :title='title'
    :primaryXAxis='primaryXAxis' :primaryYAxis='primaryYAxis'>
      <e-series-collection>
        <e-series :dataSource='seriesData' type='Radar' xName='x'
        yName='y' drawType='Line'> </e-series>
      </e-series-collection>
    </ejs-chart>
  </div>
</template>
<script>
import Vue from "vue";

```



```

import { ChartPlugin, RadarSeries, LineSeries } from "@syncfusion/ej2-vue-charts";
Vue.use(ChartPlugin);
export default {
  data() {
    return {
      seriesData:[
        { x: 2005, y: 28 }, { x: 2006, y: 25 }, { x: 2007, y: 26 },
        { x: 2008, y: 27 }, { x: 2009, y: 32 }, { x: 2010, y: 35 },
        { x: 2011, y: 30 }],
      primaryXAxis: {
        title: 'Year', coefficient: 50,
        minimum: 2004, maximum: 2012, interval: 1
      },
      primaryYAxis: {
        minimum: 20, maximum: 40, interval: 5,
        title: 'Efficiency',
        labelFormat: '{value}%'
      },
      title: "Efficiency of oil-fired power production"
    };
  },
  provide: {
    chart: [RadarSeries, LineSeries]
  },
};
</script>
<style>
  #container {
    height: 350px;
  }
</style>

```

{% previewsample "page.domainurl/code-snippet/chart/series/polar-cs13" %}

[See Also](#)

- [Data label](#)
- [Tooltip](#)

## Hilo in Vue Chart Component

### Hilo

Hilo Series illustrates the price movements in stock using the high and low values. To render a Hilo series, use series [type](#) as **Hilo** and inject **HiloSeries** into the **provide**. Hilo series requires 3 fields (x, high and low) to show the high and low price in the stock.

### APP.VUE

```

<template>
  <div id="app">
    <ejs-chart id="container" :title='title'
    :primaryXAxis='primaryXAxis' :primaryYAxis='primaryYAxis'>
      <e-series-collection>
        <e-series :dataSource='seriesData' type='Hilo' xName='x'
        high='high' low='low' name='India'> </e-series>
      </e-series-collection>
    </ejs-chart>
  </div>
</template>

```

```

        </e-series-collection>
    </ejs-chart>
</div>
</template>
<script>
import Vue from "vue";
import { ChartPlugin, HiloSeries, Category } from "@syncfusion/ej2-vue-
charts";
Vue.use(ChartPlugin);
export default {
    data() {
        return {
            seriesData: [
                { x: 'Jan', low: 87, high: 200 }, { x: 'Feb', low: 45, high: 135
            },
                { x: 'Mar', low: 19, high: 85 }, { x: 'Apr', low: 31, high: 108
            },
                { x: 'May', low: 27, high: 80 }, { x: 'June', low: 84, high: 130
            },
                { x: 'July', low: 77, high: 150 }, { x: 'Aug', low: 54, high:
125 },
                { x: 'Sep', low: 60, high: 155 }, { x: 'Oct', low: 60, high: 180
            },
                { x: 'Nov', low: 88, high: 180 }, { x: 'Dec', low: 84, high: 230
            }
        ],
        primaryXAxis: {
            valueType: 'Category',
            title: 'Months'
        },
        primaryYAxis: {
            labelFormat: '{value}mm',
            edgeLabelPlacement: 'Shift',
            title: 'Rainfall',
        },
        marker: {
            //Data label for chart series
            dataLabel: { visible: true }
        },
        title: "Maximum and Minimum Rainfall"
    };
    provide: {
        chart: [HiloSeries, Category]
    }
};
</script>
<style>
    #container {
        height: 350px;
    }
</style>

```

{% previewsample "page.domainurl/code-snippet/chart/series/hilo-cs1" %}

*Series customization*

The following properties can be used to customize the **hilo** series.

- [fill](#) – Specifies the color of the series.
- [opacity](#) – Specifies the opacity of [fill](#).

**APP.VUE**

```
<template>
  <div id="app">
    <ejs-chart id="container" :title='title'
:primaryXAxis='primaryXAxis' :primaryYAxis='primaryYAxis'>
      <e-series-collection>
        <e-series :dataSource='seriesData' type='Hilo' xName='x'
high='high' low='low' name='India' fill="blue"> </e-series>
      </e-series-collection>
    </ejs-chart>
  </div>
</template>
<script>
import Vue from "vue";
import { ChartPlugin, HiloSeries, Category } from "@syncfusion/ej2-vue-
charts";
Vue.use(ChartPlugin);
export default {
  data() {
    return {
      seriesData: [
        { x: 'Jan', low: 87, high: 200 }, { x: 'Feb', low: 45, high: 135
},
        { x: 'Mar', low: 19, high: 85 }, { x: 'Apr', low: 31, high: 108
},
        { x: 'May', low: 27, high: 80 }, { x: 'June', low: 84, high: 130
},
        { x: 'July', low: 77, high: 150 }, { x: 'Aug', low: 54, high:
125 },
        { x: 'Sep', low: 60, high: 155 }, { x: 'Oct', low: 60, high: 180
},
        { x: 'Nov', low: 88, high: 180 }, { x: 'Dec', low: 84, high: 230
}
      ],
      primaryXAxis: {
        valueType: 'Category',
        title: 'Months'
      },
      primaryYAxis: {
        labelFormat: '{value}mm',
        edgeLabelPlacement: 'Shift',
        title: 'Rainfall',
      },
      marker: {
        //Data label for chart series
        dataLabel: { visible: true }
      },
      title: "Maximum and Minimum Rainfall"
    };
  }
};
```

```

    },
    provide: {
      chart: [HiloSeries, Category]
    }
  };
</script>
<style>
  #container {
    height: 350px;
  }
</style>

```

{% previewsample "page.domainurl/code-snippet/chart/series/hilo-cs" %}

[See Also](#)

- [Data label](#)
- [Tooltip](#)

## High Low Open Close in Vue Chart Component

### [High Low Open Close](#)

HiloOpenClose series is used to represent the low, high, open and closing values over time.

To render a HiloOpenClose series, use series [type](#) as `HiloOpenClose` and

inject `HiloOpenCloseSeries` into the `provide`.

HiloOpenClose series requires 5 fields (x, high, low, open and close) to show the high, low, open and close price values in the stock.

### **APP.VUE**

```

<template>
  <div id="app">
    <ejs-chart id="container" :title='title'
    :primaryXAxis='primaryXAxis' :primaryYAxis='primaryYAxis'>
      <e-series-collection>
        <e-series :dataSource='seriesData' type='HiloOpenClose'
        xName='x' high='high' low='low' open='open' close='close' name='India'> </e-
        series>
      </e-series-collection>
    </ejs-chart>
  </div>
</template>
<script>
import Vue from "vue";
import { ChartPlugin, HiloOpenCloseSeries, Category } from "@syncfusion/ej2-
vue-charts";
Vue.use(ChartPlugin);
export default {
  data() {
    return {
      seriesData: [
        { x: 'Jan', open: 120, high: 160, low: 100, close: 140 },
        { x: 'Feb', open: 150, high: 190, low: 130, close: 170 },

```

```

        { x: 'Mar', open: 130, high: 170, low: 110, close: 150 },
        { x: 'Apr', open: 160, high: 180, low: 120, close: 140 },
        { x: 'May', open: 150, high: 170, low: 110, close: 130 }
    ],
    primaryXAxis: {
        title: 'Date',
        valueType: 'Category',
    },
    primaryYAxis: {
        title: 'Price in Dollar', minimum: 100, maximum: 200, interval:
20,
    },
    title: "Financial Analysis"
};
},
provide: {
    chart: [HiloOpenCloseSeries, Category]
}
};
</script>
<style>
    #container {
        height: 350px;
    }
</style>

```

{% previewsample "page.domainurl/code-snippet/chart/series/hiloopenclose-cs1" %}

#### Series customization

In HiloOpenClose series, [bullFillColor](#) is used to fill the

segment when the open value is greater than the close value and [bearFillColor](#) is used to fill the segment when the open value is less than the close value.

By default, bullFillColor is set as red and bearFillColor is set as green.

#### APP.VUE

```

<template>
    <div id="app">
        <ejs-chart id="container" :title='title'
:primaryXAxis='primaryXAxis':primaryYAxis='primaryYAxis'>
            <e-series-collection>
                <e-series :dataSource='seriesData' type='HiloOpenClose'
xName='x' high='high' low='low'
                    open='open' close='close' name='SHIRPUR-G' bearFillColor=
'green' bullFillColor= 'red'> </e-series>
            </e-series-collection>
        </ejs-chart>
    </div>
</template>
<script>
import Vue from "vue";
import { ChartPlugin, HiloOpenCloseSeries, Category } from "@syncfusion/ej2-
vue-charts";
Vue.use(ChartPlugin);
export default {

```

```

data() {
  return {
    seriesData: [
      { x: 'Jan', open: 120, high: 160, low: 100, close: 140 },
      { x: 'Feb', open: 150, high: 190, low: 130, close: 170 },
      { x: 'Mar', open: 130, high: 170, low: 110, close: 150 },
      { x: 'Apr', open: 160, high: 180, low: 120, close: 140 },
      { x: 'May', open: 150, high: 170, low: 110, close: 130 }
    ],
    primaryXAxis: {
      title: 'Date',
      valueType: 'Category',
    },
    primaryYAxis: {
      title: 'Price in Dollar', minimum: 100, maximum: 200, interval:
20,
    },
    title: "Financial Analysis"
  };
},
provide: {
  chart: [HiloOpenCloseSeries, Category]
}
};
</script>
<style>
#container {
  height: 350px;
}
</style>

```

{% previewsample "page.domainurl/code-snippet/chart/series/hiloopenclose-cs2" %}

[See Also](#)

- [Data label](#)
- [Tooltip](#)

## Candle in Vue Chart Component

### Candle

Candle series are similar to Hilo Open Close series, are used to represent the low, high, open and closing price over time. To render a candle series, use series [type](#) as `Candle` and inject `CandleSeries` into the `provide`.

### APP.VUE

```

<template>
  <div id="app">
    <ejs-chart id="container" :title='title'
:primaryXAxis='primaryXAxis' :primaryYAxis='primaryYAxis'>
      <e-series-collection>
        <e-series :dataSource='seriesData' type='Candle' xName='x'
high='high' low='low'
open='open' close='close' name='SHIRPUR-G'> </e-series>

```

```

        </e-series-collection>
    </ejs-chart>
</div>
</template>
<script>
import Vue from "vue";
import { ChartPlugin, CandleSeries, Category } from "@syncfusion/ej2-vue-
charts";
Vue.use(ChartPlugin);
export default {
    data() {
        return {
            seriesData: [
                { x: 'Jan', open: 120, high: 160, low: 100, close: 140 },
                { x: 'Feb', open: 150, high: 190, low: 130, close: 170 },
                { x: 'Mar', open: 130, high: 170, low: 110, close: 150 },
                { x: 'Apr', open: 160, high: 180, low: 120, close: 140 },
                { x: 'May', open: 150, high: 170, low: 110, close: 130 }
            ],
            primaryXAxis: {
                title: 'Date',
                valueType: 'Category',
            },
            primaryYAxis: {
                title: 'Price in Dollar', minimum: 100, maximum: 200, interval:
20,
            },
            title: "Financial Analysis"
        };
    },
    provide: {
        chart: [CandleSeries, Category]
    }
};
</script>
<style>
    #container {
        height: 350px;
    }
</style>

```

{% previewsample "page.domainurl/code-snippet/chart/series/candle-cs1" %}

### Hollow Candles

Candle charts allow to visually compare the current price with previous price by customizing the appearance.

Candles are filled/left as hollow based on the following criteria.

<!-- markdownlint-disable MD033 -->

States	Description
Filled	candle sticks are filled when the close value is lesser than the open value
Unfilled	candle sticks are unfilled when the close value is greater than the open value

The color of the candle will be defined by comparing with previous values.

Bear color will be applied when the current closing value is greater than the previous closing value.

Bull color will be applied when the current closing value is less than the previous closing value.

By default, bullFillColor is set as red and bearFillColor is set as green.

#### *Solid Candles*

[enableSolidCandles](#) is used to enable/disable the solid

candles. By default is set to be false. The fill color of the candle will be defined by its opening and closing values.

[bearFillColor](#) will be applied when the opening value is less than the closing value.

[bullFillColor](#) will be applied when the opening value is greater than closing value.

#### **APP.VUE**

```
<template>
  <div id="app">
    <ejs-chart id="container" :title='title'
:primaryXAxis='primaryXAxis' :primaryYAxis='primaryYAxis'>
      <e-series-collection>
        <e-series :dataSource='seriesData' type='Candle' xName='x'
high='high' low='low'
          open='open' close='close' name='SHIRPUR-G' bearFillColor=
'#e56590' bullFillColor= '#f8b883' :enableSolidCandles='enableSolidCandles'>
        </e-series>
      </e-series-collection>
    </ejs-chart>
  </div>
</template>
<script>
import Vue from "vue";
import { ChartPlugin, CandleSeries, Category } from "@syncfusion/ej2-vue-
charts";
Vue.use(ChartPlugin);
export default {
  data() {
    return {
      seriesData: [
        { x: 'Jan', open: 120, high: 160, low: 100, close: 140 },
        { x: 'Feb', open: 150, high: 190, low: 130, close: 170 },
        { x: 'Mar', open: 130, high: 170, low: 110, close: 150 },
        { x: 'Apr', open: 160, high: 180, low: 120, close: 140 },
        { x: 'May', open: 150, high: 170, low: 110, close: 130 }
      ],
      primaryXAxis: {
        title: 'Date',
        valueType: 'Category',
      },
      primaryYAxis: {
        title: 'Price in Dollar', minimum: 100, maximum: 200, interval:
20,
      },
    }
  }
}
```



```

        enableSolidCandles: true,
        title: "Financial Analysis"
    };
},
provide: {
    chart: [CandleSeries, Category]
}
};
</script>
<style>
    #container {
        height: 350px;
    }
</style>

```

{% previewsample "page.domainurl/code-snippet/chart/series/candle-cs2" %}

[See Also](#)

- [Data label](#)
- [Tooltip](#)

## Box and Whisker in Vue Chart Component

### *Box and whisker*

To render a box and whisker chart, use series [type](#) as `BoxAndWhisker` and inject

`BoxAndWhiskerSeries` into the `provide`. The field `y` requires n number of data or it should contains minimum of five values to plot a segment.

### **APP.VUE**

```

<template>
    <div id="app">
        <ejs-chart id="container" :title='title'
        :primaryXAxis='primaryXAxis'>
            <e-series-collection>
                <e-series :dataSource='seriesData' type='BoxAndWhisker'
                xName='x' yName='y' :marker='marker'> </e-series>
            </e-series-collection>
        </ejs-chart>
    </div>
</template>
<script>
import Vue from "vue";
import { ChartPlugin, BoxAndWhiskerSeries, Category } from "@syncfusion/ej2-
vue-charts";
Vue.use(ChartPlugin);
export default {
    data() {
        return {
            seriesData: [
                { x: 'Development', y: [22, 22, 23, 25, 25, 25, 26, 27,
27, 28, 28, 29, 30, 32, 34, 32, 34, 36, 35, 38] },

```

```

    { x: 'Testing', y: [22, 33, 23, 25, 26, 28, 29, 30, 34,
33, 32, 31, 50] },
    { x: 'HR', y: [22, 24, 25, 30, 32, 34, 36, 38, 39, 41,
35, 36, 40, 56] },
    { x: 'Finance', y: [26, 27, 28, 30, 32, 34, 35, 37, 35,
37, 45] },
    { x: 'R&D', y: [26, 27, 29, 32, 34, 35, 36, 37, 38, 39,
41, 43, 58] },
    { x: 'Sales', y: [27, 26, 28, 29, 29, 29, 32, 35, 32,
38, 53] },
    { x: 'Inventory', y: [21, 23, 24, 25, 26, 27, 28, 30,
34, 36, 38] },
    { x: 'Graphics', y: [26, 28, 29, 30, 32, 33, 35, 36, 52]
},
    { x: 'Training', y: [28, 29, 30, 31, 32, 34, 35, 36] }
  ],
  primaryXAxis: {
    valueType: 'Category'
  },
  marker: { visible: true },
  title: "Company Revenue and Profit"
};
},
provide: {
  chart: [BoxAndWhiskerSeries, Category]
}
};
</script>
<style>
  #container {
    height: 350px;
  }
</style>

```

{% previewsample "page.domainurl/code-snippet/chart/series/box-cs1" %}

### Box plot

You can change the rendering mode of the Box and Whisker series using the `boxPlotMode` property. The default `boxPlotMode` is `exclusive`. The other `boxPlotMode` available are `inclusive` and `normal`.

To render a box and whisker chart, use `seriesType` as `BoxAndWhisker` and inject `BoxAndWhiskerSeries` into the `provide`. The field `y` requires n number of data or it should contains minimum of five values to plot a segment.

### APP.VUE

```

<template>
  <div id="app">
    <ejs-chart id="container" :title='title'
:primaryXAxis='primaryXAxis'>
      <e-series-collection>
        <e-series :dataSource='seriesData' type='BoxAndWhisker'
xName='x' yName='y' boxPlotMode='boxPlotMode' :marker='marker'> </e-series>
      </e-series-collection>
    </div>
  </template>

```

```

    </ejs-chart>
  </div>
</template>
<script>
import Vue from "vue";
import { ChartPlugin, BoxAndWhiskerSeries, Category } from "@syncfusion/ej2-vue-charts";
Vue.use(ChartPlugin);
export default {
  data() {
    return {
      seriesData: [
        { x: 'Development', y: [22, 22, 23, 25, 25, 25, 26, 27, 27, 28, 28, 29, 30, 32, 34, 32, 34, 36, 35, 38] },
        { x: 'Testing', y: [22, 33, 23, 25, 26, 28, 29, 30, 34, 33, 32, 31, 50] },
        { x: 'HR', y: [22, 24, 25, 30, 32, 34, 36, 38, 39, 41, 35, 36, 40, 56] },
        { x: 'Finance', y: [26, 27, 28, 30, 32, 34, 35, 37, 35, 37, 45] },
        { x: 'R&D', y: [26, 27, 29, 32, 34, 35, 36, 37, 38, 39, 41, 43, 58] },
        { x: 'Sales', y: [27, 26, 28, 29, 29, 29, 32, 35, 32, 38, 53] },
        { x: 'Inventory', y: [21, 23, 24, 25, 26, 27, 28, 30, 34, 36, 38] },
        { x: 'Graphics', y: [26, 28, 29, 30, 32, 33, 35, 36, 52] },
        { x: 'Training', y: [28, 29, 30, 31, 32, 34, 35, 36] }
      ],
      primaryXAxis: {
        valueType: 'Category'
      },
      boxPlotMode: 'Normal',
      marker: { visible: true },
      title: "Company Revenue and Profit"
    };
  },
  provide: {
    chart: [BoxAndWhiskerSeries, Category]
  }
};
</script>
<style>
  #container {
    height: 350px;
  }
</style>

```

{% previewsample "page.domainurl/code-snippet/chart/series/box-cs2" %}

### Show mean

In Box and Whisker series **showMean** property is used to show the box and whisker average value. The default value of **showMean** is false.

### APP.VUE

```

<template>
  <div id="app">
    <ejs-chart id="container" :title='title'
:primaryXAxis='primaryXAxis'>
      <e-series-collection>
        <e-series :dataSource='seriesData' type='BoxAndWhisker'
xName='x' yName='y' :showMean='showMean' :marker='marker'> </e-series>
      </e-series-collection>
    </ejs-chart>
  </div>
</template>
<script>
import Vue from "vue";
import { ChartPlugin, BoxAndWhiskerSeries, Category } from "@syncfusion/ej2-
vue-charts";
Vue.use(ChartPlugin);
export default {
  data() {
    return {
      seriesData: [
        { x: 'Development', y: [22, 22, 23, 25, 25, 25, 26, 27,
27, 28, 28, 29, 30, 32, 34, 32, 34, 36, 35, 38] },
        { x: 'Testing', y: [22, 33, 23, 25, 26, 28, 29, 30, 34,
33, 32, 31, 50] },
        { x: 'HR', y: [22, 24, 25, 30, 32, 34, 36, 38, 39, 41,
35, 36, 40, 56] },
        { x: 'Finance', y: [26, 27, 28, 30, 32, 34, 35, 37, 35,
37, 45] },
        { x: 'R&D', y: [26, 27, 29, 32, 34, 35, 36, 37, 38, 39,
41, 43, 58] },
        { x: 'Sales', y: [27, 26, 28, 29, 29, 29, 32, 35, 32,
38, 53] },
        { x: 'Inventory', y: [21, 23, 24, 25, 26, 27, 28, 30,
34, 36, 38] },
        { x: 'Graphics', y: [26, 28, 29, 30, 32, 33, 35, 36, 52] },
        { x: 'Training', y: [28, 29, 30, 31, 32, 34, 35, 36] }
      ],
      primaryXAxis: {
        valueType: 'Category'
      },
      showMean: false,
      marker: { visible: true },
      title: "Company Revenue and Profit"
    };
  },
  provide: {
    chart: [BoxAndWhiskerSeries, Category]
  }
};
</script>
<style>
  #container {
    height: 350px;
  }
</style>

```

```
{% previewsample "page.domainurl/code-snippet/chart/series/box-cs3" %}
```

[See Also](#)

- [Data label](#)
- [Tooltip](#)

## Waterfall in Vue Chart Component

### Waterfall

Waterfall chart helps to understand the cumulative effect of the sequentially introduced positive and negative values. To render a Waterfall series, use series [type](#) as `Waterfall` and inject `WaterfallSeries` into the `provide`. [intermediateSumIndexes](#) property of waterfall is used to represent the cumulative sum values.

### APP.VUE

```
<template>
  <div id="app">
    <ejs-chart id="container" :title='title'
:primaryXAxis='primaryXAxis'>
      <e-series-collection>
        <e-series :dataSource='seriesData' type='Waterfall'
xName='x' yName='y' name='Male'
          :intermediateSumIndexes:='intermediate' :sumIndexes='sum'>
        </e-series>
      </e-series-collection>
    </ejs-chart>
  </div>
</template>
<script>
import Vue from "vue";
import { ChartPlugin, WaterfallSeries, Category } from "@syncfusion/ej2-vue-
charts";
Vue.use(ChartPlugin);
export default {
  data() {
    return {
      seriesData: [
        { x: 'Income', y: 4711 }, { x: 'Sales', y: -1015 },
        { x: 'Development', y: -688 },
        { x: 'Revenue', y: 1030 }, { x: 'Balance' },
        { x: 'Administrative', y: -780 },
        { x: 'Expense', y: -361 }, { x: 'Tax', y: -695 },
        { x: 'Net Profit' }
      ],
      primaryXAxis: {
        valueType: 'Category'
      },
      sum: [7],
      intermediate: [4]
      title: "Company Revenue and Profit"
    };
  },
}
```

```

    provide: {
      chart: [WaterfallSeries, Category]
    }
  };
</script>
<style>
  #container {
    height: 350px;
  }
</style>

```

{% previewsample "page.domainurl/code-snippet/chart/series/waterfall-cs1" %}

### Series customization

The negative changes of waterfall charts is represented by using [negativeFillColor](#) and the summary changes are represented by using [summaryFillColor](#) properties.

By default, the negativeFillColor as '#E94649' and the summaryFillColor as '#4E81BC'.

### APP.VUE

```

<template>
  <div id="app">
    <ejs-chart id="container" :title='title'
:primaryXAxis='primaryXAxis'>
      <e-series-collection>
        <e-series :dataSource='seriesData' type='Waterfall'
xName='x' yName='y' name='Male' summaryFillColor='#e56590'
negativeFillColor='#f8b883'
:intermediateSumIndexes:='intermediate' :sumIndexes='sum'>
      </e-series>
    </e-series-collection>
  </div>
</template>
<script>
import Vue from "vue";
import { ChartPlugin, WaterfallSeries, Category } from "@syncfusion/ej2-vue-
charts";
Vue.use(ChartPlugin);
export default {
  data() {
    return {
      seriesData: [
        { x: 'Income', y: 4711 }, { x: 'Sales', y: -1015 },
        { x: 'Development', y: -688 },
        { x: 'Revenue', y: 1030 }, { x: 'Balance' },
        { x: 'Administrative', y: -780 },
        { x: 'Expense', y: -361 }, { x: 'Tax', y: -695 },
        { x: 'Net Profit' }
      ],
      primaryXAxis: {
        valueType: 'Category'
      },
      sum: [7],
      intermediate: [4]
    };
  },
  title: "Company Revenue and Profit"
}

```

```

    };
  },
  provide: {
    chart: [WaterfallSeries, Category]
  }
};
</script>
<style>
  #container {
    height: 350px;
  }
</style>

```

{% previewsample "page.domainurl/code-snippet/chart/series/waterfall-cs2" %}

[See Also](#)

- [Data label](#)
- [Tooltip](#)

## Histogram in Vue Chart Component

### [Histogram Series](#)

Histogram type charts can provide a visual display of large amounts of data that are difficult to understand in a tabular or spreadsheet form. To render histogram chart, use series [type](#) as **Histogram** and inject **HistogramSeries** module using **provide** method.

### **APP.VUE**

```

<template>
  <div id="app">
    <ejs-chart id="container" :legendSettings='legendSettings'
:primaryYAxis='primaryYAxis'
    :title='title' :tooltip='tooltip'>
      <e-series-collection>
        <e-series type='Histogram' width=2 yName='y' name='Score'
:dataSource='dataSource'
          :binInterval='binInterval' :marker='marker'
:showNormalDistribution='showNormalDistribution' :columnWidth='columnWidth'>
        </e-series>
      </e-series-collection>
    </ejs-chart>
  </div>
</template>
<script>
import Vue from "vue";
import { ChartPlugin, ChartTheme, HistogramSeries, DataLabel, Tooltip } from
"@syncfusion/ej2-vue-charts";
Vue.use(ChartPlugin);
let chartData = [];
let points = [5.250, 7.750, 0, 8.275, 9.750, 7.750, 8.275, 6.250, 5.750,
5.250, 23.000, 26.500, 27.750, 25.025, 26.500, 26.500, 28.025,
29.250, 26.750, 27.250,
26.250, 25.250, 34.500, 25.625, 25.500, 26.625, 36.275, 36.250,
26.875, 40.000, 43.000,

```

```

    46.500, 47.750, 45.025, 56.500, 56.500, 58.025, 59.250, 56.750,
57.250,
    46.250, 55.250, 44.500, 45.525, 55.500, 46.625, 46.275, 56.250,
46.875, 43.000,
    46.250, 55.250, 44.500, 45.425, 55.500, 56.625, 46.275, 56.250,
46.875, 43.000,
    46.250, 55.250, 44.500, 45.425, 55.500, 46.625, 56.275, 46.250,
56.875, 41.000, 63.000,
    66.500, 67.750, 65.025, 66.500, 76.500, 78.025, 79.250, 76.750,
77.250,
    66.250, 75.250, 74.500, 65.625, 75.500, 76.625, 76.275, 66.250,
66.875, 80.000, 85.250,
    87.750, 89.000, 88.275, 89.750, 97.750, 98.275, 96.250, 95.750,
95.250
    ];
points.map((value) => {
    chartData.push({
        y: value
    });
});
export default {
    data() {
        return {
            legendSettings: { visible: false },
            primaryYAxis: {
                title: 'Number of Students',
                minimum: 0, maximum: 50, interval: 10,
                majorTickLines: { width: 0 }, lineStyle: { width: 0 }
            },
            marker: { dataLabel: { visible: true, position: 'Top', font: {
fontWeight: '600', color: '#ffffff' } } },
                title: 'Examination Result', tooltip: { enable: true },
                dataSource: chartData,
                tooltip: {enable: true},
                binInterval: 20,
                columnWidth: 0.99,
                showNormalDistribution: true
            };
        },
        provide: {
            chart: [HistogramSeries, DataLabel, Tooltip]
        }
    };
</script>
<style>
    #container {
        height: 350px;
    }
</style>

```

{% previewsample "page.domainurl/code-snippet/chart/series/waterfall-cs3" %}

[See Also](#)

- [Data label](#)



- [Tooltip](#)

## Error Bar in Vue Chart Component

### Error bar

Error bars are graphical representations of the variability of data and used on graphs to indicate the error or uncertainty in a reported measurement. To render the error bar for the series, set [visible](#) as `true` in error bar object and inject `ErrorBar` module into the `provide`.

### APP.VUE

```
<template>
  <div id="app">
    <ejs-chart id="container" :title='title'
:primaryXAxis='primaryXAxis'>
      <e-series-collection>
        <e-series :dataSource='seriesData' type='Line' xName='x'
yName='y' name='India' width=2 :marker='marker' :errorBar='errorBar'> </e-
series>
      </e-series-collection>
    </ejs-chart>
  </div>
</template>
<script>
import Vue from "vue";
import { ChartPlugin, LineSeries, Category, ErrorBar } from
"@syncfusion/ej2-vue-charts";
Vue.use(ChartPlugin);
export default {
  data() {
    return {
      seriesData: [
        { x: 2006, y: 7.8 }, { x: 2007, y: 7.2 },
        { x: 2008, y: 6.8 }, { x: 2009, y: 10.7 },
        { x: 2010, y: 10.8 }, { x: 2011, y: 9.8 }
      ],
      primaryXAxis: {
        minimum: 2005, maximum: 2012, interval: 1,
        title: 'Year'
      },
      errorBar: {
        visible: true
      },
      marker: {
        visible: true
      },
      title: "Olympic Medals"
    };
  },
  provide: {
    chart: [LineSeries, Category, ErrorBar]
  }
};
</script>
<style>
#container {
  height: 350px;
```

```
}
</style>
```

{% previewsample "page.domainurl/code-snippet/chart/series/errorbar-cs1" %}

### Error bar type

To change the error bar rendering type using [type](#) option of error bar. To change the error bar line length you can use [verticalError](#) property.

### APP.VUE

```
<template>
  <div id="app">
    <ejs-chart id="container" :title='title'
:primaryXAxis='primaryXAxis'>
      <e-series-collection>
        <e-series :dataSource='seriesData' type='Line' xName='x'
yName='y' name='India' width=2 :marker='marker' :errorBar='errorBar'> </e-
series>
      </e-series-collection>
    </ejs-chart>
  </div>
</template>
<script>
import Vue from "vue";
import { ChartPlugin, LineSeries, Category, ErrorBar } from
"@syncfusion/ej2-vue-charts";
Vue.use(ChartPlugin);
export default {
  data() {
    return {
      seriesData: [
        { x: 2006, y: 7.8 }, { x: 2007, y: 7.2 },
        { x: 2008, y: 6.8 }, { x: 2009, y: 10.7 },
        { x: 2010, y: 10.8 }, { x: 2011, y: 9.8 }
      ],
      primaryXAxis: {
        minimum: 2005, maximum: 2012, interval: 1,
        title: 'Year'
      },
      errorBar: {
        visible: true, type: 'Percentage', verticalError:4
      },
      marker: {
        visible: true
      },
      title: "Olympic Medals"
    };
  },
  provide: {
    chart: [LineSeries, Category, ErrorBar]
  }
};
</script>
<style>
#container {
```

```

    height: 350px;
  }
</style>

```

{% previewsample "page.domainurl/code-snippet/chart/series/errorbar-cs2" %}

### Error bar type

To customize the error bar type, set error bar [type](#) as **Custom** and then change the horizontal/vertical positive and negative error of error bar.

### APP.VUE

```

<template>
  <div id="app">
    <ejs-chart id="container" :title='title'
    :primaryXAxis='primaryXAxis'>
      <e-series-collection>
        <e-series :dataSource='seriesData' type='Line' xName='x'
        yName='y' name='India' width=2 :marker='marker' :errorBar='errorBar'> </e-
        series>
      </e-series-collection>
    </ejs-chart>
  </div>
</template>
<script>
import Vue from "vue";
import { ChartPlugin, LineSeries, Category, ErrorBar } from
"@syncfusion/ej2-vue-charts";
Vue.use(ChartPlugin);
export default {
  data() {
    return {
      seriesData: [
        { x: 2006, y: 7.8 }, { x: 2007, y: 7.2 },
        { x: 2008, y: 6.8 }, { x: 2009, y: 10.7 },
        { x: 2010, y: 10.8 }, { x: 2011, y: 9.8 }
      ],
      primaryXAxis: {
        minimum: 2005, maximum: 2012, interval: 1,
        title: 'Year'
      },
      errorBar: {
        visible: true, type: 'Custom',
        mode: 'Both'
        verticalPostiveError:3,
        horizontalPositiveError:2,
        verticalNegativeError:3,
        horizontalNegativeError:2
      },
      marker: {
        visible: true
      },
      title: "Olympic Medals"
    };
  },
  provide: {

```

```

    chart: [LineSeries, Category, ErrorBar]
  }
};
</script>
<style>
  #container {
    height: 350px;
  }
</style>

```

{% previewsample "page.domainurl/code-snippet/chart/series/errorbar-cs3" %}

### Error bar mode

Error bar mode is used to define whether the error bar line has to be drawn horizontally, vertically or in both side. To change the error bar mode use [mode](#) option.

### APP.VUE

```

<template>
  <div id="app">
    <ejs-chart id="container" :title='title'
:primaryXAxis='primaryXAxis'>
      <e-series-collection>
        <e-series :dataSource='seriesData' type='Line' xName='x'
yName='y' name='India' width=2 :marker='marker' :errorBar='errorBar'> </e-
series>
      </e-series-collection>
    </ejs-chart>
  </div>
</template>
<script>
import Vue from "vue";
import { ChartPlugin, LineSeries, Category, ErrorBar } from
"@syncfusion/ej2-vue-charts";
Vue.use(ChartPlugin);
export default {
  data() {
    return {
      seriesData: [
        { x: 2006, y: 7.8 }, { x: 2007, y: 7.2 },
        { x: 2008, y: 6.8 }, { x: 2009, y: 10.7 },
        { x: 2010, y: 10.8 }, { x: 2011, y: 9.8 }
      ],
      primaryXAxis: {
        minimum: 2005, maximum: 2012, interval: 1,
        title: 'Year'
      },
      errorBar: {
        visible: true, mode: 'Horizontal'
      },
      marker: {
        visible: true
      },
      title: "Olympic Medals"
    };
  },
},

```

```

    provide: {
      chart: [LineSeries, Category, ErrorBar]
    }
  };
</script>
<style>
  #container {
    height: 350px;
  }
</style>

```

{% previewsample "page.domainurl/code-snippet/chart/series/errorbar-cs4" %}

#### Error bar direction

To change the error bar direction to plus, minus or both side using [direction](#) option.

#### APP.VUE

```

<template>
  <div id="app">
    <ejs-chart id="container" :title='title'
    :primaryXAxis='primaryXAxis'>
      <e-series-collection>
        <e-series :dataSource='seriesData' type='Line' xName='x'
        yName='y' name='India' width=2 :marker='marker' :errorBar='errorBar'> </e-
        series>
      </e-series-collection>
    </ejs-chart>
  </div>
</template>
<script>
import Vue from "vue";
import { ChartPlugin, LineSeries, Category, ErrorBar } from
"@syncfusion/ej2-vue-charts";
Vue.use(ChartPlugin);
export default {
  data() {
    return {
      seriesData: [
        { x: 2006, y: 7.8 }, { x: 2007, y: 7.2 },
        { x: 2008, y: 6.8 }, { x: 2009, y: 10.7 },
        { x: 2010, y: 10.8 }, { x: 2011, y: 9.8 }
      ],
      primaryXAxis: {
        minimum: 2005, maximum: 2012, interval: 1,
        title: 'Year'
      },
      errorBar: {
        visible: true, mode:'Vertical', direction:'Minus'
      },
      marker: {
        visible: true
      },
      title: "Olympic Medals"
    };
  },
},

```

```

    provide: {
      chart: [LineSeries, Category, ErrorBar]
    }
  };
</script>
<style>
  #container {
    height: 350px;
  }
</style>

```

{% previewsample "page.domainurl/code-snippet/chart/series/errorbar-cs5" %}

### *Customizing error bar cap*

To customize the error bar cap length, width and fill color, you can use [errorBarCap](#) option.

### **APP.VUE**

```

<template>
  <div id="app">
    <ejs-chart id="container" :title='title'
:primaryXAxis='primaryXAxis'>
      <e-series-collection>
        <e-series :dataSource='seriesData' type='Line' xName='x'
yName='y' name='India' width=2 :marker='marker' :errorBar='errorBar'> </e-
series>
      </e-series-collection>
    </ejs-chart>
  </div>
</template>
<script>
import Vue from "vue";
import { ChartPlugin, LineSeries, Category, ErrorBar } from
"@syncfusion/ej2-vue-charts";
Vue.use(ChartPlugin);
export default {
  data() {
    return {
      seriesData: [
        { x: 2006, y: 7.8 }, { x: 2007, y: 7.2 },
        { x: 2008, y: 6.8 }, { x: 2009, y: 10.7 },
        { x: 2010, y: 10.8 }, { x: 2011, y: 9.8 }
      ],
      primaryXAxis: {
        minimum: 2005, maximum: 2012, interval: 1,
        title: 'Year'
      },
      errorBar: { visible: true, errorBarCap:{ length:10, width:10,
color:'#0000ff' } },
      marker: {
        visible: true
      },
      title: "Olympic Medals"
    };
  },
  provide: {

```

```

    chart: [LineSeries, Category, ErrorBar]
  }
};
</script>
<style>
  #container {
    height: 350px;
  }
</style>

```

{% previewsample "page.domainurl/code-snippet/chart/series/errorbar-cs6" %}

#### Customizing Error Bar Color

To customize the error bar color for individual errors, use the [errorBarColorMapping](#) property. You can also customize the vertical error, horizontal error, horizontal negative and positive error and vertical negative and positive error for an individual point using [verticalError](#), [horizontalError](#), [horizontalNegativeError](#), [horizontalPositiveError](#), [verticalNegativeError](#) and [verticalPositiveError](#) properties.

#### APP.VUE

```

<template>
  <div id="app">
    <ejs-chart id="container" :title='title'
:primaryXAxis='primaryXAxis'>
      <e-series-collection>
        <e-series :dataSource='seriesData' type='Line' xName='x'
yName='y' name='India' width=2 :marker='marker' :errorBar='errorBar'> </e-
series>
      </e-series-collection>
    </ejs-chart>
  </div>
</template>
<script>
import Vue from "vue";
import { ChartPlugin, LineSeries, Category, ErrorBar } from
"@syncfusion/ej2-vue-charts";
Vue.use(ChartPlugin);
export default {
  data() {
    return {
      seriesData: [
        { x: 2006, y: 7.8, color: 'red', error: 1.5 }, { x: 2007, y:
7.2, color: 'blue', error: 1.6 },
        { x: 2008, y: 6.8, color: 'green', error: 1.9 }, { x: 2009, y:
10.7, color: 'brown', error: .5 },
        { x: 2010, y: 10.8, color: 'orange', error: 1 }, { x: 2011, y:
9.8, color: 'yellow', error: 1.7 }
      ],
      primaryXAxis: {
        minimum: 2005, maximum: 2012, interval: 1,
        title: 'Year'
      },
      errorBar: { visible: true, errorBarColorMapping: 'color',
verticalError: 'error' },
      marker: {

```

```

        visible: true
      },
      title: "Olympic Medals"
    };
  },
  provide: {
    chart: [LineSeries, Category, ErrorBar]
  }
};
</script>
<style>
  #container {
    height: 350px;
  }
</style>

```

{% previewsample "page.domainurl/code-snippet/chart/series/errorbar-cs7" %}

[See Also](#)

- [Data label](#)
- [Tooltip](#)

## Vertical in Vue Chart Component

### Vertical Chart

In EJ2 chart, you can draw a chart in vertical manner by changing orientation of the axis. All series types support this feature. You can use `isTransposed` property in chart to render a chart in vertical manner.

### APP.VUE

```

<template>
  <div id="app">
    <ejs-chart id="container" :title='title'
:primaryXAxis='primaryXAxis' :primaryYAxis='primaryYAxis'
isTransposed='true'>
      <e-series-collection>
        <e-series :dataSource='seriesData' type='Spline' xName='x'
yName='y' name='London'> </e-series>
      </e-series-collection>
    </ejs-chart>
  </div>
</template>
<script>
import Vue from "vue";
import { ChartPlugin, SplineSeries, Category, ScatterSeries } from
"@syncfusion/ej2-vue-charts";
Vue.use(ChartPlugin);
export default {
  data() {
    return {
      seriesData:[
        { x: 'Jan', y: -1 }, { x: 'Feb', y: -1 }, { x: 'Mar', y: 2 },
        { x: 'Apr', y: 8 }, { x: 'May', y: 13 }, { x: 'Jun', y: 18 },
        { x: 'Jul', y: 21 }, { x: 'Aug', y: 20 }, { x: 'Sep', y: 16 },
        { x: 'Oct', y: 10 }, { x: 'Nov', y: 4 }, { x: 'Dec', y: 0 }
      ]
    }
  }
}

```



```

    ],
    primaryXAxis: {
      valueType: 'Category'
    },
    primaryYAxis: {
      minimum: -5, maximum: 35, interval: 10,
      title: 'Temperature in Celsius',
      labelFormat: '{value}C'
    },
    title: "Climate Graph-2012"
  };
},
provide: {
  chart: [SplineSeries, Category, ScatterSeries]
}
};
</script>
<style>
  #container {
    height: 350px;
  }
</style>

```

{% previewsample "page.domainurl/code-snippet/chart/series/line-cs9" %}

[See Also](#)

- [Data label](#)
- [Tooltip](#)

## Pareto in Vue Chart Component

### [Pareto chart](#)

Pareto charts are used to find the cumulative values of data in different categories. It is a combination of Column and Line series. The initial values are represented by column chart and the cumulative values are represented by Line chart. To render a pareto chart, use series [type](#) as **Pareto** and inject **ParetoSeries** **ColumnSeries** and **LineSeries** module.

### APP.VUE

```

<template>
  <div id="app">
    <ejs-chart id="container":title='title'
:primaryXAxis='primaryXAxis' :primaryYAxis='primaryYAxis'>
      <e-series-collection>
        <e-series :dataSource='seriesData' type='Pareto' xName='x'
yName='y' name='Defect' width='2'> </e-series>
      </e-series-collection>
    </ejs-chart>
  </div>
</template>
<script>
import Vue from "vue";

```

```

import {ChartPlugin, LineSeries, StackingColumnSeries, Tooltip,
ColumnSeries, Category, Legend, ParetoSeries} from "@syncfusion/ej2-vue-
charts";
Vue.use(ChartPlugin);
export default {
  data: function() {
    return {
      seriesData: [
        { x: 'Traffic', y: 56 }, { x: 'Child Care', y: 44.8 },
        { x: 'Transport', y: 27.2 }, { x: 'Weather', y: 19.6 },
        { x: 'Emergency', y: 6.6 }
      ],
      primaryXAxis: {
        valueType: 'Category',
        title: 'Defects'
      },
      primaryYAxis: {
        title: 'Frequency',
        minimum: 0,
        maximum: 150,
        interval: 30,
      },
      title: 'Defect vs Frequency',
      tooltip: {
        enable: true,
        shared: true
      }
    };
  },
  provide: {
    chart: [StackingColumnSeries, LineSeries, Category, ColumnSeries,
Legend, Tooltip, ParetoSeries]
  }
};
</script>
<style>
  #container {
    height: 350px;
  }
</style>

```

{% previewsample "page.domainurl/code-snippet/chart/series/line-cs10" %}

#### *Pareto customization*

The pareto line series can be customized by using the [marker](#), [width](#), [dashArray](#), and [fill](#) properties in the [paretoOptions](#). The secondary axis for the pareto series can be shown or hidden using the [showAxis](#) property.

#### **APP.VUE**

```

<template>
  <div id="app">
    <ejs-chart id="container":title='title'
:primaryXAxis='primaryXAxis' :primaryYAxis='primaryYAxis'
:legendSettings='legendSettings' :tooltip='tooltip' :chartArea='chartArea'>

```

```

        <e-series-collection>
            <e-series :dataSource='seriesData' type='Pareto' xName='x'
yName='y' name='Defect' width='2' opacity= 0.75 columnWidth= 0.4
                :paretoOptions='paretoOptions'
:cornerRadius='cornerRadius'> </e-series>
        </e-series-collection>
    </ejs-chart>
</div>
</template>
<script>
import Vue from "vue";
import { Browser } from '@syncfusion/ej2-base';
import { ChartPlugin, LineSeries, StackingColumnSeries, Tooltip,
ColumnSeries, Category, Legend, ParetoSeries, Highlight } from
"@syncfusion/ej2-vue-charts";
Vue.use(ChartPlugin);
export default {
    data: function() {
        return {
            seriesData: [
                { x: 'Button Defect', y: 23 }, { x: 'Pocket Defect', y: 16 },
                { x: 'Collar Defect', y: 10 }, { x: 'Cuff Defect', y: 7 },
                { x: 'Sleeve Defect', y: 6 }, { x: 'Other Defect', y: 2 }
            ],
            primaryXAxis: {
                interval: 1,
                valueType: 'Category',
                majorGridLines: { width: 0 }, minorGridLines: { width: 0 },
                majorTickLines: { width: 0 }, minorTickLines: { width: 0 },
                lineStyle: { width: 0 }, labelIntersectAction: 'Rotate45',
            },
            primaryYAxis:
            {
                title: 'Frequency of Occurrence',
                minimum: 0,
                maximum: 25,
                interval: 5,
                lineStyle: { width: 0 },
                majorTickLines: { width: 0 }, majorGridLines: { width: 1 },
                minorGridLines: { width: 1 }, minorTickLines: { width: 0 }
            },
            chartArea: {
                border: {
                    width: 0
                }
            },
            paretoOptions: {
                marker: { visible: true, isFilled: true, width: 7, height: 7 },
                dashArray: '3,2',
                width: 2,
                fill: '#F7523F'
            },
            cornerRadius: { topLeft: Browser.isDevice? 4 : 6, topRight:
Browser.isDevice ? 4 : 6 },
            title: 'Defects in Shirts',
            legendSettings: {
                visible: true,

```

```

        enableHighlight: true
      },
      tooltip: {
        enable: true,
        shared: true,
        format: '${series.name} : <b>${point.y}</b>'
      },
    };
  },
  provide: {
    chart: [StackingColumnSeries, LineSeries, Category, ColumnSeries,
    Legend, Tooltip, ParetoSeries, Highlight]
  }
};
</script>
<style>
  #container {
    height: 350px;
  }
</style>

```

{% previewsample "page.domainurl/code-snippet/chart/series/line-cs11" %}

[See also](#)

- [Data label](#)
- [Tooltip](#)

## Chart series in Vue Chart component

### Multiple Series

You can add multiple series to the chart by using [series](#) property.

The series are rendered in the order as it is added to the series array.

### APP.VUE

```

<template>
  <div id="app">
    <ejs-chart id="container" :title='title'
    :primaryXAxis='primaryXAxis'>
      <e-series-collection>
        <e-series :dataSource='seriesData' type='Column'
        xName='country' yName='gold' name='Gold'> </e-series>
        <e-series :dataSource='seriesData' type='Column'
        xName='country' yName='silver' name='Silver'> </e-series>
        <e-series :dataSource='seriesData' type='Column'
        xName='country' yName='bronze' name='Bronze'> </e-series>
      </e-series-collection>
    </ejs-chart>
  </div>
</template>
<script>
import Vue from "vue";
import { ChartPlugin, ColumnSeries, Category } from "@syncfusion/ej2-vue-
charts";

```

```

Vue.use(ChartPlugin);
export default {
  data() {
    return {
      seriesData: [
        { country: "USA", gold: 50, silver: 70, bronze: 45 },
        { country: "China", gold: 40, silver: 60, bronze: 55 },
        { country: "Japan", gold: 70, silver: 60, bronze: 50 },
        { country: "Australia", gold: 60, silver: 56, bronze: 40 },
        { country: "France", gold: 50, silver: 45, bronze: 35 },
        { country: "Germany", gold: 40, silver: 30, bronze: 22 },
        { country: "Italy", gold: 40, silver: 35, bronze: 37 },
        { country: "Sweden", gold: 30, silver: 25, bronze: 27 }
      ],
      primaryXAxis: {
        valueType: 'Category',
        title: 'Countries'
      },
      animation: { enable: false },
      title: "Olympic Medals"
    };
  },
  provide: {
    chart: [ColumnSeries, Category]
  }
};
</script>
<style>
#container {
  height: 350px;
}
</style>

```

{% previewsample "page.domainurl/code-snippet/chart/axis/category-cs27" %}

### Combination Series

Combination of different types like Line, column etc, can be rendered in a chart.

Bar series cannot be combined with any other series as the axis orientation is different from other series.

### APP.VUE

```

<template>
  <div id="app">
    <ejs-chart id="container" :title='title'
    :primaryXAxis='primaryXAxis'>
      <e-series-collection>
        <e-series :dataSource='seriesData' type='StackingColumn'
        xName='x' yName='y' name='USA'> </e-series>
        <e-series :dataSource='seriesData' type='StackingColumn'
        xName='x' yName='y1' name='UK'> </e-series>
        <e-series :dataSource='seriesData' type='StackingColumn'
        xName='x' yName='y2' name='Canada'> </e-series>
        <e-series :dataSource='seriesData' type='StackingColumn'
        xName='x' yName='y3' name='China'> </e-series>
      </e-series-collection>
    </ejs-chart>
  </div>
</template>

```

```

        <e-series :dataSource='seriesData' type='Line' xName='x'
yName='y4' name='GDP' :marker='marker' opacity=0.6> </e-series>
    </e-series-collection>
</ejs-chart>
</div>
</template>
<script>
import Vue from "vue";
import { ChartPlugin, StackingColumnSeries, LineSeries, Category } from
"@syncfusion/ej2-vue-charts";
Vue.use(ChartPlugin);
export default {
    data() {
        return {
            seriesData:[
                { x: '2005', y: 1.2, y1: 0.5, y2: 0.7, y3: -0.8, y4: 1.5 },
                { x: '2006', y: 1, y1: 0.5, y2: 1.4, y3: 0, y4: 2.3 },
                { x: '2007', y: 1, y1: 0.5, y2: 1.5, y3: -1, y4: 2 },
                { x: '2008', y: 0.25, y1: 0.35, y2: 0.35, y3: -.35, y4: 0.1 },
                { x: '2009', y: 0.1, y1: 0.9, y2: -2.7, y3: -0.3, y4: -2.7 },
                { x: '2010', y: 1, y1: 0.5, y2: 0.5, y3: -0.5, y4: 1.8 },
                { x: '2011', y: 0.1, y1: 0.25, y2: 0.25, y3: 0, y4: 2 },
                { x: '2012', y: -0.25, y1: -0.5, y2: -0.1, y3: -0.4, y4: 0.4
            },
                { x: '2013', y: 0.25, y1: 0.5, y2: -0.3, y3: 0, y4: 0.9 },
                { x: '2014', y: 0.6, y1: 0.6, y2: -0.6, y3: -0.6, y4: 0.4 },
                { x: '2015', y: 0.9, y1: 0.5, y2: 0, y3: -0.3, y4: 1.3 }
            ],
            primaryXAxis: {
                title: 'Years',
                interval: 1,
                labelIntersectAction : 'Rotate45',
                valueType: 'Category'
            },
            marker: { visible: true, width: 10, opacity: 0.6, height: 10 },
            title: "Annual Growth GDP in France"
        };
    },
    provide: {
        chart: [StackingColumnSeries, LineSeries, Category]
    },
};
</script>
<style>
    #container {
        height: 350px;
    }
</style>

```

{% previewsample "page.domainurl/code-snippet/chart/series/combination-cs1" %}

### Enable Complex Property in Series

By setting `enableComplexProperty` value as `true` in series you can bind complex data to the chart.

### APP.VUE

```

<template>
  <div id="app">
    <ejs-chart id="container" :title='title'
:primaryXAxis='primaryXAxis'>
      <e-series-collection>
        <e-series :dataSource='seriesData' type='Column'
xName='group.x' yName='group.y' name='USA' enableComplexProperty="true">
      </e-series>
        <e-series :dataSource='seriesData' type='Column'
xName='group.x' yName='y' name='UK' enableComplexProperty="true"> </e-
series>
      </e-series-collection>
    </ejs-chart>
  </div>
</template>
<script>
import Vue from "vue";
import { ChartPlugin, ColumnSeries, LineSeries, Category } from
"@syncfusion/ej2-vue-charts";
Vue.use(ChartPlugin);
export default {
  data() {
    return {
      seriesData:[
        {group: { x: 'Aaa', y: 10}, y: 20},
        {group: { x: 'Baa', y: 30}, y: 10}
      ],
      primaryXAxis: {
        valueType: 'Category'
      },
      marker: { visible: true, width: 10, opacity: 0.6, height: 10 },
      title: "Annual Growth GDP in France"
    };
  },
  provide: {
    chart: [ColumnSeries, LineSeries, Category]
  },
};
</script>
<style>
  #container {
    height: 350px;
  }
</style>

```

{% previewsample "page.domainurl/code-snippet/chart/series/combination-cs2" %}

### Technical indicators in Vue Chart component

A technical indicator is a mathematical calculation based on historic price, volume or open interest information that aims to forecast financial market direction.

Chart supports 10 types of technical indicators.

#### Accumulation Distribution

Accumulation Distribution combines price and volume to show how money may be flowing into or out of stock.

To render an Accumulation Distribution Indicator, use indicator `type` as `AccumulationDistribution` and inject `AccumulationDistributionIndicator` into the `provide`.

To calculate the signal line `volume` field is additionally added with `dataSource`.

### APP.VUE

```
<template>
  <div id="app">
    <ejs-chart id="container" :title='title'
:primaryXAxis='primaryXAxis' :primaryYAxis='primaryYAxis'
      :tooltip='tooltip' :chartArea='chartArea' :crosshair='crosshair'
:indicators='indicators' :legendSettings='legendSettings'
:zoomSettings='zoomSettings'
      :rows='rows' :axes='axes' :axisLabelRender='axisLabelRender'>
      <e-series-collection>
        <e-series :dataSource='seriesData1' :animation='animation'
type='Candle' xName='x' yName='y' name='Apple Inc' width=2 low='low'
high='high' close='close' open='open' volume='volume'
bearFillColor='#2ecd71' bullFillColor='#e74c3d' > </e-series>
      </e-series-collection>
    </ejs-chart>
  </div>
</template>
<script>
import Vue from "vue";
import { ChartPlugin, CandleSeries, Category, Tooltip, DateTime, Zoom,
Logarithmic, Crosshair, LineSeries, AccumulationDistributionIndicator,
StripLine } from "@syncfusion/ej2-vue-charts";
Vue.use(ChartPlugin);
let series1: Object[] = [
  {x: new Date('2012-10-15'), open: 90.3357, high: 93.2557, low:
87.0885,close: 87.12,volume: 646996264},
  {x: new Date('2012-10-22'), open: 87.4885, high: 90.7685, low:
84.4285,close: 86.2857,volume: 866040680 },
  {x: new Date('2012-10-29'), open: 84.9828, high: 86.1428, low:
82.1071,close: 82.4,volume: 367371310},
  {x: new Date('2012-11-05'), open: 83.3593, high: 84.3914, low:
76.2457,close: 78.1514,volume: 919719846},
  {x: new Date('2012-11-12'), open: 79.1643, high: 79.2143, low:
72.25,close: 75.3825,volume: 894382149},
  {x: new Date('2012-11-19'), open: 77.2443, high: 81.7143, low:
77.1257,close: 81.6428,volume: 527416747},
  {x: new Date('2012-11-26'), open: 82.2714, high: 84.8928, low:
81.7514,close: 83.6114,volume: 646467974},
  {x: new Date('2012-12-03'), open: 84.8071, high: 84.9414, low:
74.09,close: 76.1785,volume: 980096264},
  {x: new Date('2012-12-10'), open: 75, high: 78.5085, low: 72.2257,close:
72.8277,volume: 835016110},
  {x: new Date('2012-12-17'), open: 72.7043, high: 76.4143, low:
71.6043,close: 74.19,volume: 726150329},
  {x: new Date('2012-12-24'), open: 74.3357, high: 74.8928, low:
72.0943,close: 72.7984,volume: 321104733},
  {x: new Date('2012-12-31'), open: 72.9328, high: 79.2857, low:
72.7143,close: 75.2857,volume: 540854882},
  {x: new Date('2013-01-07'), open: 74.5714, high: 75.9843, low:
73.6,close: 74.3285,volume: 574594262},
```



```

    {x: new Date('2013-01-14'), open: 71.8114, high: 72.9643, low:
69.0543,close: 71.4285,volume: 803105621},
    {x: new Date('2013-01-21'), open: 72.08, high: 73.57, low:
62.1428,close: 62.84,volume: 971912560},
    {x: new Date('2013-01-28'), open: 62.5464, high: 66.0857, low:
62.2657,close: 64.8028,volume: 656549587},
    {x: new Date('2013-02-04'), open: 64.8443, high: 68.4014, low:
63.1428,close: 67.8543,volume: 743778993},
    {x: new Date('2013-02-11'), open: 68.0714, high: 69.2771, low:
65.7028,close: 65.7371,volume: 585292366},
    {x: new Date('2013-02-18'), open: 65.8714, high: 66.1043, low:
63.26,close: 64.4014,volume: 421766997},
    {x: new Date('2013-02-25'), open: 64.8357, high: 65.0171, low:
61.4257,close: 61.4957,volume: 582741215},
    {x: new Date('2013-03-04'), open: 61.1143, high: 62.2043, low:
59.8571,close: 61.6743,volume: 632856539},
    {x: new Date('2013-03-11'), open: 61.3928, high: 63.4614, low:
60.7343,close: 63.38,volume: 572066981},
    {x: new Date('2013-03-18'), open: 63.0643, high: 66.0143, low:
63.0286,close: 65.9871,volume: 552156035},
    {x: new Date('2013-03-25'), open: 66.3843, high: 67.1357, low:
63.0886,close: 63.2371,volume: 390762517},
    {x: new Date('2013-04-01'), open: 63.1286, high: 63.3854, low:
59.9543,close: 60.4571,volume: 505273732},
    {x: new Date('2013-04-08'), open: 60.6928, high: 62.57, low:
60.3557,close: 61.4,volume: 387323550},
    {x: new Date('2013-04-15'), open: 61, high: 61.1271, low: 55.0143,close:
55.79,volume: 709945604},
    {x: new Date('2013-04-22'), open: 56.0914, high: 59.8241, low:
55.8964,close: 59.6007,volume: 787007506},
    {x: new Date('2013-04-29'), open: 60.0643, high: 64.7471, low: 60,close:
64.2828,volume: 655020017},
    {x: new Date('2013-05-06'), open: 65.1014, high: 66.5357, low:
64.3543,close: 64.71,volume: 545488533},
    {x: new Date('2013-05-13'), open: 64.5014, high: 65.4143, low:
59.8428,close: 61.8943,volume: 633706550},
    {x: new Date('2013-05-20'), open: 61.7014, high: 64.05, low:
61.4428,close: 63.5928,volume: 494379068},
    {x: new Date('2013-05-27'), open: 64.2714, high: 65.3, low:
62.7714,close: 64.2478,volume: 362907830},
    {x: new Date('2013-06-03'), open: 64.39, high: 64.9186, low:
61.8243,close: 63.1158,volume: 443249793},
    {x: new Date('2013-06-10'), open: 63.5328, high: 64.1541, low:
61.2143,close: 61.4357,volume: 389680092},
    {x: new Date('2013-06-17'), open: 61.6343, high: 62.2428, low:
58.3,close: 59.0714,volume: 400384818},
    {x: new Date('2013-06-24'), open: 58.2, high: 58.38, low: 55.5528,close:
56.6471,volume: 519314826},
    {x: new Date('2013-07-01'), open: 57.5271, high: 60.47, low:
57.3171,close: 59.6314,volume: 343878841},
    {x: new Date('2013-07-08'), open: 60.0157, high: 61.3986, low:
58.6257,close: 60.93,volume: 384106977},
    {x: new Date('2013-07-15'), open: 60.7157, high: 62.1243, low:
60.5957,close: 60.7071,volume: 286035513},
    {x: new Date('2013-07-22'), open: 61.3514, high: 63.5128, low:
59.8157,close: 62.9986,volume: 395816827},

```

```

    {x: new Date('2013-07-29'), open: 62.9714, high: 66.1214, low:
62.8857,close: 66.0771,volume: 339668858},
    {x: new Date('2013-08-12'), open: 65.2657, high: 72.0357, low:
65.2328,close: 71.7614,volume: 711563584},
    {x: new Date('2013-08-19'), open: 72.0485, high: 73.3914, low:
71.1714,close: 71.5743,volume: 417119660},
    {x: new Date('2013-08-26'), open: 71.5357, high: 72.8857, low:
69.4286,close: 69.6023,volume: 392805888},
    {x: new Date('2013-09-02'), open: 70.4428, high: 71.7485, low:
69.6214,close: 71.1743,volume: 317244380},
    {x: new Date('2013-09-09'), open: 72.1428, high: 72.56, low:
66.3857,close: 66.4143,volume: 669376320},
    {x: new Date('2013-09-16'), open: 65.8571, high: 68.3643, low:
63.8886,close: 66.7728,volume: 625142677},
    {x: new Date('2013-09-23'), open: 70.8714, high: 70.9871, low:
68.6743,close: 68.9643,volume: 475274537},
    {x: new Date('2013-09-30'), open: 68.1786, high: 70.3357, low:
67.773,close: 69.0043,volume: 368198906},
    {x: new Date('2013-10-07'), open: 69.5086, high: 70.5486, low:
68.3257,close: 70.4017,volume: 361437661},
    {x: new Date('2013-10-14'), open: 69.9757, high: 72.7514, low:
69.9071,close: 72.6985,volume: 342694379},
    {x: new Date('2013-10-21'), open: 73.11, high: 76.1757, low:
72.5757,close: 75.1368,volume: 490458997},
    {x: new Date('2013-10-28'), open: 75.5771, high: 77.0357, low:
73.5057,close: 74.29,volume: 508130174},
    {x: new Date('2013-11-04'), open: 74.4428, high: 75.555, low:
73.1971,close: 74.3657,volume: 318132218},
    {x: new Date('2013-11-11'), open: 74.2843, high: 75.6114, low:
73.4871,close: 74.9987,volume: 306711021},
    {x: new Date('2013-11-18'), open: 74.9985, high: 75.3128, low:
73.3814,close: 74.2571,volume: 282778778},
  ];
export default {
  data() {
    return {
      seriesData1: series1,
      primaryXAxis: {
        valueType: 'DateTime',
        majorGridLines: { width: 0 },
        zoomFactor: 0.2, zoomPosition: 0.6,
        crosshairTooltip: { enable: true }
      },
      //Initializing Primary Y Axis
      primaryYAxis: {
        title: 'Price',
        labelFormat: '${value}',
        minimum: 50, maximum: 170,
        plotOffset: 25,
        interval: 30, rowIndex: 1, opposedPosition: true, lineStyle: {
width: 0 }
      },
      axes: [{
        name: 'secondary',
        opposedPosition: true, rowIndex: 0,
        majorGridLines: { width: 0 }, lineStyle: { width: 0 }, minimum:
-7000000000, maximum: 5000000000,

```

```

        interval: 6000000000, majorTickLines: { width: 0 }, title:
'Accumulation Distribution',
        stripLines: [
            {
                start: -7000000000, end: 6000000000, text: '', color:
'#6063ff', visible: true,
                opacity: 0.1, zIndex: 'Behind'
            }
        ],
        rows: [
            {
                height: '40%'
            }, {
                height: '60%'
            }
        ],
        animation: { enable: true },
        indicators: [{
            type: 'AccumulationDistribution', field: 'Close', seriesName:
'Apple Inc', yAxisName: 'secondary', fill: '#6063ff',
            period: 3, animation: { enable: true }
        }],
        zoomSettings:
        {
            enableSelectionZooming: true,
            mode: 'X',
            enablePan : true
        },
        legendSettings: {
            visible: false
        },
        crosshair: { enable: true, lineType: 'Vertical' },
        chartArea: { border: { width: 0 } },
        tooltip: {
            enable: true,
            shared: true
        },
        title: "AAPL 2012-2017"
    };
},
provide: {
    chart: [CandleSeries, Category, LineSeries, Tooltip, DateTime,
StripLine, Zoom, Logarithmic, Crosshair, AccumulationDistributionIndicator]
},
methods: {
    axisLabelRender: function(args) {
        if (args.axis.name === 'secondary') {
            let value = Number(args.text) / 1000000000;
            args.text = String(value) + 'bn';
        }
    }
}
};
</script>
<style>
#container {
    height: 350px;

```

```
}
</style>
```

```
{% previewsample "page.domainurl/code-snippet/chart/technical-indicators/ad-cs1" %}
```

### Average True Range (ATR)

ATR measures the stock volatility by comparing the current value with the previous value. To render a Average True Range (ATR) Indicator, use indicator [type](#) as `Atr` and inject `AtrIndicator` into the `provide`.

### APP.VUE

```
<template>
  <div id="app">
    <ejs-chart id="container" :title='title'
:primaryXAxis='primaryXAxis' :primaryYAxis='primaryYAxis'
      :chartArea='chartArea' :legendSettings='legendSettings'
:indicators='indicators' :crosshair='crosshair' :tooltip='tooltip'
:zoomSettings='zoomSettings'
      :axes='axes' :rows='rows'>
      <e-series-collection>
        <e-series :dataSource='seriesData1' type='Candle' xName='x'
yName='y' name='Apple Inc' width=2 low='low' high='high' close='close'
open='open' volume='volume' bearFillColor='#2ecd71' bullFillColor='#e74c3d'
:animation='animation'> </e-series>
      </e-series-collection>
    </ejs-chart>
  </div>
</template>
<script>
import Vue from "vue";
import { ChartPlugin, Category, CandleSeries, Tooltip, DateTime, Zoom,
Crosshair, LineSeries, Logarithmic, StripLine, AtrIndicator } from
"@syncfusion/ej2-vue-charts";
Vue.use(ChartPlugin);
let series1: Object[] = [
  {x: new Date('2012-10-15'), open: 90.3357, high: 93.2557, low:
87.0885,close: 87.12,volume: 646996264},
  {x: new Date('2012-10-22'), open: 87.4885, high: 90.7685, low:
84.4285,close: 86.2857,volume: 866040680 },
  {x: new Date('2012-10-29'), open: 84.9828, high: 86.1428, low:
82.1071,close: 82.4,volume: 367371310},
  {x: new Date('2012-11-05'), open: 83.3593, high: 84.3914, low:
76.2457,close: 78.1514,volume: 919719846},
  {x: new Date('2012-11-12'), open: 79.1643, high: 79.2143, low:
72.25,close: 75.3825,volume: 894382149},
  {x: new Date('2012-11-19'), open: 77.2443, high: 81.7143, low:
77.1257,close: 81.6428,volume: 527416747},
  {x: new Date('2012-11-26'), open: 82.2714, high: 84.8928, low:
81.7514,close: 83.6114,volume: 646467974},
  {x: new Date('2012-12-03'), open: 84.8071, high: 84.9414, low:
74.09,close: 76.1785,volume: 980096264},
  {x: new Date('2012-12-10'), open: 75, high: 78.5085, low: 72.2257,close:
72.8277,volume: 835016110},
  {x: new Date('2012-12-17'), open: 72.7043, high: 76.4143, low:
71.6043,close: 74.19,volume: 726150329},
```

```

    {x: new Date('2012-12-24'), open: 74.3357, high: 74.8928, low:
72.0943,close: 72.7984,volume: 321104733},
    {x: new Date('2012-12-31'), open: 72.9328, high: 79.2857, low:
72.7143,close: 75.2857,volume: 540854882},
    {x: new Date('2013-01-07'), open: 74.5714, high: 75.9843, low:
73.6,close: 74.3285,volume: 574594262},
    {x: new Date('2013-01-14'), open: 71.8114, high: 72.9643, low:
69.0543,close: 71.4285,volume: 803105621},
    {x: new Date('2013-01-21'), open: 72.08, high: 73.57, low:
62.1428,close: 62.84,volume: 971912560},
    {x: new Date('2013-01-28'), open: 62.5464, high: 66.0857, low:
62.2657,close: 64.8028,volume: 656549587},
    {x: new Date('2013-02-04'), open: 64.8443, high: 68.4014, low:
63.1428,close: 67.8543,volume: 743778993},
    {x: new Date('2013-02-11'), open: 68.0714, high: 69.2771, low:
65.7028,close: 65.7371,volume: 585292366},
    {x: new Date('2013-02-18'), open: 65.8714, high: 66.1043, low:
63.26,close: 64.4014,volume: 421766997},
    {x: new Date('2013-02-25'), open: 64.8357, high: 65.0171, low:
61.4257,close: 61.4957,volume: 582741215},
    {x: new Date('2013-03-04'), open: 61.1143, high: 62.2043, low:
59.8571,close: 61.6743,volume: 632856539},
    {x: new Date('2013-03-11'), open: 61.3928, high: 63.4614, low:
60.7343,close: 63.38,volume: 572066981},
    {x: new Date('2013-03-18'), open: 63.0643, high: 66.0143, low:
63.0286,close: 65.9871,volume: 552156035},
    {x: new Date('2013-03-25'), open: 66.3843, high: 67.1357, low:
63.0886,close: 63.2371,volume: 390762517},
    {x: new Date('2013-04-01'), open: 63.1286, high: 63.3854, low:
59.9543,close: 60.4571,volume: 505273732},
    {x: new Date('2013-04-08'), open: 60.6928, high: 62.57, low:
60.3557,close: 61.4,volume: 387323550},
    {x: new Date('2013-04-15'), open: 61, high: 61.1271, low: 55.0143,close:
55.79,volume: 709945604},
    {x: new Date('2013-04-22'), open: 56.0914, high: 59.8241, low:
55.8964,close: 59.6007,volume: 787007506},
    {x: new Date('2013-04-29'), open: 60.0643, high: 64.7471, low: 60,close:
64.2828,volume: 655020017},
    {x: new Date('2013-05-06'), open: 65.1014, high: 66.5357, low:
64.3543,close: 64.71,volume: 545488533},
    {x: new Date('2013-05-13'), open: 64.5014, high: 65.4143, low:
59.8428,close: 61.8943,volume: 633706550},
    {x: new Date('2013-05-20'), open: 61.7014, high: 64.05, low:
61.4428,close: 63.5928,volume: 494379068},
    {x: new Date('2013-05-27'), open: 64.2714, high: 65.3, low:
62.7714,close: 64.2478,volume: 362907830},
    {x: new Date('2013-06-03'), open: 64.39, high: 64.9186, low:
61.8243,close: 63.1158,volume: 443249793},
    {x: new Date('2013-06-10'), open: 63.5328, high: 64.1541, low:
61.2143,close: 61.4357,volume: 389680092},
    {x: new Date('2013-06-17'), open: 61.6343, high: 62.2428, low:
58.3,close: 59.0714,volume: 400384818},
    {x: new Date('2013-06-24'), open: 58.2, high: 58.38, low: 55.5528,close:
56.6471,volume: 519314826},
    {x: new Date('2013-07-01'), open: 57.5271, high: 60.47, low:
57.3171,close: 59.6314,volume: 343878841},

```

```

    {x: new Date('2013-07-08'), open: 60.0157, high: 61.3986, low:
58.6257,close: 60.93,volume: 384106977},
    {x: new Date('2013-07-15'), open: 60.7157, high: 62.1243, low:
60.5957,close: 60.7071,volume: 286035513},
    {x: new Date('2013-07-22'), open: 61.3514, high: 63.5128, low:
59.8157,close: 62.9986,volume: 395816827},
    {x: new Date('2013-07-29'), open: 62.9714, high: 66.1214, low:
62.8857,close: 66.0771,volume: 339668858},
    {x: new Date('2013-08-12'), open: 65.2657, high: 72.0357, low:
65.2328,close: 71.7614,volume: 711563584},
    {x: new Date('2013-08-19'), open: 72.0485, high: 73.3914, low:
71.1714,close: 71.5743,volume: 417119660},
    {x: new Date('2013-08-26'), open: 71.5357, high: 72.8857, low:
69.4286,close: 69.6023,volume: 392805888},
    {x: new Date('2013-09-02'), open: 70.4428, high: 71.7485, low:
69.6214,close: 71.1743,volume: 317244380},
    {x: new Date('2013-09-09'), open: 72.1428, high: 72.56, low:
66.3857,close: 66.4143,volume: 669376320},
    {x: new Date('2013-09-16'), open: 65.8571, high: 68.3643, low:
63.8886,close: 66.7728,volume: 625142677},
    {x: new Date('2013-09-23'), open: 70.8714, high: 70.9871, low:
68.6743,close: 68.9643,volume: 475274537},
    {x: new Date('2013-09-30'), open: 68.1786, high: 70.3357, low:
67.773,close: 69.0043,volume: 368198906},
    {x: new Date('2013-10-07'), open: 69.5086, high: 70.5486, low:
68.3257,close: 70.4017,volume: 361437661},
    {x: new Date('2013-10-14'), open: 69.9757, high: 72.7514, low:
69.9071,close: 72.6985,volume: 342694379},
    {x: new Date('2013-10-21'), open: 73.11, high: 76.1757, low:
72.5757,close: 75.1368,volume: 490458997},
    {x: new Date('2013-10-28'), open: 75.5771, high: 77.0357, low:
73.5057,close: 74.29,volume: 508130174},
    {x: new Date('2013-11-04'), open: 74.4428, high: 75.555, low:
73.1971,close: 74.3657,volume: 318132218},
    {x: new Date('2013-11-11'), open: 74.2843, high: 75.6114, low:
73.4871,close: 74.9987,volume: 306711021},
    {x: new Date('2013-11-18'), open: 74.9985, high: 75.3128, low:
73.3814,close: 74.2571,volume: 282778778},
];
export default {
  data() {
    return {
      seriesData1: series1,
      primaryXAxis: {
        valueType: 'DateTime',
        majorGridLines: { width: 0 },
        zoomFactor: 0.2, zoomPosition: 0.6,
        crosshairTooltip: { enable: true },
      },
      //Initializing Primary Y Axis
      primaryYAxis: {
        title: 'Price',
        labelFormat: '${value}',
        minimum: 50, maximum: 170,
        interval: 30, rowIndex: 1,
        plotOffset: 25,
      },
    };
  }
};

```

```

        majorGridLines: { width: 1 }, opposedPosition: true, lineStyle:
{ width: 0 }
    },
    axes: [{
        name: 'secondary',
        opposedPosition: true, rowIndex: 0,
        majorGridLines: { width: 0 }, lineStyle: { width: 0 },
majorTickLines: { width: 0 },
        maximum: 14, minimum: 0, interval: 7, title: 'ATR',
        stripLines: [
            {
                start: 0, end: 14, text: '', color: '#6063ff', visible:
true,
                opacity: 0.1, zIndex: 'Behind'
            }
        ]
    }],
    rows: [
        {
            height: '40%'
        }, {
            height: '60%'
        }
    ],
    indicators: [{
        type: 'Atr', field: 'Close', seriesName: 'Apple Inc', yAxisName:
'secondary', fill: '#6063ff',
        period: 3, animation: { enable: true }
    }],
    tooltip: {
        enable: true, shared: true
    },
    animation: { enable: true },
    crosshair: { enable: true, lineType: 'Vertical' },
    chartArea: { border: { width: 0 } },
    zoomSettings:
    {
        enableSelectionZooming: true,
        mode: 'X',
        enablePan : true
    },
    legendSettings: { visible: false },
    title: "AAPL 2012-2017"
};
    },
    provide: {
        chart: [CandleSeries, Category, Tooltip, DateTime, Zoom, Logarithmic,
Crosshair, LineSeries, AtrIndicator, StripLine]
    }
};
</script>
<style>
    #container {
        height: 350px;
    }
</style>

```

```
{% previewsample "page.domainurl/code-snippet/chart/technical-indicators/atr-cs1" %}
```

### Bollinger Band

A chart overlay that shows the upper and lower limits of normal price movements based on the standard deviation of prices. To render a Bollinger Band, use indicator [type](#) as `BollingerBand` and inject `BollingerBands` into the `provide`. Bollinger band will be represented by three lines (upperLine, lowerLine, signalLine). Bollinger Band default values of the [period](#) is 14 and [standardDeviations](#) is 2.

### APP.VUE

```
<template>
  <div id="app">
    <ejs-chart id="container" :title='title'
      :primaryXAxis='primaryXAxis' :primaryYAxis='primaryYAxis'
      :tooltip='tooltip' :chartArea='chartArea' :crosshair='crosshair'
      :indicators='indicators' :legendSettings='legendSettings'
      :zoomSettings='zoomSettings'>
      <e-series-collection>
        <e-series :dataSource='seriesData1' width=2 xName='x'
          yName='y' low='low' high='high' close='close' open='open' volume='volume'
          name='Apple Inc' bearFillColor='#2ecd71' bullFillColor='#e74c3d'
          type='Candle' :animation='animation'> </e-series>
      </e-series-collection>
    </ejs-chart>
  </div>
</template>
<script>
import Vue from "vue";
import { ChartPlugin, CandleSeries, Category, Tooltip, DateTime, Zoom,
  Logarithmic, Crosshair, LineSeries, BollingerBands, RangeAreaSeries } from
"@syncfusion/ej2-vue-charts";
Vue.use(ChartPlugin);
let series1: Object[] = [
  {x: new Date('2012-10-15'), open: 90.3357, high: 93.2557, low:
87.0885, close: 87.12, volume: 646996264},
  {x: new Date('2012-10-22'), open: 87.4885, high: 90.7685, low:
84.4285, close: 86.2857, volume: 866040680 },
  {x: new Date('2012-10-29'), open: 84.9828, high: 86.1428, low:
82.1071, close: 82.4, volume: 367371310},
  {x: new Date('2012-11-05'), open: 83.3593, high: 84.3914, low:
76.2457, close: 78.1514, volume: 919719846},
  {x: new Date('2012-11-12'), open: 79.1643, high: 79.2143, low:
72.25, close: 75.3825, volume: 894382149},
  {x: new Date('2012-11-19'), open: 77.2443, high: 81.7143, low:
77.1257, close: 81.6428, volume: 527416747},
  {x: new Date('2012-11-26'), open: 82.2714, high: 84.8928, low:
81.7514, close: 83.6114, volume: 646467974},
  {x: new Date('2012-12-03'), open: 84.8071, high: 84.9414, low:
74.09, close: 76.1785, volume: 980096264},
  {x: new Date('2012-12-10'), open: 75, high: 78.5085, low: 72.2257, close:
72.8277, volume: 835016110},
  {x: new Date('2012-12-17'), open: 72.7043, high: 76.4143, low:
71.6043, close: 74.19, volume: 726150329},
  {x: new Date('2012-12-24'), open: 74.3357, high: 74.8928, low:
72.0943, close: 72.7984, volume: 321104733},
```



```
{x: new Date('2012-12-31'), open: 72.9328, high: 79.2857, low: 72.7143, close: 75.2857, volume: 540854882},
{x: new Date('2013-01-07'), open: 74.5714, high: 75.9843, low: 73.6, close: 74.3285, volume: 574594262},
{x: new Date('2013-01-14'), open: 71.8114, high: 72.9643, low: 69.0543, close: 71.4285, volume: 803105621},
{x: new Date('2013-01-21'), open: 72.08, high: 73.57, low: 62.1428, close: 62.84, volume: 971912560},
{x: new Date('2013-01-28'), open: 62.5464, high: 66.0857, low: 62.2657, close: 64.8028, volume: 656549587},
{x: new Date('2013-02-04'), open: 64.8443, high: 68.4014, low: 63.1428, close: 67.8543, volume: 743778993},
{x: new Date('2013-02-11'), open: 68.0714, high: 69.2771, low: 65.7028, close: 65.7371, volume: 585292366},
{x: new Date('2013-02-18'), open: 65.8714, high: 66.1043, low: 63.26, close: 64.4014, volume: 421766997},
{x: new Date('2013-02-25'), open: 64.8357, high: 65.0171, low: 61.4257, close: 61.4957, volume: 582741215},
{x: new Date('2013-03-04'), open: 61.1143, high: 62.2043, low: 59.8571, close: 61.6743, volume: 632856539},
{x: new Date('2013-03-11'), open: 61.3928, high: 63.4614, low: 60.7343, close: 63.38, volume: 572066981},
{x: new Date('2013-03-18'), open: 63.0643, high: 66.0143, low: 63.0286, close: 65.9871, volume: 552156035},
{x: new Date('2013-03-25'), open: 66.3843, high: 67.1357, low: 63.0886, close: 63.2371, volume: 390762517},
{x: new Date('2013-04-01'), open: 63.1286, high: 63.3854, low: 59.9543, close: 60.4571, volume: 505273732},
{x: new Date('2013-04-08'), open: 60.6928, high: 62.57, low: 60.3557, close: 61.4, volume: 387323550},
{x: new Date('2013-04-15'), open: 61, high: 61.1271, low: 55.0143, close: 55.79, volume: 709945604},
{x: new Date('2013-04-22'), open: 56.0914, high: 59.8241, low: 55.8964, close: 59.6007, volume: 787007506},
{x: new Date('2013-04-29'), open: 60.0643, high: 64.7471, low: 60, close: 64.2828, volume: 655020017},
{x: new Date('2013-05-06'), open: 65.1014, high: 66.5357, low: 64.3543, close: 64.71, volume: 545488533},
{x: new Date('2013-05-13'), open: 64.5014, high: 65.4143, low: 59.8428, close: 61.8943, volume: 633706550},
{x: new Date('2013-05-20'), open: 61.7014, high: 64.05, low: 61.4428, close: 63.5928, volume: 494379068},
{x: new Date('2013-05-27'), open: 64.2714, high: 65.3, low: 62.7714, close: 64.2478, volume: 362907830},
{x: new Date('2013-06-03'), open: 64.39, high: 64.9186, low: 61.8243, close: 63.1158, volume: 443249793},
{x: new Date('2013-06-10'), open: 63.5328, high: 64.1541, low: 61.2143, close: 61.4357, volume: 389680092},
{x: new Date('2013-06-17'), open: 61.6343, high: 62.2428, low: 58.3, close: 59.0714, volume: 400384818},
{x: new Date('2013-06-24'), open: 58.2, high: 58.38, low: 55.5528, close: 56.6471, volume: 519314826},
{x: new Date('2013-07-01'), open: 57.5271, high: 60.47, low: 57.3171, close: 59.6314, volume: 343878841},
{x: new Date('2013-07-08'), open: 60.0157, high: 61.3986, low: 58.6257, close: 60.93, volume: 384106977},
```

```

    {x: new Date('2013-07-15'), open: 60.7157, high: 62.1243, low:
60.5957,close: 60.7071,volume: 286035513},
    {x: new Date('2013-07-22'), open: 61.3514, high: 63.5128, low:
59.8157,close: 62.9986,volume: 395816827},
    {x: new Date('2013-07-29'), open: 62.9714, high: 66.1214, low:
62.8857,close: 66.0771,volume: 339668858},
    {x: new Date('2013-08-12'), open: 65.2657, high: 72.0357, low:
65.2328,close: 71.7614,volume: 711563584},
    {x: new Date('2013-08-19'), open: 72.0485, high: 73.3914, low:
71.1714,close: 71.5743,volume: 417119660},
    {x: new Date('2013-08-26'), open: 71.5357, high: 72.8857, low:
69.4286,close: 69.6023,volume: 392805888},
    {x: new Date('2013-09-02'), open: 70.4428, high: 71.7485, low:
69.6214,close: 71.1743,volume: 317244380},
    {x: new Date('2013-09-09'), open: 72.1428, high: 72.56, low:
66.3857,close: 66.4143,volume: 669376320},
    {x: new Date('2013-09-16'), open: 65.8571, high: 68.3643, low:
63.8886,close: 66.7728,volume: 625142677},
    {x: new Date('2013-09-23'), open: 70.8714, high: 70.9871, low:
68.6743,close: 68.9643,volume: 475274537},
    {x: new Date('2013-09-30'), open: 68.1786, high: 70.3357, low:
67.773,close: 69.0043,volume: 368198906},
    {x: new Date('2013-10-07'), open: 69.5086, high: 70.5486, low:
68.3257,close: 70.4017,volume: 361437661},
    {x: new Date('2013-10-14'), open: 69.9757, high: 72.7514, low:
69.9071,close: 72.6985,volume: 342694379},
    {x: new Date('2013-10-21'), open: 73.11, high: 76.1757, low:
72.5757,close: 75.1368,volume: 490458997},
    {x: new Date('2013-10-28'), open: 75.5771, high: 77.0357, low:
73.5057,close: 74.29,volume: 508130174},
    {x: new Date('2013-11-04'), open: 74.4428, high: 75.555, low:
73.1971,close: 74.3657,volume: 318132218},
    {x: new Date('2013-11-11'), open: 74.2843, high: 75.6114, low:
73.4871,close: 74.9987,volume: 306711021},
    {x: new Date('2013-11-18'), open: 74.9985, high: 75.3128, low:
73.3814,close: 74.2571,volume: 282778778},
];
export default {
  data() {
    return {
      seriesData1: series1,
      primaryXAxis: {
        valueType: 'DateTime',
        majorGridLines: { width: 0 },
        zoomFactor: 0.2, zoomPosition: 0.6,
        crosshairTooltip: { enable: true }
      },
      chartArea: {
        border: {
          width: 0
        }
      },
      //Initializing Primary Y Axis
      primaryYAxis: {
        title: 'Price',
        labelFormat: '${value}M',
        minimum: 50, maximum: 170, interval: 30,

```

```

        majorGridLines: { width: 1 },
        lineStyle: { width: 0 }
    },
    indicators: [{
        type: 'BollingerBands', field: 'Close', seriesName: 'Apple Inc',
        fill: '#606eff',
        period: 14, animation: { enable: true }, upperLine: { color:
        '#ffb735', width: 1 }, lowerLine: { color: '#f2ec2f', width: 1 }
    }],
    animation: { enable: true },
    zoomSettings: {
        enableSelectionZooming: true,
        mode: 'X',
        enablePan : true
    },
    legendSettings: {
        visible: false
    },
    crosshair: { enable: true, lineType: 'Vertical' },
    tooltip: {
        enable: true, shared: true
    },
    title: "AAPL - 2012-2017"
    };
    },
    provide: {
        chart: [CandleSeries, Category, LineSeries, Tooltip, DateTime, Zoom,
        Logarithmic, Crosshair, RangeAreaSeries, BollingerBands]
    }
    };
</script>
<style>
    #container {
        height: 350px;
    }
</style>

```

{% previewsample "page.domainurl/code-snippet/chart/technical-indicators/bollinger-cs1" %}

- Customization of BollingerBand

stroke, stroke-width, and color of upperLine can be customized by using, [upperLine](#), and the lowerLine can be customized by using [lowerLine](#) properties of indicator.

#### APP.VUE

```

<template>
    <div id="app">
        <ejs-chart id="container" :title='title'
        :primaryXAxis='primaryXAxis' :primaryYAxis='primaryYAxis'
        :tooltip='tooltip' :chartArea='chartArea' :crosshair='crosshair'
        :indicators='indicators' :legendSettings='legendSettings'
        :zoomSettings='zoomSettings'>
            <e-series-collection>

```

```

        <e-series :dataSource='seriesData1' width=2 xName='x'
yName='y' low='low' high='high' close='close' open='open' volume='volume'
name='Apple Inc' bearFillColor='#2ecd71' bullFillColor='#e74c3d'
type='Candle' :animation='animation'> </e-series>
    </e-series-collection>
</ejs-chart>
</div>
</template>
<script>
import Vue from "vue";
import { ChartPlugin, CandleSeries, Category, Tooltip, DateTime, Zoom,
Logarithmic, Crosshair, LineSeries, BollingerBands, RangeAreaSeries } from
"@syncfusion/ej2-vue-charts";
Vue.use(ChartPlugin);
let series1: Object[] = [
    {x: new Date('2012-10-15'), open: 90.3357, high: 93.2557, low:
87.0885,close: 87.12,volume: 646996264},
    {x: new Date('2012-10-22'), open: 87.4885, high: 90.7685, low:
84.4285,close: 86.2857,volume: 866040680 },
    {x: new Date('2012-10-29'), open: 84.9828, high: 86.1428, low:
82.1071,close: 82.4,volume: 367371310},
    {x: new Date('2012-11-05'), open: 83.3593, high: 84.3914, low:
76.2457,close: 78.1514,volume: 919719846},
    {x: new Date('2012-11-12'), open: 79.1643, high: 79.2143, low:
72.25,close: 75.3825,volume: 894382149},
    {x: new Date('2012-11-19'), open: 77.2443, high: 81.7143, low:
77.1257,close: 81.6428,volume: 527416747},
    {x: new Date('2012-11-26'), open: 82.2714, high: 84.8928, low:
81.7514,close: 83.6114,volume: 646467974},
    {x: new Date('2012-12-03'), open: 84.8071, high: 84.9414, low:
74.09,close: 76.1785,volume: 980096264},
    {x: new Date('2012-12-10'), open: 75, high: 78.5085, low: 72.2257,close:
72.8277,volume: 835016110},
    {x: new Date('2012-12-17'), open: 72.7043, high: 76.4143, low:
71.6043,close: 74.19,volume: 726150329},
    {x: new Date('2012-12-24'), open: 74.3357, high: 74.8928, low:
72.0943,close: 72.7984,volume: 321104733},
    {x: new Date('2012-12-31'), open: 72.9328, high: 79.2857, low:
72.7143,close: 75.2857,volume: 540854882},
    {x: new Date('2013-01-07'), open: 74.5714, high: 75.9843, low:
73.6,close: 74.3285,volume: 574594262},
    {x: new Date('2013-01-14'), open: 71.8114, high: 72.9643, low:
69.0543,close: 71.4285,volume: 803105621},
    {x: new Date('2013-01-21'), open: 72.08, high: 73.57, low:
62.1428,close: 62.84,volume: 971912560},
    {x: new Date('2013-01-28'), open: 62.5464, high: 66.0857, low:
62.2657,close: 64.8028,volume: 656549587},
    {x: new Date('2013-02-04'), open: 64.8443, high: 68.4014, low:
63.1428,close: 67.8543,volume: 743778993},
    {x: new Date('2013-02-11'), open: 68.0714, high: 69.2771, low:
65.7028,close: 65.7371,volume: 585292366},
    {x: new Date('2013-02-18'), open: 65.8714, high: 66.1043, low:
63.26,close: 64.4014,volume: 421766997},
    {x: new Date('2013-02-25'), open: 64.8357, high: 65.0171, low:
61.4257,close: 61.4957,volume: 582741215},
    {x: new Date('2013-03-04'), open: 61.1143, high: 62.2043, low:
59.8571,close: 61.6743,volume: 632856539},

```

```
{x: new Date('2013-03-11'), open: 61.3928, high: 63.4614, low: 60.7343, close: 63.38, volume: 572066981},
{x: new Date('2013-03-18'), open: 63.0643, high: 66.0143, low: 63.0286, close: 65.9871, volume: 552156035},
{x: new Date('2013-03-25'), open: 66.3843, high: 67.1357, low: 63.0886, close: 63.2371, volume: 390762517},
{x: new Date('2013-04-01'), open: 63.1286, high: 63.3854, low: 59.9543, close: 60.4571, volume: 505273732},
{x: new Date('2013-04-08'), open: 60.6928, high: 62.57, low: 60.3557, close: 61.4, volume: 387323550},
{x: new Date('2013-04-15'), open: 61, high: 61.1271, low: 55.0143, close: 55.79, volume: 709945604},
{x: new Date('2013-04-22'), open: 56.0914, high: 59.8241, low: 55.8964, close: 59.6007, volume: 787007506},
{x: new Date('2013-04-29'), open: 60.0643, high: 64.7471, low: 60, close: 64.2828, volume: 655020017},
{x: new Date('2013-05-06'), open: 65.1014, high: 66.5357, low: 64.3543, close: 64.71, volume: 545488533},
{x: new Date('2013-05-13'), open: 64.5014, high: 65.4143, low: 59.8428, close: 61.8943, volume: 633706550},
{x: new Date('2013-05-20'), open: 61.7014, high: 64.05, low: 61.4428, close: 63.5928, volume: 494379068},
{x: new Date('2013-05-27'), open: 64.2714, high: 65.3, low: 62.7714, close: 64.2478, volume: 362907830},
{x: new Date('2013-06-03'), open: 64.39, high: 64.9186, low: 61.8243, close: 63.1158, volume: 443249793},
{x: new Date('2013-06-10'), open: 63.5328, high: 64.1541, low: 61.2143, close: 61.4357, volume: 389680092},
{x: new Date('2013-06-17'), open: 61.6343, high: 62.2428, low: 58.3, close: 59.0714, volume: 400384818},
{x: new Date('2013-06-24'), open: 58.2, high: 58.38, low: 55.5528, close: 56.6471, volume: 519314826},
{x: new Date('2013-07-01'), open: 57.5271, high: 60.47, low: 57.3171, close: 59.6314, volume: 343878841},
{x: new Date('2013-07-08'), open: 60.0157, high: 61.3986, low: 58.6257, close: 60.93, volume: 384106977},
{x: new Date('2013-07-15'), open: 60.7157, high: 62.1243, low: 60.5957, close: 60.7071, volume: 286035513},
{x: new Date('2013-07-22'), open: 61.3514, high: 63.5128, low: 59.8157, close: 62.9986, volume: 395816827},
{x: new Date('2013-07-29'), open: 62.9714, high: 66.1214, low: 62.8857, close: 66.0771, volume: 339668858},
{x: new Date('2013-08-12'), open: 65.2657, high: 72.0357, low: 65.2328, close: 71.7614, volume: 711563584},
{x: new Date('2013-08-19'), open: 72.0485, high: 73.3914, low: 71.1714, close: 71.5743, volume: 417119660},
{x: new Date('2013-08-26'), open: 71.5357, high: 72.8857, low: 69.4286, close: 69.6023, volume: 392805888},
{x: new Date('2013-09-02'), open: 70.4428, high: 71.7485, low: 69.6214, close: 71.1743, volume: 317244380},
{x: new Date('2013-09-09'), open: 72.1428, high: 72.56, low: 66.3857, close: 66.4143, volume: 669376320},
{x: new Date('2013-09-16'), open: 65.8571, high: 68.3643, low: 63.8886, close: 66.7728, volume: 625142677},
{x: new Date('2013-09-23'), open: 70.8714, high: 70.9871, low: 68.6743, close: 68.9643, volume: 475274537},
```

```

    {x: new Date('2013-09-30'), open: 68.1786, high: 70.3357, low:
67.773,close: 69.0043,volume: 368198906},
    {x: new Date('2013-10-07'), open: 69.5086, high: 70.5486, low:
68.3257,close: 70.4017,volume: 361437661},
    {x: new Date('2013-10-14'), open: 69.9757, high: 72.7514, low:
69.9071,close: 72.6985,volume: 342694379},
    {x: new Date('2013-10-21'), open: 73.11, high: 76.1757, low:
72.5757,close: 75.1368,volume: 490458997},
    {x: new Date('2013-10-28'), open: 75.5771, high: 77.0357, low:
73.5057,close: 74.29,volume: 508130174},
    {x: new Date('2013-11-04'), open: 74.4428, high: 75.555, low:
73.1971,close: 74.3657,volume: 318132218},
    {x: new Date('2013-11-11'), open: 74.2843, high: 75.6114, low:
73.4871,close: 74.9987,volume: 306711021},
    {x: new Date('2013-11-18'), open: 74.9985, high: 75.3128, low:
73.3814,close: 74.2571,volume: 282778778},
];
export default {
  data() {
    return {
      seriesData1: series1,
      primaryXAxis: {
        valueType: 'DateTime',
        majorGridLines: { width: 0 },
        zoomFactor: 0.2, zoomPosition: 0.6,
        crosshairTooltip: { enable: true }
      },
      chartArea: {
        border: {
          width: 0
        }
      },
      //Initializing Primary Y Axis
      primaryYAxis: {
        title: 'Price',
        labelFormat: '${value}M',
        minimum: 50, maximum: 170, interval: 30,
        majorGridLines: { width: 1 },
        lineStyle: { width: 0 }
      },
      indicators: [{
        type: 'BollingerBands', field: 'Close', seriesName: 'Apple Inc',
        fill: '#606eff',
        period: 14, animation: { enable: true }, upperLine: { color:
'#ffb735', width: 1 }, lowerLine: { color: '#f2ec2f', width: 1 }
      }],
      animation: { enable: true },
      zoomSettings: {
        enableSelectionZooming: true,
        mode: 'X',
        enablePan : true
      },
      legendSettings: {
        visible: false
      },
      crosshair: { enable: true, lineType: 'Vertical' },
      tooltip: {

```

```

        enable: true, shared: true
      },
      title: "AAPL - 2012-2017"
    };
  },
  provide: {
    chart: [CandleSeries, Category, LineSeries, Tooltip, DateTime, Zoom,
    Logarithmic, Crosshair, RangeAreaSeries, BollingerBands]
  }
};
</script>
<style>
  #container {
    height: 350px;
  }
</style>

```

{% previewsample "page.domainurl/code-snippet/chart/technical-indicators/bollinger-cs2" %}

### Exponential Moving Average (EMA)

Moving average Indicators are used to define the direction of the trend. To render a EMA Indicator, use indicator [type](#) as `Ema` and inject `EMAIndicator` into the `provide`.

### APP.VUE

```

<template>
  <div id="app">
    <ejs-chart id="container" :title='title'
    :primaryXAxis='primaryXAxis' :primaryYAxis='primaryYAxis'
    :chartArea='chartArea' :legendSettings='legendSettings'
    :indicators='indicators' :crosshair='crosshair' :tooltip='tooltip'
    :zoomSettings='zoomSettings'>
      <e-series-collection>
        <e-series :dataSource='seriesData1' type='Candle' xName='x'
        yName='y' name='Apple Inc' width=2 low='low' high='high' close='close'
        open='open' volume='volume' bearFillColor='#2ecd71' bullFillColor='#e74c3d'
        :animation='animation'> </e-series>
      </e-series-collection>
    </ejs-chart>
  </div>
</template>
<script>
import Vue from "vue";
import { ChartPlugin, Category, CandleSeries, Tooltip, DateTime, Zoom,
Crosshair, LineSeries, Logarithmic, EmaIndicator } from "@syncfusion/ej2-
vue-charts";
Vue.use(ChartPlugin);
let series1: Object[] = [
  {x: new Date('2012-10-15'), open: 90.3357, high: 93.2557, low:
87.0885,close: 87.12,volume: 646996264},
  {x: new Date('2012-10-22'), open: 87.4885, high: 90.7685, low:
84.4285,close: 86.2857,volume: 866040680 },
  {x: new Date('2012-10-29'), open: 84.9828, high: 86.1428, low:
82.1071,close: 82.4,volume: 367371310},
  {x: new Date('2012-11-05'), open: 83.3593, high: 84.3914, low:
76.2457,close: 78.1514,volume: 919719846},

```

```
{x: new Date('2012-11-12'), open: 79.1643, high: 79.2143, low: 72.25, close: 75.3825, volume: 894382149},
{x: new Date('2012-11-19'), open: 77.2443, high: 81.7143, low: 77.1257, close: 81.6428, volume: 527416747},
{x: new Date('2012-11-26'), open: 82.2714, high: 84.8928, low: 81.7514, close: 83.6114, volume: 646467974},
{x: new Date('2012-12-03'), open: 84.8071, high: 84.9414, low: 74.09, close: 76.1785, volume: 980096264},
{x: new Date('2012-12-10'), open: 75, high: 78.5085, low: 72.2257, close: 72.8277, volume: 835016110},
{x: new Date('2012-12-17'), open: 72.7043, high: 76.4143, low: 71.6043, close: 74.19, volume: 726150329},
{x: new Date('2012-12-24'), open: 74.3357, high: 74.8928, low: 72.0943, close: 72.7984, volume: 321104733},
{x: new Date('2012-12-31'), open: 72.9328, high: 79.2857, low: 72.7143, close: 75.2857, volume: 540854882},
{x: new Date('2013-01-07'), open: 74.5714, high: 75.9843, low: 73.6, close: 74.3285, volume: 574594262},
{x: new Date('2013-01-14'), open: 71.8114, high: 72.9643, low: 69.0543, close: 71.4285, volume: 803105621},
{x: new Date('2013-01-21'), open: 72.08, high: 73.57, low: 62.1428, close: 62.84, volume: 971912560},
{x: new Date('2013-01-28'), open: 62.5464, high: 66.0857, low: 62.2657, close: 64.8028, volume: 656549587},
{x: new Date('2013-02-04'), open: 64.8443, high: 68.4014, low: 63.1428, close: 67.8543, volume: 743778993},
{x: new Date('2013-02-11'), open: 68.0714, high: 69.2771, low: 65.7028, close: 65.7371, volume: 585292366},
{x: new Date('2013-02-18'), open: 65.8714, high: 66.1043, low: 63.26, close: 64.4014, volume: 421766997},
{x: new Date('2013-02-25'), open: 64.8357, high: 65.0171, low: 61.4257, close: 61.4957, volume: 582741215},
{x: new Date('2013-03-04'), open: 61.1143, high: 62.2043, low: 59.8571, close: 61.6743, volume: 632856539},
{x: new Date('2013-03-11'), open: 61.3928, high: 63.4614, low: 60.7343, close: 63.38, volume: 572066981},
{x: new Date('2013-03-18'), open: 63.0643, high: 66.0143, low: 63.0286, close: 65.9871, volume: 552156035},
{x: new Date('2013-03-25'), open: 66.3843, high: 67.1357, low: 63.0886, close: 63.2371, volume: 390762517},
{x: new Date('2013-04-01'), open: 63.1286, high: 63.3854, low: 59.9543, close: 60.4571, volume: 505273732},
{x: new Date('2013-04-08'), open: 60.6928, high: 62.57, low: 60.3557, close: 61.4, volume: 387323550},
{x: new Date('2013-04-15'), open: 61, high: 61.1271, low: 55.0143, close: 55.79, volume: 709945604},
{x: new Date('2013-04-22'), open: 56.0914, high: 59.8241, low: 55.8964, close: 59.6007, volume: 787007506},
{x: new Date('2013-04-29'), open: 60.0643, high: 64.7471, low: 60, close: 64.2828, volume: 655020017},
{x: new Date('2013-05-06'), open: 65.1014, high: 66.5357, low: 64.3543, close: 64.71, volume: 545488533},
{x: new Date('2013-05-13'), open: 64.5014, high: 65.4143, low: 59.8428, close: 61.8943, volume: 633706550},
{x: new Date('2013-05-20'), open: 61.7014, high: 64.05, low: 61.4428, close: 63.5928, volume: 494379068},
```



```

    {x: new Date('2013-05-27'), open: 64.2714, high: 65.3, low:
62.7714,close: 64.2478,volume: 362907830},
    {x: new Date('2013-06-03'), open: 64.39, high: 64.9186, low:
61.8243,close: 63.1158,volume: 443249793},
    {x: new Date('2013-06-10'), open: 63.5328, high: 64.1541, low:
61.2143,close: 61.4357,volume: 389680092},
    {x: new Date('2013-06-17'), open: 61.6343, high: 62.2428, low:
58.3,close: 59.0714,volume: 400384818},
    {x: new Date('2013-06-24'), open: 58.2, high: 58.38, low: 55.5528,close:
56.6471,volume: 519314826},
    {x: new Date('2013-07-01'), open: 57.5271, high: 60.47, low:
57.3171,close: 59.6314,volume: 343878841},
    {x: new Date('2013-07-08'), open: 60.0157, high: 61.3986, low:
58.6257,close: 60.93,volume: 384106977},
    {x: new Date('2013-07-15'), open: 60.7157, high: 62.1243, low:
60.5957,close: 60.7071,volume: 286035513},
    {x: new Date('2013-07-22'), open: 61.3514, high: 63.5128, low:
59.8157,close: 62.9986,volume: 395816827},
    {x: new Date('2013-07-29'), open: 62.9714, high: 66.1214, low:
62.8857,close: 66.0771,volume: 339668858},
    {x: new Date('2013-08-12'), open: 65.2657, high: 72.0357, low:
65.2328,close: 71.7614,volume: 711563584},
    {x: new Date('2013-08-19'), open: 72.0485, high: 73.3914, low:
71.1714,close: 71.5743,volume: 417119660},
    {x: new Date('2013-08-26'), open: 71.5357, high: 72.8857, low:
69.4286,close: 69.6023,volume: 392805888},
    {x: new Date('2013-09-02'), open: 70.4428, high: 71.7485, low:
69.6214,close: 71.1743,volume: 317244380},
    {x: new Date('2013-09-09'), open: 72.1428, high: 72.56, low:
66.3857,close: 66.4143,volume: 669376320},
    {x: new Date('2013-09-16'), open: 65.8571, high: 68.3643, low:
63.8886,close: 66.7728,volume: 625142677},
    {x: new Date('2013-09-23'), open: 70.8714, high: 70.9871, low:
68.6743,close: 68.9643,volume: 475274537},
    {x: new Date('2013-09-30'), open: 68.1786, high: 70.3357, low:
67.773,close: 69.0043,volume: 368198906},
    {x: new Date('2013-10-07'), open: 69.5086, high: 70.5486, low:
68.3257,close: 70.4017,volume: 361437661},
    {x: new Date('2013-10-14'), open: 69.9757, high: 72.7514, low:
69.9071,close: 72.6985,volume: 342694379},
    {x: new Date('2013-10-21'), open: 73.11, high: 76.1757, low:
72.5757,close: 75.1368,volume: 490458997},
    {x: new Date('2013-10-28'), open: 75.5771, high: 77.0357, low:
73.5057,close: 74.29,volume: 508130174},
    {x: new Date('2013-11-04'), open: 74.4428, high: 75.555, low:
73.1971,close: 74.3657,volume: 318132218},
    {x: new Date('2013-11-11'), open: 74.2843, high: 75.6114, low:
73.4871,close: 74.9987,volume: 306711021},
    {x: new Date('2013-11-18'), open: 74.9985, high: 75.3128, low:
73.3814,close: 74.2571,volume: 282778778},
  ];
export default {
  data() {
    return {
      seriesData1: series1,
      primaryXAxis: {
        valueType: 'DateTime',

```

```

        majorGridLines: { width: 0 },
        zoomFactor: 0.2, zoomPosition: 0.6,
        crosshairTooltip: { enable: true },
      }, chartArea: {
        border: {
          width: 0
        }
      },
    },
    //Initializing Primary Y Axis
    primaryYAxis: {
      title: 'Price',
      labelFormat: '${value}M',
      minimum: 50, maximum: 170, interval: 30,
      majorGridLines: { width: 1 },
      lineStyle: { width: 0 }
    },
    indicators: [{
      type: 'Ema', field: 'Close', seriesName: 'Apple Inc', fill:
'#606eff',
      period: 14, animation: { enable: true }
    }],
    tooltip: {
      enable: true, shared: true
    },
    animation: { enable: false },
    crosshair: { enable: true, lineType: 'Vertical' },
    zoomSettings: {
      enableSelectionZooming: true,
      mode: 'X',
      enablePan : true
    },
    legendSettings: { visible: false },
    title: "AAPL - 2012-2017"
  };
},
provide: {
  chart: [CandleSeries, Category, Tooltip, DateTime, Zoom, Logarithmic,
Crosshair, LineSeries, EmaIndicator]
}
};
</script>
<style>
  #container {
    height: 350px;
  }
</style>

```

{% previewsample "page.domainurl/code-snippet/chart/technical-indicators/ema-cs1" %}

### Momentum

Momentum shows the speed at which the price of the stock is changing. To render a Momentum indicator, use indicator `type` as `Momentum` and inject `MomentumIndicator` into the `provide`. Momentum indicator will be represented by two lines (upperLine, signalLine). In momentum indicator the upperBand value is always render at the value 100.

**APP.VUE**

```

<template>
  <div id="app">
    <ejs-chart id="container" :title='title'
:primaryXAxis='primaryXAxis' :primaryYAxis='primaryYAxis'
    :chartArea='chartArea' :legendSettings='legendSettings'
:indicators='indicators' :crosshair='crosshair' :tooltip='tooltip'
:zoomSettings='zoomSettings' :axes='axes' :rows='rows'>
      <e-series-collection>
        <e-series :dataSource='seriesData1' type='Candle' xName='x'
yName='y' name='Apple Inc' width=2 low='low' high='high' close='close'
open='open' volume='volume' bearFillColor='#2ecd71' bullFillColor='#e74c3d'
:animation='animation'> </e-series>
      </e-series-collection>
    </ejs-chart>
  </div>
</template>
<script>
import Vue from "vue";
import { ChartPlugin, Category, CandleSeries, Tooltip, DateTime, Zoom,
Crosshair, LineSeries, Logarithmic, StripLine, MomentumIndicator } from
"@syncfusion/ej2-vue-charts";
Vue.use(ChartPlugin);
let series1: Object[] = [
  {x: new Date('2012-10-15'), open: 90.3357, high: 93.2557, low:
87.0885,close: 87.12,volume: 646996264},
  {x: new Date('2012-10-22'), open: 87.4885, high: 90.7685, low:
84.4285,close: 86.2857,volume: 866040680 },
  {x: new Date('2012-10-29'), open: 84.9828, high: 86.1428, low:
82.1071,close: 82.4,volume: 367371310},
  {x: new Date('2012-11-05'), open: 83.3593, high: 84.3914, low:
76.2457,close: 78.1514,volume: 919719846},
  {x: new Date('2012-11-12'), open: 79.1643, high: 79.2143, low:
72.25,close: 75.3825,volume: 894382149},
  {x: new Date('2012-11-19'), open: 77.2443, high: 81.7143, low:
77.1257,close: 81.6428,volume: 527416747},
  {x: new Date('2012-11-26'), open: 82.2714, high: 84.8928, low:
81.7514,close: 83.6114,volume: 646467974},
  {x: new Date('2012-12-03'), open: 84.8071, high: 84.9414, low:
74.09,close: 76.1785,volume: 980096264},
  {x: new Date('2012-12-10'), open: 75, high: 78.5085, low: 72.2257,close:
72.8277,volume: 835016110},
  {x: new Date('2012-12-17'), open: 72.7043, high: 76.4143, low:
71.6043,close: 74.19,volume: 726150329},
  {x: new Date('2012-12-24'), open: 74.3357, high: 74.8928, low:
72.0943,close: 72.7984,volume: 321104733},
  {x: new Date('2012-12-31'), open: 72.9328, high: 79.2857, low:
72.7143,close: 75.2857,volume: 540854882},
  {x: new Date('2013-01-07'), open: 74.5714, high: 75.9843, low:
73.6,close: 74.3285,volume: 574594262},
  {x: new Date('2013-01-14'), open: 71.8114, high: 72.9643, low:
69.0543,close: 71.4285,volume: 803105621},
  {x: new Date('2013-01-21'), open: 72.08, high: 73.57, low:
62.1428,close: 62.84,volume: 971912560},
  {x: new Date('2013-01-28'), open: 62.5464, high: 66.0857, low:
62.2657,close: 64.8028,volume: 656549587},

```

```
{x: new Date('2013-02-04'), open: 64.8443, high: 68.4014, low: 63.1428, close: 67.8543, volume: 743778993},
{x: new Date('2013-02-11'), open: 68.0714, high: 69.2771, low: 65.7028, close: 65.7371, volume: 585292366},
{x: new Date('2013-02-18'), open: 65.8714, high: 66.1043, low: 63.26, close: 64.4014, volume: 421766997},
{x: new Date('2013-02-25'), open: 64.8357, high: 65.0171, low: 61.4257, close: 61.4957, volume: 582741215},
{x: new Date('2013-03-04'), open: 61.1143, high: 62.2043, low: 59.8571, close: 61.6743, volume: 632856539},
{x: new Date('2013-03-11'), open: 61.3928, high: 63.4614, low: 60.7343, close: 63.38, volume: 572066981},
{x: new Date('2013-03-18'), open: 63.0643, high: 66.0143, low: 63.0286, close: 65.9871, volume: 552156035},
{x: new Date('2013-03-25'), open: 66.3843, high: 67.1357, low: 63.0886, close: 63.2371, volume: 390762517},
{x: new Date('2013-04-01'), open: 63.1286, high: 63.3854, low: 59.9543, close: 60.4571, volume: 505273732},
{x: new Date('2013-04-08'), open: 60.6928, high: 62.57, low: 60.3557, close: 61.4, volume: 387323550},
{x: new Date('2013-04-15'), open: 61, high: 61.1271, low: 55.0143, close: 55.79, volume: 709945604},
{x: new Date('2013-04-22'), open: 56.0914, high: 59.8241, low: 55.8964, close: 59.6007, volume: 787007506},
{x: new Date('2013-04-29'), open: 60.0643, high: 64.7471, low: 60, close: 64.2828, volume: 655020017},
{x: new Date('2013-05-06'), open: 65.1014, high: 66.5357, low: 64.3543, close: 64.71, volume: 545488533},
{x: new Date('2013-05-13'), open: 64.5014, high: 65.4143, low: 59.8428, close: 61.8943, volume: 633706550},
{x: new Date('2013-05-20'), open: 61.7014, high: 64.05, low: 61.4428, close: 63.5928, volume: 494379068},
{x: new Date('2013-05-27'), open: 64.2714, high: 65.3, low: 62.7714, close: 64.2478, volume: 362907830},
{x: new Date('2013-06-03'), open: 64.39, high: 64.9186, low: 61.8243, close: 63.1158, volume: 443249793},
{x: new Date('2013-06-10'), open: 63.5328, high: 64.1541, low: 61.2143, close: 61.4357, volume: 389680092},
{x: new Date('2013-06-17'), open: 61.6343, high: 62.2428, low: 58.3, close: 59.0714, volume: 400384818},
{x: new Date('2013-06-24'), open: 58.2, high: 58.38, low: 55.5528, close: 56.6471, volume: 519314826},
{x: new Date('2013-07-01'), open: 57.5271, high: 60.47, low: 57.3171, close: 59.6314, volume: 343878841},
{x: new Date('2013-07-08'), open: 60.0157, high: 61.3986, low: 58.6257, close: 60.93, volume: 384106977},
{x: new Date('2013-07-15'), open: 60.7157, high: 62.1243, low: 60.5957, close: 60.7071, volume: 286035513},
{x: new Date('2013-07-22'), open: 61.3514, high: 63.5128, low: 59.8157, close: 62.9986, volume: 395816827},
{x: new Date('2013-07-29'), open: 62.9714, high: 66.1214, low: 62.8857, close: 66.0771, volume: 339668858},
{x: new Date('2013-08-12'), open: 65.2657, high: 72.0357, low: 65.2328, close: 71.7614, volume: 711563584},
{x: new Date('2013-08-19'), open: 72.0485, high: 73.3914, low: 71.1714, close: 71.5743, volume: 417119660},
```

```

    {x: new Date('2013-08-26'), open: 71.5357, high: 72.8857, low:
69.4286,close: 69.6023,volume: 392805888},
    {x: new Date('2013-09-02'), open: 70.4428, high: 71.7485, low:
69.6214,close: 71.1743,volume: 317244380},
    {x: new Date('2013-09-09'), open: 72.1428, high: 72.56, low:
66.3857,close: 66.4143,volume: 669376320},
    {x: new Date('2013-09-16'), open: 65.8571, high: 68.3643, low:
63.8886,close: 66.7728,volume: 625142677},
    {x: new Date('2013-09-23'), open: 70.8714, high: 70.9871, low:
68.6743,close: 68.9643,volume: 475274537},
    {x: new Date('2013-09-30'), open: 68.1786, high: 70.3357, low:
67.773,close: 69.0043,volume: 368198906},
    {x: new Date('2013-10-07'), open: 69.5086, high: 70.5486, low:
68.3257,close: 70.4017,volume: 361437661},
    {x: new Date('2013-10-14'), open: 69.9757, high: 72.7514, low:
69.9071,close: 72.6985,volume: 342694379},
    {x: new Date('2013-10-21'), open: 73.11, high: 76.1757, low:
72.5757,close: 75.1368,volume: 490458997},
    {x: new Date('2013-10-28'), open: 75.5771, high: 77.0357, low:
73.5057,close: 74.29,volume: 508130174},
    {x: new Date('2013-11-04'), open: 74.4428, high: 75.555, low:
73.1971,close: 74.3657,volume: 318132218},
    {x: new Date('2013-11-11'), open: 74.2843, high: 75.6114, low:
73.4871,close: 74.9987,volume: 306711021},
    {x: new Date('2013-11-18'), open: 74.9985, high: 75.3128, low:
73.3814,close: 74.2571,volume: 282778778},
];
export default {
  data() {
    return {
      seriesData1: series1,
      primaryXAxis: {
        valueType: 'DateTime',
        majorGridLines: { width: 0 },
        zoomFactor: 0.2, zoomPosition: 0.6,
        crosshairTooltip: { enable: true }
      },
      //Initializing Primary Y Axis
      primaryYAxis: {
        title: 'Price',
        labelFormat: '${value}',
        plotOffset: 25,
        minimum: 50, maximum: 170,
        interval: 30, rowIndex: 1, opposedPosition: true, lineStyle: {
width: 0 },
      },
      rows: [
        {
          height: '40%'
        }, {
          height: '60%'
        }
      ],
      axes: [{
        name: 'secondary',
        opposedPosition: true, rowIndex: 0,

```

```

        majorGridLines: { width: 0 }, lineStyle: { width: 0 }, minimum:
80, maximum: 120, interval: 20,
        majorTickLines: { width: 0 }, title: 'Momentum', stripLines: [
            {
                start: 80, end: 120, text: '', color: 'black', visible:
true,
                opacity: 0.03, zIndex: 'Behind'
            }
        ]],
        indicators: [{
            type: 'Momentum', field: 'Close', seriesName: 'Apple Inc',
yAxisName: 'secondary', fill: '#6063ff',
            period: 3, animation: { enable: true }, upperLine: { color:
'#e74c3d' }
        }],
        tooltip: {
            enable: true, shared: true
        },
        animation: { enable: true },
        crosshair: { enable: true, lineType: 'Vertical' },
        chartArea: { border: { width: 0 } },
        zoomSettings:
        {
            enableSelectionZooming: true,
            mode: 'X',
            enablePan : true
        },
        legendSettings: { visible: false },
        title: "AAPL 2012-2017"
    };
},
provide: {
    chart: [CandleSeries, Category, Tooltip, DateTime, Zoom, Logarithmic,
Crosshair, LineSeries, MomentumIndicator, StripLine]
}
};
</script>
<style>
    #container {
        height: 350px;
    }
</style>

```

{% previewsample "page.domainurl/code-snippet/chart/technical-indicators/momentum-cs1" %}

- Customization of MomentumIndicator

stroke, stroke-width, and color of upperLine can be customized by using [upperLine](#), property of indicator.

#### APP.VUE

```

<template>
    <div id="app">

```

```

    <ejs-chart id="container" :title='title'
:primaryXAxis='primaryXAxis' :primaryYAxis='primaryYAxis'
    :chartArea='chartArea' :legendSettings='legendSettings'
:indicators='indicators' :crosshair='crosshair' :tooltip='tooltip'
:zoomSettings='zoomSettings' :axes='axes' :rows='rows'>
      <e-series-collection>
        <e-series :dataSource='seriesData1' type='Candle' xName='x'
yName='y' name='Apple Inc' width=2 low='low' high='high' close='close'
open='open' volume='volume' bearFillColor='#2ecd71' bullFillColor='#e74c3d'
:animation='animation'> </e-series>
      </e-series-collection>
    </ejs-chart>
  </div>
</template>
<script>
import Vue from "vue";
import { ChartPlugin, Category, CandleSeries, Tooltip, DateTime, Zoom,
Crosshair, LineSeries, Logarithmic, StripLine, MomentumIndicator } from
"@syncfusion/ej2-vue-charts";
Vue.use(ChartPlugin);
let series1: Object[] = [
  {x: new Date('2012-10-15'), open: 90.3357, high: 93.2557, low:
87.0885,close: 87.12,volume: 646996264},
  {x: new Date('2012-10-22'), open: 87.4885, high: 90.7685, low:
84.4285,close: 86.2857,volume: 866040680 },
  {x: new Date('2012-10-29'), open: 84.9828, high: 86.1428, low:
82.1071,close: 82.4,volume: 367371310},
  {x: new Date('2012-11-05'), open: 83.3593, high: 84.3914, low:
76.2457,close: 78.1514,volume: 919719846},
  {x: new Date('2012-11-12'), open: 79.1643, high: 79.2143, low:
72.25,close: 75.3825,volume: 894382149},
  {x: new Date('2012-11-19'), open: 77.2443, high: 81.7143, low:
77.1257,close: 81.6428,volume: 527416747},
  {x: new Date('2012-11-26'), open: 82.2714, high: 84.8928, low:
81.7514,close: 83.6114,volume: 646467974},
  {x: new Date('2012-12-03'), open: 84.8071, high: 84.9414, low:
74.09,close: 76.1785,volume: 980096264},
  {x: new Date('2012-12-10'), open: 75, high: 78.5085, low: 72.2257,close:
72.8277,volume: 835016110},
  {x: new Date('2012-12-17'), open: 72.7043, high: 76.4143, low:
71.6043,close: 74.19,volume: 726150329},
  {x: new Date('2012-12-24'), open: 74.3357, high: 74.8928, low:
72.0943,close: 72.7984,volume: 321104733},
  {x: new Date('2012-12-31'), open: 72.9328, high: 79.2857, low:
72.7143,close: 75.2857,volume: 540854882},
  {x: new Date('2013-01-07'), open: 74.5714, high: 75.9843, low:
73.6,close: 74.3285,volume: 574594262},
  {x: new Date('2013-01-14'), open: 71.8114, high: 72.9643, low:
69.0543,close: 71.4285,volume: 803105621},
  {x: new Date('2013-01-21'), open: 72.08, high: 73.57, low:
62.1428,close: 62.84,volume: 971912560},
  {x: new Date('2013-01-28'), open: 62.5464, high: 66.0857, low:
62.2657,close: 64.8028,volume: 656549587},
  {x: new Date('2013-02-04'), open: 64.8443, high: 68.4014, low:
63.1428,close: 67.8543,volume: 743778993},
  {x: new Date('2013-02-11'), open: 68.0714, high: 69.2771, low:
65.7028,close: 65.7371,volume: 585292366},

```

```
{x: new Date('2013-02-18'), open: 65.8714, high: 66.1043, low: 63.26, close: 64.4014, volume: 421766997},
{x: new Date('2013-02-25'), open: 64.8357, high: 65.0171, low: 61.4257, close: 61.4957, volume: 582741215},
{x: new Date('2013-03-04'), open: 61.1143, high: 62.2043, low: 59.8571, close: 61.6743, volume: 632856539},
{x: new Date('2013-03-11'), open: 61.3928, high: 63.4614, low: 60.7343, close: 63.38, volume: 572066981},
{x: new Date('2013-03-18'), open: 63.0643, high: 66.0143, low: 63.0286, close: 65.9871, volume: 552156035},
{x: new Date('2013-03-25'), open: 66.3843, high: 67.1357, low: 63.0886, close: 63.2371, volume: 390762517},
{x: new Date('2013-04-01'), open: 63.1286, high: 63.3854, low: 59.9543, close: 60.4571, volume: 505273732},
{x: new Date('2013-04-08'), open: 60.6928, high: 62.57, low: 60.3557, close: 61.4, volume: 387323550},
{x: new Date('2013-04-15'), open: 61, high: 61.1271, low: 55.0143, close: 55.79, volume: 709945604},
{x: new Date('2013-04-22'), open: 56.0914, high: 59.8241, low: 55.8964, close: 59.6007, volume: 787007506},
{x: new Date('2013-04-29'), open: 60.0643, high: 64.7471, low: 60, close: 64.2828, volume: 655020017},
{x: new Date('2013-05-06'), open: 65.1014, high: 66.5357, low: 64.3543, close: 64.71, volume: 545488533},
{x: new Date('2013-05-13'), open: 64.5014, high: 65.4143, low: 59.8428, close: 61.8943, volume: 633706550},
{x: new Date('2013-05-20'), open: 61.7014, high: 64.05, low: 61.4428, close: 63.5928, volume: 494379068},
{x: new Date('2013-05-27'), open: 64.2714, high: 65.3, low: 62.7714, close: 64.2478, volume: 362907830},
{x: new Date('2013-06-03'), open: 64.39, high: 64.9186, low: 61.8243, close: 63.1158, volume: 443249793},
{x: new Date('2013-06-10'), open: 63.5328, high: 64.1541, low: 61.2143, close: 61.4357, volume: 389680092},
{x: new Date('2013-06-17'), open: 61.6343, high: 62.2428, low: 58.3, close: 59.0714, volume: 400384818},
{x: new Date('2013-06-24'), open: 58.2, high: 58.38, low: 55.5528, close: 56.6471, volume: 519314826},
{x: new Date('2013-07-01'), open: 57.5271, high: 60.47, low: 57.3171, close: 59.6314, volume: 343878841},
{x: new Date('2013-07-08'), open: 60.0157, high: 61.3986, low: 58.6257, close: 60.93, volume: 384106977},
{x: new Date('2013-07-15'), open: 60.7157, high: 62.1243, low: 60.5957, close: 60.7071, volume: 286035513},
{x: new Date('2013-07-22'), open: 61.3514, high: 63.5128, low: 59.8157, close: 62.9986, volume: 395816827},
{x: new Date('2013-07-29'), open: 62.9714, high: 66.1214, low: 62.8857, close: 66.0771, volume: 339668858},
{x: new Date('2013-08-12'), open: 65.2657, high: 72.0357, low: 65.2328, close: 71.7614, volume: 711563584},
{x: new Date('2013-08-19'), open: 72.0485, high: 73.3914, low: 71.1714, close: 71.5743, volume: 417119660},
{x: new Date('2013-08-26'), open: 71.5357, high: 72.8857, low: 69.4286, close: 69.6023, volume: 392805888},
{x: new Date('2013-09-02'), open: 70.4428, high: 71.7485, low: 69.6214, close: 71.1743, volume: 317244380},
```



```

    {x: new Date('2013-09-09'), open: 72.1428, high: 72.56, low:
66.3857,close: 66.4143,volume: 669376320},
    {x: new Date('2013-09-16'), open: 65.8571, high: 68.3643, low:
63.8886,close: 66.7728,volume: 625142677},
    {x: new Date('2013-09-23'), open: 70.8714, high: 70.9871, low:
68.6743,close: 68.9643,volume: 475274537},
    {x: new Date('2013-09-30'), open: 68.1786, high: 70.3357, low:
67.773,close: 69.0043,volume: 368198906},
    {x: new Date('2013-10-07'), open: 69.5086, high: 70.5486, low:
68.3257,close: 70.4017,volume: 361437661},
    {x: new Date('2013-10-14'), open: 69.9757, high: 72.7514, low:
69.9071,close: 72.6985,volume: 342694379},
    {x: new Date('2013-10-21'), open: 73.11, high: 76.1757, low:
72.5757,close: 75.1368,volume: 490458997},
    {x: new Date('2013-10-28'), open: 75.5771, high: 77.0357, low:
73.5057,close: 74.29,volume: 508130174},
    {x: new Date('2013-11-04'), open: 74.4428, high: 75.555, low:
73.1971,close: 74.3657,volume: 318132218},
    {x: new Date('2013-11-11'), open: 74.2843, high: 75.6114, low:
73.4871,close: 74.9987,volume: 306711021},
    {x: new Date('2013-11-18'), open: 74.9985, high: 75.3128, low:
73.3814,close: 74.2571,volume: 282778778},
];
export default {
  data() {
    return {
      seriesData1: series1,
      primaryXAxis: {
        valueType: 'DateTime',
        majorGridLines: { width: 0 },
        zoomFactor: 0.2, zoomPosition: 0.6,
        crosshairTooltip: { enable: true }
      },
      //Initializing Primary Y Axis
      primaryYAxis: {
        title: 'Price',
        labelFormat: '${value}',
        plotOffset: 25,
        minimum: 50, maximum: 170,
        interval: 30, rowIndex: 1, opposedPosition: true, lineStyle: {
width: 0 },
      },
      rows: [
        {
          height: '40%'
        }, {
          height: '60%'
        }
      ],
      axes: [{
        name: 'secondary',
        opposedPosition: true, rowIndex: 0,
        majorGridLines: { width: 0 }, lineStyle: { width: 0 }, minimum:
80, maximum: 120, interval: 20,
        majorTickLines: { width: 0 }, title: 'Momentum', stripLines: [

```

```

        start: 80, end: 120, text: '', color: 'black', visible:
true,
        opacity: 0.03, zIndex: 'Behind'
    }],
    indicators: [{
        type: 'Momentum', field: 'Close', seriesName: 'Apple Inc',
yAxisName: 'secondary', fill: '#6063ff',
        period: 3, animation: { enable: true }, upperLine: { color:
'#e74c3d' }
    }],
    tooltip: {
        enable: true, shared: true
    },
    animation: { enable: true },
    crosshair: { enable: true, lineType: 'Vertical' },
    chartArea: { border: { width: 0 } },
    zoomSettings:
    {
        enableSelectionZooming: true,
        mode: 'X',
        enablePan : true
    },
    legendSettings: { visible: false },
    title: "AAPL 2012-2017"
    };
    },
    provide: {
        chart: [CandleSeries, Category, Tooltip, DateTime, Zoom, Logarithmic,
Crosshair, LineSeries, MomentumIndicator, StripLine]
    }
    };
</script>
<style>
    #container {
        height: 350px;
    }
</style>

```

{% previewsample "page.domainurl/code-snippet/chart/technical-indicators/momentum-cs2" %}

### Moving Average Convergence Divergence (MACD)

MACD is based on the difference between two EMA's. To render a MACD Indicator, use indicator [type](#) as **MACD** and inject **MACDIndicator** into the **provide**. MACD indicator will be represented by MACD line, signal line, MACD histogram. MACD histogram is used to differentiate MACD line and signal line.

### APP.VUE

```

<template>
    <div id="app">
        <ejs-chart id="container" :title='title'
:primaryXAxis='primaryXAxis' :primaryYAxis='primaryYAxis'
        :chartArea='chartArea' :legendSettings='legendSettings'
:indicators='indicators' :crosshair='crosshair' :tooltip='tooltip'
:zoomSettings='zoomSettings' :axes='axes' :rows='rows'>
            <e-series-collection>

```

```

        <e-series :dataSource='seriesData1' type='Candle' xName='x'
yName='y' name='Apple Inc' width=2 low='low' high='high' close='close'
open='open' volume='volume' bearFillColor='#2ecd71' bullFillColor='#e74c3d'
:animation='animation'> </e-series>
    </e-series-collection>
</ejs-chart>
</div>
</template>
<script>
import Vue from "vue";
import { ChartPlugin, Category, CandleSeries, Tooltip, DateTime, Zoom,
Crosshair, LineSeries, Logarithmic, StripLine, ColumnSeries, MacdIndicator }
from "@syncfusion/ej2-vue-charts";
Vue.use(ChartPlugin);
let series1: Object[] = [
    {x: new Date('2012-10-15'), open: 90.3357, high: 93.2557, low:
87.0885,close: 87.12,volume: 646996264},
    {x: new Date('2012-10-22'), open: 87.4885, high: 90.7685, low:
84.4285,close: 86.2857,volume: 866040680 },
    {x: new Date('2012-10-29'), open: 84.9828, high: 86.1428, low:
82.1071,close: 82.4,volume: 367371310},
    {x: new Date('2012-11-05'), open: 83.3593, high: 84.3914, low:
76.2457,close: 78.1514,volume: 919719846},
    {x: new Date('2012-11-12'), open: 79.1643, high: 79.2143, low:
72.25,close: 75.3825,volume: 894382149},
    {x: new Date('2012-11-19'), open: 77.2443, high: 81.7143, low:
77.1257,close: 81.6428,volume: 527416747},
    {x: new Date('2012-11-26'), open: 82.2714, high: 84.8928, low:
81.7514,close: 83.6114,volume: 646467974},
    {x: new Date('2012-12-03'), open: 84.8071, high: 84.9414, low:
74.09,close: 76.1785,volume: 980096264},
    {x: new Date('2012-12-10'), open: 75, high: 78.5085, low: 72.2257,close:
72.8277,volume: 835016110},
    {x: new Date('2012-12-17'), open: 72.7043, high: 76.4143, low:
71.6043,close: 74.19,volume: 726150329},
    {x: new Date('2012-12-24'), open: 74.3357, high: 74.8928, low:
72.0943,close: 72.7984,volume: 321104733},
    {x: new Date('2012-12-31'), open: 72.9328, high: 79.2857, low:
72.7143,close: 75.2857,volume: 540854882},
    {x: new Date('2013-01-07'), open: 74.5714, high: 75.9843, low:
73.6,close: 74.3285,volume: 574594262},
    {x: new Date('2013-01-14'), open: 71.8114, high: 72.9643, low:
69.0543,close: 71.4285,volume: 803105621},
    {x: new Date('2013-01-21'), open: 72.08, high: 73.57, low:
62.1428,close: 62.84,volume: 971912560},
    {x: new Date('2013-01-28'), open: 62.5464, high: 66.0857, low:
62.2657,close: 64.8028,volume: 656549587},
    {x: new Date('2013-02-04'), open: 64.8443, high: 68.4014, low:
63.1428,close: 67.8543,volume: 743778993},
    {x: new Date('2013-02-11'), open: 68.0714, high: 69.2771, low:
65.7028,close: 65.7371,volume: 585292366},
    {x: new Date('2013-02-18'), open: 65.8714, high: 66.1043, low:
63.26,close: 64.4014,volume: 421766997},
    {x: new Date('2013-02-25'), open: 64.8357, high: 65.0171, low:
61.4257,close: 61.4957,volume: 582741215},
    {x: new Date('2013-03-04'), open: 61.1143, high: 62.2043, low:
59.8571,close: 61.6743,volume: 632856539},

```

```
{x: new Date('2013-03-11'), open: 61.3928, high: 63.4614, low: 60.7343, close: 63.38, volume: 572066981},
{x: new Date('2013-03-18'), open: 63.0643, high: 66.0143, low: 63.0286, close: 65.9871, volume: 552156035},
{x: new Date('2013-03-25'), open: 66.3843, high: 67.1357, low: 63.0886, close: 63.2371, volume: 390762517},
{x: new Date('2013-04-01'), open: 63.1286, high: 63.3854, low: 59.9543, close: 60.4571, volume: 505273732},
{x: new Date('2013-04-08'), open: 60.6928, high: 62.57, low: 60.3557, close: 61.4, volume: 387323550},
{x: new Date('2013-04-15'), open: 61, high: 61.1271, low: 55.0143, close: 55.79, volume: 709945604},
{x: new Date('2013-04-22'), open: 56.0914, high: 59.8241, low: 55.8964, close: 59.6007, volume: 787007506},
{x: new Date('2013-04-29'), open: 60.0643, high: 64.7471, low: 60, close: 64.2828, volume: 655020017},
{x: new Date('2013-05-06'), open: 65.1014, high: 66.5357, low: 64.3543, close: 64.71, volume: 545488533},
{x: new Date('2013-05-13'), open: 64.5014, high: 65.4143, low: 59.8428, close: 61.8943, volume: 633706550},
{x: new Date('2013-05-20'), open: 61.7014, high: 64.05, low: 61.4428, close: 63.5928, volume: 494379068},
{x: new Date('2013-05-27'), open: 64.2714, high: 65.3, low: 62.7714, close: 64.2478, volume: 362907830},
{x: new Date('2013-06-03'), open: 64.39, high: 64.9186, low: 61.8243, close: 63.1158, volume: 443249793},
{x: new Date('2013-06-10'), open: 63.5328, high: 64.1541, low: 61.2143, close: 61.4357, volume: 389680092},
{x: new Date('2013-06-17'), open: 61.6343, high: 62.2428, low: 58.3, close: 59.0714, volume: 400384818},
{x: new Date('2013-06-24'), open: 58.2, high: 58.38, low: 55.5528, close: 56.6471, volume: 519314826},
{x: new Date('2013-07-01'), open: 57.5271, high: 60.47, low: 57.3171, close: 59.6314, volume: 343878841},
{x: new Date('2013-07-08'), open: 60.0157, high: 61.3986, low: 58.6257, close: 60.93, volume: 384106977},
{x: new Date('2013-07-15'), open: 60.7157, high: 62.1243, low: 60.5957, close: 60.7071, volume: 286035513},
{x: new Date('2013-07-22'), open: 61.3514, high: 63.5128, low: 59.8157, close: 62.9986, volume: 395816827},
{x: new Date('2013-07-29'), open: 62.9714, high: 66.1214, low: 62.8857, close: 66.0771, volume: 339668858},
{x: new Date('2013-08-12'), open: 65.2657, high: 72.0357, low: 65.2328, close: 71.7614, volume: 711563584},
{x: new Date('2013-08-19'), open: 72.0485, high: 73.3914, low: 71.1714, close: 71.5743, volume: 417119660},
{x: new Date('2013-08-26'), open: 71.5357, high: 72.8857, low: 69.4286, close: 69.6023, volume: 392805888},
{x: new Date('2013-09-02'), open: 70.4428, high: 71.7485, low: 69.6214, close: 71.1743, volume: 317244380},
{x: new Date('2013-09-09'), open: 72.1428, high: 72.56, low: 66.3857, close: 66.4143, volume: 669376320},
{x: new Date('2013-09-16'), open: 65.8571, high: 68.3643, low: 63.8886, close: 66.7728, volume: 625142677},
{x: new Date('2013-09-23'), open: 70.8714, high: 70.9871, low: 68.6743, close: 68.9643, volume: 475274537},
```

```

    {x: new Date('2013-09-30'), open: 68.1786, high: 70.3357, low:
67.773,close: 69.0043,volume: 368198906},
    {x: new Date('2013-10-07'), open: 69.5086, high: 70.5486, low:
68.3257,close: 70.4017,volume: 361437661},
    {x: new Date('2013-10-14'), open: 69.9757, high: 72.7514, low:
69.9071,close: 72.6985,volume: 342694379},
    {x: new Date('2013-10-21'), open: 73.11, high: 76.1757, low:
72.5757,close: 75.1368,volume: 490458997},
    {x: new Date('2013-10-28'), open: 75.5771, high: 77.0357, low:
73.5057,close: 74.29,volume: 508130174},
    {x: new Date('2013-11-04'), open: 74.4428, high: 75.555, low:
73.1971,close: 74.3657,volume: 318132218},
    {x: new Date('2013-11-11'), open: 74.2843, high: 75.6114, low:
73.4871,close: 74.9987,volume: 306711021},
    {x: new Date('2013-11-18'), open: 74.9985, high: 75.3128, low:
73.3814,close: 74.2571,volume: 282778778},
];
export default {
  data() {
    return {
      seriesData1: series1,
      primaryXAxis: {
        valueType: 'DateTime',
        majorGridLines: { width: 0 },
        zoomFactor: 0.2, zoomPosition: 0.6,
        crosshairTooltip: { enable: true }
      },
      //Initializing Primary Y Axis
      primaryYAxis: {
        title: 'Price',
        labelFormat: '${value}',
        plotOffset: 25,
        minimum: 50, maximum: 170,
        interval: 30, rowIndex: 1, opposedPosition: true, lineStyle: {
width: 0 }
      },
      rows: [
        {
          height: '40%'
        }, {
          height: '60%'
        }
      ],
      axes: [{
        name: 'secondary',
        opposedPosition: true, rowIndex: 0,
        majorGridLines: { width: 0 }, lineStyle: { width: 0 }, minimum:
-3.5, maximum: 3.5, interval: 3.5,
        majorTickLines: { width: 0 }, title: 'MACD', stripLines: [
          {
            start: -3.5, end: 3.5, text: '', color: 'black',
visible: true,
            opacity: 0.03, zIndex: 'Behind'
          }
        ]
      }],
      indicators: [{
        type: 'Macd',

```

```

        period: 3,
        fastPeriod: 8,
        slowPeriod: 5,
        seriesName: 'Apple Inc',
        macdType: 'Both',
        width: 2,
        macdPositiveColor: '#2ecd71',
        macdNegativeColor: '#e74c3d',
        fill: '#6063ff',
        yAxisName: 'secondary'
    }],
    tooltip: {
        enable: true, shared: true
    },
    animation: { enable: true },
    crosshair: { enable: true, lineType: 'Vertical' },
    chartArea: { border: { width: 0 } },
    zoomSettings: {
        enableSelectionZooming: true,
        mode: 'X',
        enablePan : true
    },
    legendSettings: { visible: false },
    title: "AAPL 2012-2017"
};
},
provide: {
    chart: [CandleSeries, Category, Tooltip, DateTime, Zoom, Logarithmic,
    Crosshair, LineSeries, MacdIndicator, StripLine, ColumnSeries]
}
};
</script>
<style>
    #container {
        height: 350px;
    }
</style>

```

{% previewsample "page.domainurl/code-snippet/chart/technical-indicators/macd-cs1" %}

- Customization of MACD

`stroke`, `stroke-width`, and `color` of `macdLine` can be customized by using `macdLine` property of indicator. The positive and negative changes of histogram can be customized by `macdPositiveColor` and `macdNegativeColor` properties. The `macdType` is used to define the type of MACD indicator. To customize the MACD period using `slowPeriod` and `fastPeriod` properties.

#### APP.VUE

```

<template>
  <div id="app">

```

```

<ejs-chart id="container" :title='title'
:primaryXAxis='primaryXAxis' :primaryYAxis='primaryYAxis'
:chartArea='chartArea' :legendSettings='legendSettings'
:indicators='indicators' :crosshair='crosshair' :tooltip='tooltip'
:zoomSettings='zoomSettings' :axes='axes' :rows='rows'>
  <e-series-collection>
    <e-series :dataSource='seriesData1' type='Candle' xName='x'
yName='y' name='Apple Inc' width=2 low='low' high='high' close='close'
open='open' volume='volume' bearFillColor='#2ecd71' bullFillColor='#e74c3d'
:animation='animation'> </e-series>
  </e-series-collection>
</ejs-chart>
</div>
</template>
<script>
import Vue from "vue";
import { ChartPlugin, Category, CandleSeries, Tooltip, DateTime, Zoom,
Crosshair, LineSeries, Logarithmic, StripLine, ColumnSeries, MacdIndicator }
from "@syncfusion/ej2-vue-charts";
Vue.use(ChartPlugin);
let series1: Object[] = [
  {x: new Date('2012-10-15'), open: 90.3357, high: 93.2557, low:
87.0885,close: 87.12,volume: 646996264},
  {x: new Date('2012-10-22'), open: 87.4885, high: 90.7685, low:
84.4285,close: 86.2857,volume: 866040680 },
  {x: new Date('2012-10-29'), open: 84.9828, high: 86.1428, low:
82.1071,close: 82.4,volume: 367371310},
  {x: new Date('2012-11-05'), open: 83.3593, high: 84.3914, low:
76.2457,close: 78.1514,volume: 919719846},
  {x: new Date('2012-11-12'), open: 79.1643, high: 79.2143, low:
72.25,close: 75.3825,volume: 894382149},
  {x: new Date('2012-11-19'), open: 77.2443, high: 81.7143, low:
77.1257,close: 81.6428,volume: 527416747},
  {x: new Date('2012-11-26'), open: 82.2714, high: 84.8928, low:
81.7514,close: 83.6114,volume: 646467974},
  {x: new Date('2012-12-03'), open: 84.8071, high: 84.9414, low:
74.09,close: 76.1785,volume: 980096264},
  {x: new Date('2012-12-10'), open: 75, high: 78.5085, low: 72.2257,close:
72.8277,volume: 835016110},
  {x: new Date('2012-12-17'), open: 72.7043, high: 76.4143, low:
71.6043,close: 74.19,volume: 726150329},
  {x: new Date('2012-12-24'), open: 74.3357, high: 74.8928, low:
72.0943,close: 72.7984,volume: 321104733},
  {x: new Date('2012-12-31'), open: 72.9328, high: 79.2857, low:
72.7143,close: 75.2857,volume: 540854882},
  {x: new Date('2013-01-07'), open: 74.5714, high: 75.9843, low:
73.6,close: 74.3285,volume: 574594262},
  {x: new Date('2013-01-14'), open: 71.8114, high: 72.9643, low:
69.0543,close: 71.4285,volume: 803105621},
  {x: new Date('2013-01-21'), open: 72.08, high: 73.57, low:
62.1428,close: 62.84,volume: 971912560},
  {x: new Date('2013-01-28'), open: 62.5464, high: 66.0857, low:
62.2657,close: 64.8028,volume: 656549587},
  {x: new Date('2013-02-04'), open: 64.8443, high: 68.4014, low:
63.1428,close: 67.8543,volume: 743778993},
  {x: new Date('2013-02-11'), open: 68.0714, high: 69.2771, low:
65.7028,close: 65.7371,volume: 585292366},

```

```

    {x: new Date('2013-02-18'), open: 65.8714, high: 66.1043, low:
63.26,close: 64.4014,volume: 421766997},
    {x: new Date('2013-02-25'), open: 64.8357, high: 65.0171, low:
61.4257,close: 61.4957,volume: 582741215},
    {x: new Date('2013-03-04'), open: 61.1143, high: 62.2043, low:
59.8571,close: 61.6743,volume: 632856539},
    {x: new Date('2013-03-11'), open: 61.3928, high: 63.4614, low:
60.7343,close: 63.38,volume: 572066981},
    {x: new Date('2013-03-18'), open: 63.0643, high: 66.0143, low:
63.0286,close: 65.9871,volume: 552156035},
    {x: new Date('2013-03-25'), open: 66.3843, high: 67.1357, low:
63.0886,close: 63.2371,volume: 390762517},
    {x: new Date('2013-04-01'), open: 63.1286, high: 63.3854, low:
59.9543,close: 60.4571,volume: 505273732},
    {x: new Date('2013-04-08'), open: 60.6928, high: 62.57, low:
60.3557,close: 61.4,volume: 387323550},
    {x: new Date('2013-04-15'), open: 61, high: 61.1271, low: 55.0143,close:
55.79,volume: 709945604},
    {x: new Date('2013-04-22'), open: 56.0914, high: 59.8241, low:
55.8964,close: 59.6007,volume: 787007506},
    {x: new Date('2013-04-29'), open: 60.0643, high: 64.7471, low: 60,close:
64.2828,volume: 655020017},
    {x: new Date('2013-05-06'), open: 65.1014, high: 66.5357, low:
64.3543,close: 64.71,volume: 545488533},
    {x: new Date('2013-05-13'), open: 64.5014, high: 65.4143, low:
59.8428,close: 61.8943,volume: 633706550},
    {x: new Date('2013-05-20'), open: 61.7014, high: 64.05, low:
61.4428,close: 63.5928,volume: 494379068},
    {x: new Date('2013-05-27'), open: 64.2714, high: 65.3, low:
62.7714,close: 64.2478,volume: 362907830},
    {x: new Date('2013-06-03'), open: 64.39, high: 64.9186, low:
61.8243,close: 63.1158,volume: 443249793},
    {x: new Date('2013-06-10'), open: 63.5328, high: 64.1541, low:
61.2143,close: 61.4357,volume: 389680092},
    {x: new Date('2013-06-17'), open: 61.6343, high: 62.2428, low:
58.3,close: 59.0714,volume: 400384818},
    {x: new Date('2013-06-24'), open: 58.2, high: 58.38, low: 55.5528,close:
56.6471,volume: 519314826},
    {x: new Date('2013-07-01'), open: 57.5271, high: 60.47, low:
57.3171,close: 59.6314,volume: 343878841},
    {x: new Date('2013-07-08'), open: 60.0157, high: 61.3986, low:
58.6257,close: 60.93,volume: 384106977},
    {x: new Date('2013-07-15'), open: 60.7157, high: 62.1243, low:
60.5957,close: 60.7071,volume: 286035513},
    {x: new Date('2013-07-22'), open: 61.3514, high: 63.5128, low:
59.8157,close: 62.9986,volume: 395816827},
    {x: new Date('2013-07-29'), open: 62.9714, high: 66.1214, low:
62.8857,close: 66.0771,volume: 339668858},
    {x: new Date('2013-08-12'), open: 65.2657, high: 72.0357, low:
65.2328,close: 71.7614,volume: 711563584},
    {x: new Date('2013-08-19'), open: 72.0485, high: 73.3914, low:
71.1714,close: 71.5743,volume: 417119660},
    {x: new Date('2013-08-26'), open: 71.5357, high: 72.8857, low:
69.4286,close: 69.6023,volume: 392805888},
    {x: new Date('2013-09-02'), open: 70.4428, high: 71.7485, low:
69.6214,close: 71.1743,volume: 317244380},

```



```

    {x: new Date('2013-09-09'), open: 72.1428, high: 72.56, low:
66.3857,close: 66.4143,volume: 669376320},
    {x: new Date('2013-09-16'), open: 65.8571, high: 68.3643, low:
63.8886,close: 66.7728,volume: 625142677},
    {x: new Date('2013-09-23'), open: 70.8714, high: 70.9871, low:
68.6743,close: 68.9643,volume: 475274537},
    {x: new Date('2013-09-30'), open: 68.1786, high: 70.3357, low:
67.773,close: 69.0043,volume: 368198906},
    {x: new Date('2013-10-07'), open: 69.5086, high: 70.5486, low:
68.3257,close: 70.4017,volume: 361437661},
    {x: new Date('2013-10-14'), open: 69.9757, high: 72.7514, low:
69.9071,close: 72.6985,volume: 342694379},
    {x: new Date('2013-10-21'), open: 73.11, high: 76.1757, low:
72.5757,close: 75.1368,volume: 490458997},
    {x: new Date('2013-10-28'), open: 75.5771, high: 77.0357, low:
73.5057,close: 74.29,volume: 508130174},
    {x: new Date('2013-11-04'), open: 74.4428, high: 75.555, low:
73.1971,close: 74.3657,volume: 318132218},
    {x: new Date('2013-11-11'), open: 74.2843, high: 75.6114, low:
73.4871,close: 74.9987,volume: 306711021},
    {x: new Date('2013-11-18'), open: 74.9985, high: 75.3128, low:
73.3814,close: 74.2571,volume: 282778778},
];
export default {
  data() {
    return {
      seriesData1: series1,
      primaryXAxis: {
        valueType: 'DateTime',
        majorGridLines: { width: 0 },
        zoomFactor: 0.2, zoomPosition: 0.6,
        crosshairTooltip: { enable: true }
      },
      //Initializing Primary Y Axis
      primaryYAxis: {
        title: 'Price',
        labelFormat: '${value}',
        plotOffset: 25,
        minimum: 50, maximum: 170,
        interval: 30, rowIndex: 1, opposedPosition: true, lineStyle: {
width: 0 }
      },
      rows: [
        {
          height: '40%'
        }, {
          height: '60%'
        }
      ],
      axes: [{
        name: 'secondary',
        opposedPosition: true, rowIndex: 0,
        majorGridLines: { width: 0 }, lineStyle: { width: 0 }, minimum:
-3.5, maximum: 3.5, interval: 3.5,
        majorTickLines: { width: 0 }, title: 'MACD', stripLines: [

```

```

        start: -3.5, end: 3.5, text: '', color: 'black',
visible: true,
        opacity: 0.03, zIndex: 'Behind'
    }],
    indicators: [{
        type: 'Macd',
        period: 3,
        fastPeriod: 8,
        slowPeriod: 5,
        seriesName: 'Apple Inc',
        macdType: 'Both',
        width: 2,
        macdPositiveColor: '#2ecd71',
        macdNegativeColor: '#e74c3d',
        fill: '#6063ff',
        yAxisName: 'secondary'
    }],
    tooltip: {
        enable: true, shared: true
    },
    animation: { enable: true },
    crosshair: { enable: true, lineType: 'Vertical' },
    chartArea: { border: { width: 0 } },
    zoomSettings: {
        enableSelectionZooming: true,
        mode: 'X',
        enablePan : true
    },
    legendSettings: { visible: false },
    title: "AAPL 2012-2017"
};
},
provide: {
    chart: [CandleSeries, Category, Tooltip, DateTime, Zoom, Logarithmic,
Crosshair, LineSeries, MacdIndicator, StripLine, ColumnSeries]
}
};
</script>
<style>
#container {
    height: 350px;
}
</style>

```

{% previewsample "page.domainurl/code-snippet/chart/technical-indicators/macd-cs2" %}

### Relative Strength Index (RSI)

RSI shows how strongly a stock is moving in its current direction. To render a RSI Indicator, use

indicator `type` as `Rsi` and inject `RsiIndicator` into the `.provide`. RSI indicator will be represented by three lines (upperBand, lowerBand, signalLine). The upperBand and lowerBand values are customized by `overBought` and `overSold` properties of indicator and the signalLine is calculated by RSI formula.

### APP.VUE

```

<template>
  <div id="app">
    <ejs-chart id="container" :title='title'
:primaryXAxis='primaryXAxis' :primaryYAxis='primaryYAxis'
      :chartArea='chartArea' :legendSettings='legendSettings'
:indicators='indicators' :crosshair='crosshair' :tooltip='tooltip'
:zoomSettings='zoomSettings' :axes='axes' :rows='rows'>
      <e-series-collection>
        <e-series :dataSource='seriesData1' type='Candle' xName='x'
yName='y' name='Apple Inc' width=2 low='low' high='high' close='close'
open='open' volume='volume' bearFillColor='#2ecd71' bullFillColor='#e74c3d'
:animation='animation'> </e-series>
      </e-series-collection>
    </ejs-chart>
  </div>
</template>
<script>
import Vue from "vue";
import { ChartPlugin, Category, CandleSeries, Tooltip, DateTime, Zoom,
Crosshair, LineSeries, Logarithmic, StripLine, RsiIndicator } from
"@syncfusion/ej2-vue-charts";
Vue.use(ChartPlugin);
let series1: Object[] = [
  {x: new Date('2012-10-15'), open: 90.3357, high: 93.2557, low:
87.0885,close: 87.12,volume: 646996264},
  {x: new Date('2012-10-22'), open: 87.4885, high: 90.7685, low:
84.4285,close: 86.2857,volume: 866040680 },
  {x: new Date('2012-10-29'), open: 84.9828, high: 86.1428, low:
82.1071,close: 82.4,volume: 367371310},
  {x: new Date('2012-11-05'), open: 83.3593, high: 84.3914, low:
76.2457,close: 78.1514,volume: 919719846},
  {x: new Date('2012-11-12'), open: 79.1643, high: 79.2143, low:
72.25,close: 75.3825,volume: 894382149},
  {x: new Date('2012-11-19'), open: 77.2443, high: 81.7143, low:
77.1257,close: 81.6428,volume: 527416747},
  {x: new Date('2012-11-26'), open: 82.2714, high: 84.8928, low:
81.7514,close: 83.6114,volume: 646467974},
  {x: new Date('2012-12-03'), open: 84.8071, high: 84.9414, low:
74.09,close: 76.1785,volume: 980096264},
  {x: new Date('2012-12-10'), open: 75, high: 78.5085, low: 72.2257,close:
72.8277,volume: 835016110},
  {x: new Date('2012-12-17'), open: 72.7043, high: 76.4143, low:
71.6043,close: 74.19,volume: 726150329},
  {x: new Date('2012-12-24'), open: 74.3357, high: 74.8928, low:
72.0943,close: 72.7984,volume: 321104733},
  {x: new Date('2012-12-31'), open: 72.9328, high: 79.2857, low:
72.7143,close: 75.2857,volume: 540854882},
  {x: new Date('2013-01-07'), open: 74.5714, high: 75.9843, low:
73.6,close: 74.3285,volume: 574594262},
  {x: new Date('2013-01-14'), open: 71.8114, high: 72.9643, low:
69.0543,close: 71.4285,volume: 803105621},
  {x: new Date('2013-01-21'), open: 72.08, high: 73.57, low:
62.1428,close: 62.84,volume: 971912560},
  {x: new Date('2013-01-28'), open: 62.5464, high: 66.0857, low:
62.2657,close: 64.8028,volume: 656549587},
  {x: new Date('2013-02-04'), open: 64.8443, high: 68.4014, low:
63.1428,close: 67.8543,volume: 743778993},

```

```

    {x: new Date('2013-02-11'), open: 68.0714, high: 69.2771, low:
65.7028,close: 65.7371,volume: 585292366},
    {x: new Date('2013-02-18'), open: 65.8714, high: 66.1043, low:
63.26,close: 64.4014,volume: 421766997},
    {x: new Date('2013-02-25'), open: 64.8357, high: 65.0171, low:
61.4257,close: 61.4957,volume: 582741215},
    {x: new Date('2013-03-04'), open: 61.1143, high: 62.2043, low:
59.8571,close: 61.6743,volume: 632856539},
    {x: new Date('2013-03-11'), open: 61.3928, high: 63.4614, low:
60.7343,close: 63.38,volume: 572066981},
    {x: new Date('2013-03-18'), open: 63.0643, high: 66.0143, low:
63.0286,close: 65.9871,volume: 552156035},
    {x: new Date('2013-03-25'), open: 66.3843, high: 67.1357, low:
63.0886,close: 63.2371,volume: 390762517},
    {x: new Date('2013-04-01'), open: 63.1286, high: 63.3854, low:
59.9543,close: 60.4571,volume: 505273732},
    {x: new Date('2013-04-08'), open: 60.6928, high: 62.57, low:
60.3557,close: 61.4,volume: 387323550},
    {x: new Date('2013-04-15'), open: 61, high: 61.1271, low: 55.0143,close:
55.79,volume: 709945604},
    {x: new Date('2013-04-22'), open: 56.0914, high: 59.8241, low:
55.8964,close: 59.6007,volume: 787007506},
    {x: new Date('2013-04-29'), open: 60.0643, high: 64.7471, low: 60,close:
64.2828,volume: 655020017},
    {x: new Date('2013-05-06'), open: 65.1014, high: 66.5357, low:
64.3543,close: 64.71,volume: 545488533},
    {x: new Date('2013-05-13'), open: 64.5014, high: 65.4143, low:
59.8428,close: 61.8943,volume: 633706550},
    {x: new Date('2013-05-20'), open: 61.7014, high: 64.05, low:
61.4428,close: 63.5928,volume: 494379068},
    {x: new Date('2013-05-27'), open: 64.2714, high: 65.3, low:
62.7714,close: 64.2478,volume: 362907830},
    {x: new Date('2013-06-03'), open: 64.39, high: 64.9186, low:
61.8243,close: 63.1158,volume: 443249793},
    {x: new Date('2013-06-10'), open: 63.5328, high: 64.1541, low:
61.2143,close: 61.4357,volume: 389680092},
    {x: new Date('2013-06-17'), open: 61.6343, high: 62.2428, low:
58.3,close: 59.0714,volume: 400384818},
    {x: new Date('2013-06-24'), open: 58.2, high: 58.38, low: 55.5528,close:
56.6471,volume: 519314826},
    {x: new Date('2013-07-01'), open: 57.5271, high: 60.47, low:
57.3171,close: 59.6314,volume: 343878841},
    {x: new Date('2013-07-08'), open: 60.0157, high: 61.3986, low:
58.6257,close: 60.93,volume: 384106977},
    {x: new Date('2013-07-15'), open: 60.7157, high: 62.1243, low:
60.5957,close: 60.7071,volume: 286035513},
    {x: new Date('2013-07-22'), open: 61.3514, high: 63.5128, low:
59.8157,close: 62.9986,volume: 395816827},
    {x: new Date('2013-07-29'), open: 62.9714, high: 66.1214, low:
62.8857,close: 66.0771,volume: 339668858},
    {x: new Date('2013-08-12'), open: 65.2657, high: 72.0357, low:
65.2328,close: 71.7614,volume: 711563584},
    {x: new Date('2013-08-19'), open: 72.0485, high: 73.3914, low:
71.1714,close: 71.5743,volume: 417119660},
    {x: new Date('2013-08-26'), open: 71.5357, high: 72.8857, low:
69.4286,close: 69.6023,volume: 392805888},

```

```

    {x: new Date('2013-09-02'), open: 70.4428, high: 71.7485, low:
69.6214,close: 71.1743,volume: 317244380},
    {x: new Date('2013-09-09'), open: 72.1428, high: 72.56, low:
66.3857,close: 66.4143,volume: 669376320},
    {x: new Date('2013-09-16'), open: 65.8571, high: 68.3643, low:
63.8886,close: 66.7728,volume: 625142677},
    {x: new Date('2013-09-23'), open: 70.8714, high: 70.9871, low:
68.6743,close: 68.9643,volume: 475274537},
    {x: new Date('2013-09-30'), open: 68.1786, high: 70.3357, low:
67.773,close: 69.0043,volume: 368198906},
    {x: new Date('2013-10-07'), open: 69.5086, high: 70.5486, low:
68.3257,close: 70.4017,volume: 361437661},
    {x: new Date('2013-10-14'), open: 69.9757, high: 72.7514, low:
69.9071,close: 72.6985,volume: 342694379},
    {x: new Date('2013-10-21'), open: 73.11, high: 76.1757, low:
72.5757,close: 75.1368,volume: 490458997},
    {x: new Date('2013-10-28'), open: 75.5771, high: 77.0357, low:
73.5057,close: 74.29,volume: 508130174},
    {x: new Date('2013-11-04'), open: 74.4428, high: 75.555, low:
73.1971,close: 74.3657,volume: 318132218},
    {x: new Date('2013-11-11'), open: 74.2843, high: 75.6114, low:
73.4871,close: 74.9987,volume: 306711021},
    {x: new Date('2013-11-18'), open: 74.9985, high: 75.3128, low:
73.3814,close: 74.2571,volume: 282778778},
  ];
export default {
  data() {
    return {
      seriesData1: series1,
      primaryXAxis: {
        valueType: 'DateTime',
        majorGridLines: { width: 0 },
        zoomFactor: 0.2, zoomPosition: 0.6,
        crosshairTooltip: { enable: true },
      },
      //Initializing Primary Y Axis
      primaryYAxis: {
        title: 'Price',
        labelFormat: '${value}',
        plotOffset: 25,
        minimum: 50, maximum: 170,
        interval: 30, rowIndex: 1, opposedPosition: true, lineStyle: {
width: 0 }
      },
      rows: [
        {
          height: '40%'
        }, {
          height: '60%'
        }
      ],
      axes: [{
        name: 'secondary',
        opposedPosition: true, rowIndex: 0,
        majorGridLines: { width: 0 }, lineStyle: { width: 0 }, minimum:
0, maximum: 120, interval: 60,
        majorTickLines: { width: 0 }, title: 'RSI', stripLines: [

```

```

        {
            start: 0, end: 120, text: '', color: 'black', visible:
true,
            opacity: 0.03, zIndex: 'Behind'
        }
    ],
    indicators: [{
        type: 'Rsi', field: 'Close', seriesName: 'Apple Inc', yAxisName:
'secondary', fill: '#6063ff',
        showZones: true, overBought: 70, overSold: 30,
        period: 3, animation: { enable: true }, upperLine: { color:
'#e74c3d' }, lowerLine: { color: '#2ecd71' }
    }],
    tooltip: {
        enable: true, shared: true
    },
    animation: { enable: true },
    crosshair: { enable: true, lineType: 'Vertical' },
    chartArea: { border: { width: 0 } },
    zoomSettings:
    {
        enableSelectionZooming: true,
        mode: 'X',
        enablePan : true
    },
    legendSettings: { visible: false },
    title: "AAPL 2012-2017"
};
},
provide: {
    chart: [CandleSeries, Category, Tooltip, DateTime, Zoom, Logarithmic,
Crosshair, LineSeries, RsiIndicator, StripLine]
}
};
</script>
<style>
#container {
    height: 350px;
}
</style>

```

{% previewsample "page.domainurl/code-snippet/chart/technical-indicators/rsi-cs1" %}

### Simple Moving Average (SMA)

Moving average Indicators are used to define the direction of the trend. To render a SMA Indicator, use indicator [type](#) as `Sma` and inject `SmaIndicator` module using `provide`.

### APP.VUE

```

<template>
  <div id="app">
    <ejs-chart id="container" :title='title'
:primaryXAxis='primaryXAxis' :primaryYAxis='primaryYAxis'
      :chartArea='chartArea' :legendSettings='legendSettings'
:indicators='indicators' :crosshair='crosshair' :tooltip='tooltip'
:zoomSettings='zoomSettings' >

```

```

        <e-series-collection>
          <e-series :dataSource='seriesData1' type='Candle' xName='x'
yName='y' name='Apple Inc' width=2 low='low' high='high' close='close'
open='open' volume='volume' bearFillColor='#2ecd71' bullFillColor='#e74c3d'
:animation='animation'> </e-series>
        </e-series-collection>
      </ejs-chart>
    </div>
  </template>
</script>
import Vue from "vue";
import { ChartPlugin, Category, CandleSeries, Tooltip, DateTime, Zoom,
Crosshair, LineSeries, Logarithmic, SmaIndicator } from "@syncfusion/ej2-
vue-charts";
Vue.use(ChartPlugin);
let series1: Object[] = [
  {x: new Date('2012-10-15'), open: 90.3357, high: 93.2557, low:
87.0885,close: 87.12,volume: 646996264},
  {x: new Date('2012-10-22'), open: 87.4885, high: 90.7685, low:
84.4285,close: 86.2857,volume: 866040680 },
  {x: new Date('2012-10-29'), open: 84.9828, high: 86.1428, low:
82.1071,close: 82.4,volume: 367371310},
  {x: new Date('2012-11-05'), open: 83.3593, high: 84.3914, low:
76.2457,close: 78.1514,volume: 919719846},
  {x: new Date('2012-11-12'), open: 79.1643, high: 79.2143, low:
72.25,close: 75.3825,volume: 894382149},
  {x: new Date('2012-11-19'), open: 77.2443, high: 81.7143, low:
77.1257,close: 81.6428,volume: 527416747},
  {x: new Date('2012-11-26'), open: 82.2714, high: 84.8928, low:
81.7514,close: 83.6114,volume: 646467974},
  {x: new Date('2012-12-03'), open: 84.8071, high: 84.9414, low:
74.09,close: 76.1785,volume: 980096264},
  {x: new Date('2012-12-10'), open: 75, high: 78.5085, low: 72.2257,close:
72.8277,volume: 835016110},
  {x: new Date('2012-12-17'), open: 72.7043, high: 76.4143, low:
71.6043,close: 74.19,volume: 726150329},
  {x: new Date('2012-12-24'), open: 74.3357, high: 74.8928, low:
72.0943,close: 72.7984,volume: 321104733},
  {x: new Date('2012-12-31'), open: 72.9328, high: 79.2857, low:
72.7143,close: 75.2857,volume: 540854882},
  {x: new Date('2013-01-07'), open: 74.5714, high: 75.9843, low:
73.6,close: 74.3285,volume: 574594262},
  {x: new Date('2013-01-14'), open: 71.8114, high: 72.9643, low:
69.0543,close: 71.4285,volume: 803105621},
  {x: new Date('2013-01-21'), open: 72.08, high: 73.57, low:
62.1428,close: 62.84,volume: 971912560},
  {x: new Date('2013-01-28'), open: 62.5464, high: 66.0857, low:
62.2657,close: 64.8028,volume: 656549587},
  {x: new Date('2013-02-04'), open: 64.8443, high: 68.4014, low:
63.1428,close: 67.8543,volume: 743778993},
  {x: new Date('2013-02-11'), open: 68.0714, high: 69.2771, low:
65.7028,close: 65.7371,volume: 585292366},
  {x: new Date('2013-02-18'), open: 65.8714, high: 66.1043, low:
63.26,close: 64.4014,volume: 421766997},
  {x: new Date('2013-02-25'), open: 64.8357, high: 65.0171, low:
61.4257,close: 61.4957,volume: 582741215},

```

```
{x: new Date('2013-03-04'), open: 61.1143, high: 62.2043, low: 59.8571, close: 61.6743, volume: 632856539},
{x: new Date('2013-03-11'), open: 61.3928, high: 63.4614, low: 60.7343, close: 63.38, volume: 572066981},
{x: new Date('2013-03-18'), open: 63.0643, high: 66.0143, low: 63.0286, close: 65.9871, volume: 552156035},
{x: new Date('2013-03-25'), open: 66.3843, high: 67.1357, low: 63.0886, close: 63.2371, volume: 390762517},
{x: new Date('2013-04-01'), open: 63.1286, high: 63.3854, low: 59.9543, close: 60.4571, volume: 505273732},
{x: new Date('2013-04-08'), open: 60.6928, high: 62.57, low: 60.3557, close: 61.4, volume: 387323550},
{x: new Date('2013-04-15'), open: 61, high: 61.1271, low: 55.0143, close: 55.79, volume: 709945604},
{x: new Date('2013-04-22'), open: 56.0914, high: 59.8241, low: 55.8964, close: 59.6007, volume: 787007506},
{x: new Date('2013-04-29'), open: 60.0643, high: 64.7471, low: 60, close: 64.2828, volume: 655020017},
{x: new Date('2013-05-06'), open: 65.1014, high: 66.5357, low: 64.3543, close: 64.71, volume: 545488533},
{x: new Date('2013-05-13'), open: 64.5014, high: 65.4143, low: 59.8428, close: 61.8943, volume: 633706550},
{x: new Date('2013-05-20'), open: 61.7014, high: 64.05, low: 61.4428, close: 63.5928, volume: 494379068},
{x: new Date('2013-05-27'), open: 64.2714, high: 65.3, low: 62.7714, close: 64.2478, volume: 362907830},
{x: new Date('2013-06-03'), open: 64.39, high: 64.9186, low: 61.8243, close: 63.1158, volume: 443249793},
{x: new Date('2013-06-10'), open: 63.5328, high: 64.1541, low: 61.2143, close: 61.4357, volume: 389680092},
{x: new Date('2013-06-17'), open: 61.6343, high: 62.2428, low: 58.3, close: 59.0714, volume: 400384818},
{x: new Date('2013-06-24'), open: 58.2, high: 58.38, low: 55.5528, close: 56.6471, volume: 519314826},
{x: new Date('2013-07-01'), open: 57.5271, high: 60.47, low: 57.3171, close: 59.6314, volume: 343878841},
{x: new Date('2013-07-08'), open: 60.0157, high: 61.3986, low: 58.6257, close: 60.93, volume: 384106977},
{x: new Date('2013-07-15'), open: 60.7157, high: 62.1243, low: 60.5957, close: 60.7071, volume: 286035513},
{x: new Date('2013-07-22'), open: 61.3514, high: 63.5128, low: 59.8157, close: 62.9986, volume: 395816827},
{x: new Date('2013-07-29'), open: 62.9714, high: 66.1214, low: 62.8857, close: 66.0771, volume: 339668858},
{x: new Date('2013-08-12'), open: 65.2657, high: 72.0357, low: 65.2328, close: 71.7614, volume: 711563584},
{x: new Date('2013-08-19'), open: 72.0485, high: 73.3914, low: 71.1714, close: 71.5743, volume: 417119660},
{x: new Date('2013-08-26'), open: 71.5357, high: 72.8857, low: 69.4286, close: 69.6023, volume: 392805888},
{x: new Date('2013-09-02'), open: 70.4428, high: 71.7485, low: 69.6214, close: 71.1743, volume: 317244380},
{x: new Date('2013-09-09'), open: 72.1428, high: 72.56, low: 66.3857, close: 66.4143, volume: 669376320},
{x: new Date('2013-09-16'), open: 65.8571, high: 68.3643, low: 63.8886, close: 66.7728, volume: 625142677},
```



```

    {x: new Date('2013-09-23'), open: 70.8714, high: 70.9871, low:
68.6743,close: 68.9643,volume: 475274537},
    {x: new Date('2013-09-30'), open: 68.1786, high: 70.3357, low:
67.773,close: 69.0043,volume: 368198906},
    {x: new Date('2013-10-07'), open: 69.5086, high: 70.5486, low:
68.3257,close: 70.4017,volume: 361437661},
    {x: new Date('2013-10-14'), open: 69.9757, high: 72.7514, low:
69.9071,close: 72.6985,volume: 342694379},
    {x: new Date('2013-10-21'), open: 73.11, high: 76.1757, low:
72.5757,close: 75.1368,volume: 490458997},
    {x: new Date('2013-10-28'), open: 75.5771, high: 77.0357, low:
73.5057,close: 74.29,volume: 508130174},
    {x: new Date('2013-11-04'), open: 74.4428, high: 75.555, low:
73.1971,close: 74.3657,volume: 318132218},
    {x: new Date('2013-11-11'), open: 74.2843, high: 75.6114, low:
73.4871,close: 74.9987,volume: 306711021},
    {x: new Date('2013-11-18'), open: 74.9985, high: 75.3128, low:
73.3814,close: 74.2571,volume: 282778778},
];
export default {
  data() {
    return {
      seriesData1: series1,
      primaryXAxis: {
        valueType: 'DateTime',
        majorGridLines: { width: 0 },
        zoomFactor: 0.2, zoomPosition: 0.6,
        crosshairTooltip: { enable: true },
      },
      chartArea: {
        border: {
          width: 0
        }
      },
      //Initializing Primary Y Axis
      primaryYAxis: {
        title: 'Price',
        labelFormat: '${value}M',
        minimum: 50, maximum: 170, interval: 30,
        majorGridLines: { width: 1 },
        lineStyle: { width: 0 }
      },
      indicators: [{
        type: 'Sma', field: 'Close', seriesName: 'Apple Inc', fill:
'#6063ff',
        period: 14, animation: { enable: true }
      }],
      tooltip: {
        enable: true, shared: true
      },
      animation: { enable: false },
      crosshair: { enable: true, lineType: 'Vertical' },
      chartArea: { border: { width: 0 } },
      zoomSettings: {
        enableSelectionZooming: true,
        mode: 'X',
        enablePan : true
      }
    }
  }
}

```

```

    },
    legendSettings: { visible: false },
    title: "AAPL - 2012-2017"
  };
},
provide: {
  chart: [CandleSeries, Category, Tooltip, DateTime, Zoom, Logarithmic,
Crosshair, LineSeries, SmaIndicator]
}
};
</script>
<style>
  #container {
    height: 350px;
  }
</style>

```

{% previewsample "page.domainurl/code-snippet/chart/technical-indicators/sma-cs1" %}

### Stochastic

It shows how a stock is, when compared to previous state. To render a Stochastic indicator, use indicator [type](#) as Stochastic and inject StochasticIndicator module using provide method. stochastic indicator will be represented by four lines (upperLine, lowerLine, periodLine, signalLine). In stochastic indicator the upperBand value and lowerBand value is customized by [overBought](#) and [overBought](#) properties of indicators and the periodLine and

signalLine is render based on stochastic formula.

### APP.VUE

```

<template>
  <div id="app">
    <ejs-chart id="container" :title='title'
:primaryXAxis='primaryXAxis' :primaryYAxis='primaryYAxis'
      :chartArea='chartArea' :legendSettings='legendSettings'
:indicators='indicators' :crosshair='crosshair' :tooltip='tooltip'
:zoomSettings='zoomSettings' :axes='axes' :rows='rows'>
      <e-series-collection>
        <e-series :dataSource='seriesData1' type='Candle' xName='x'
yName='y' name='Apple Inc' width=2 low='low' high='high' close='close'
open='open' volume='volume' bearFillColor='#2ecd71' bullFillColor='#e74c3d'
:animation='animation'> </e-series>
      </e-series-collection>
    </ejs-chart>
  </div>
</template>
<script>
import Vue from "vue";
import { ChartPlugin, Category, CandleSeries, Tooltip, DateTime, Zoom,
Crosshair, LineSeries, Logarithmic, StripLine, StochasticIndicator } from
"@syncfusion/ej2-vue-charts";
Vue.use(ChartPlugin);
let series1: Object[] = [
  {x: new Date('2012-10-15'), open: 90.3357, high: 93.2557, low:
87.0885, close: 87.12, volume: 646996264},

```

```
{x: new Date('2012-10-22'), open: 87.4885, high: 90.7685, low: 84.4285, close: 86.2857, volume: 866040680 },
{x: new Date('2012-10-29'), open: 84.9828, high: 86.1428, low: 82.1071, close: 82.4, volume: 367371310},
{x: new Date('2012-11-05'), open: 83.3593, high: 84.3914, low: 76.2457, close: 78.1514, volume: 919719846},
{x: new Date('2012-11-12'), open: 79.1643, high: 79.2143, low: 72.25, close: 75.3825, volume: 894382149},
{x: new Date('2012-11-19'), open: 77.2443, high: 81.7143, low: 77.1257, close: 81.6428, volume: 527416747},
{x: new Date('2012-11-26'), open: 82.2714, high: 84.8928, low: 81.7514, close: 83.6114, volume: 646467974},
{x: new Date('2012-12-03'), open: 84.8071, high: 84.9414, low: 74.09, close: 76.1785, volume: 980096264},
{x: new Date('2012-12-10'), open: 75, high: 78.5085, low: 72.2257, close: 72.8277, volume: 835016110},
{x: new Date('2012-12-17'), open: 72.7043, high: 76.4143, low: 71.6043, close: 74.19, volume: 726150329},
{x: new Date('2012-12-24'), open: 74.3357, high: 74.8928, low: 72.0943, close: 72.7984, volume: 321104733},
{x: new Date('2012-12-31'), open: 72.9328, high: 79.2857, low: 72.7143, close: 75.2857, volume: 540854882},
{x: new Date('2013-01-07'), open: 74.5714, high: 75.9843, low: 73.6, close: 74.3285, volume: 574594262},
{x: new Date('2013-01-14'), open: 71.8114, high: 72.9643, low: 69.0543, close: 71.4285, volume: 803105621},
{x: new Date('2013-01-21'), open: 72.08, high: 73.57, low: 62.1428, close: 62.84, volume: 971912560},
{x: new Date('2013-01-28'), open: 62.5464, high: 66.0857, low: 62.2657, close: 64.8028, volume: 656549587},
{x: new Date('2013-02-04'), open: 64.8443, high: 68.4014, low: 63.1428, close: 67.8543, volume: 743778993},
{x: new Date('2013-02-11'), open: 68.0714, high: 69.2771, low: 65.7028, close: 65.7371, volume: 585292366},
{x: new Date('2013-02-18'), open: 65.8714, high: 66.1043, low: 63.26, close: 64.4014, volume: 421766997},
{x: new Date('2013-02-25'), open: 64.8357, high: 65.0171, low: 61.4257, close: 61.4957, volume: 582741215},
{x: new Date('2013-03-04'), open: 61.1143, high: 62.2043, low: 59.8571, close: 61.6743, volume: 632856539},
{x: new Date('2013-03-11'), open: 61.3928, high: 63.4614, low: 60.7343, close: 63.38, volume: 572066981},
{x: new Date('2013-03-18'), open: 63.0643, high: 66.0143, low: 63.0286, close: 65.9871, volume: 552156035},
{x: new Date('2013-03-25'), open: 66.3843, high: 67.1357, low: 63.0886, close: 63.2371, volume: 390762517},
{x: new Date('2013-04-01'), open: 63.1286, high: 63.3854, low: 59.9543, close: 60.4571, volume: 505273732},
{x: new Date('2013-04-08'), open: 60.6928, high: 62.57, low: 60.3557, close: 61.4, volume: 387323550},
{x: new Date('2013-04-15'), open: 61, high: 61.1271, low: 55.0143, close: 55.79, volume: 709945604},
{x: new Date('2013-04-22'), open: 56.0914, high: 59.8241, low: 55.8964, close: 59.6007, volume: 787007506},
{x: new Date('2013-04-29'), open: 60.0643, high: 64.7471, low: 60, close: 64.2828, volume: 655020017},
```

```

    {x: new Date('2013-05-06'), open: 65.1014, high: 66.5357, low:
64.3543,close: 64.71,volume: 545488533},
    {x: new Date('2013-05-13'), open: 64.5014, high: 65.4143, low:
59.8428,close: 61.8943,volume: 633706550},
    {x: new Date('2013-05-20'), open: 61.7014, high: 64.05, low:
61.4428,close: 63.5928,volume: 494379068},
    {x: new Date('2013-05-27'), open: 64.2714, high: 65.3, low:
62.7714,close: 64.2478,volume: 362907830},
    {x: new Date('2013-06-03'), open: 64.39, high: 64.9186, low:
61.8243,close: 63.1158,volume: 443249793},
    {x: new Date('2013-06-10'), open: 63.5328, high: 64.1541, low:
61.2143,close: 61.4357,volume: 389680092},
    {x: new Date('2013-06-17'), open: 61.6343, high: 62.2428, low:
58.3,close: 59.0714,volume: 400384818},
    {x: new Date('2013-06-24'), open: 58.2, high: 58.38, low: 55.5528,close:
56.6471,volume: 519314826},
    {x: new Date('2013-07-01'), open: 57.5271, high: 60.47, low:
57.3171,close: 59.6314,volume: 343878841},
    {x: new Date('2013-07-08'), open: 60.0157, high: 61.3986, low:
58.6257,close: 60.93,volume: 384106977},
    {x: new Date('2013-07-15'), open: 60.7157, high: 62.1243, low:
60.5957,close: 60.7071,volume: 286035513},
    {x: new Date('2013-07-22'), open: 61.3514, high: 63.5128, low:
59.8157,close: 62.9986,volume: 395816827},
    {x: new Date('2013-07-29'), open: 62.9714, high: 66.1214, low:
62.8857,close: 66.0771,volume: 339668858},
    {x: new Date('2013-08-12'), open: 65.2657, high: 72.0357, low:
65.2328,close: 71.7614,volume: 711563584},
    {x: new Date('2013-08-19'), open: 72.0485, high: 73.3914, low:
71.1714,close: 71.5743,volume: 417119660},
    {x: new Date('2013-08-26'), open: 71.5357, high: 72.8857, low:
69.4286,close: 69.6023,volume: 392805888},
    {x: new Date('2013-09-02'), open: 70.4428, high: 71.7485, low:
69.6214,close: 71.1743,volume: 317244380},
    {x: new Date('2013-09-09'), open: 72.1428, high: 72.56, low:
66.3857,close: 66.4143,volume: 669376320},
    {x: new Date('2013-09-16'), open: 65.8571, high: 68.3643, low:
63.8886,close: 66.7728,volume: 625142677},
    {x: new Date('2013-09-23'), open: 70.8714, high: 70.9871, low:
68.6743,close: 68.9643,volume: 475274537},
    {x: new Date('2013-09-30'), open: 68.1786, high: 70.3357, low:
67.773,close: 69.0043,volume: 368198906},
    {x: new Date('2013-10-07'), open: 69.5086, high: 70.5486, low:
68.3257,close: 70.4017,volume: 361437661},
    {x: new Date('2013-10-14'), open: 69.9757, high: 72.7514, low:
69.9071,close: 72.6985,volume: 342694379},
    {x: new Date('2013-10-21'), open: 73.11, high: 76.1757, low:
72.5757,close: 75.1368,volume: 490458997},
    {x: new Date('2013-10-28'), open: 75.5771, high: 77.0357, low:
73.5057,close: 74.29,volume: 508130174},
    {x: new Date('2013-11-04'), open: 74.4428, high: 75.555, low:
73.1971,close: 74.3657,volume: 318132218},
    {x: new Date('2013-11-11'), open: 74.2843, high: 75.6114, low:
73.4871,close: 74.9987,volume: 306711021},
    {x: new Date('2013-11-18'), open: 74.9985, high: 75.3128, low:
73.3814,close: 74.2571,volume: 282778778},
  ];

```

```

export default {
  data() {
    return {
      seriesData1: series1,
      primaryXAxis: {
        valueType: 'DateTime',
        majorGridLines: { width: 0 },
        zoomFactor: 0.2, zoomPosition: 0.6,
        crosshairTooltip: { enable: true },
      },
      //Initializing Primary Y Axis
      primaryYAxis: {
        title: 'Price',
        labelFormat: '${value}',
        minimum: 80, maximum: 170,
        plotOffset: 25,
        interval: 30, rowIndex: 1, opposedPosition: true, lineStyle: {
width: 0 }
      },
      rows: [
        {
          height: '40%'
        }, {
          height: '60%'
        }
      ],
      axes: [{
        name: 'secondary',
        opposedPosition: true, rowIndex: 0,
        majorGridLines: { width: 0 }, lineStyle: { width: 0 }, minimum:
0, maximum: 120, interval: 60,
        majorTickLines: { width: 0 }, title: 'Stochastic', stripLines: [
          {
            start: 0, end: 120, text: '', color: 'black', visible:
true,
            opacity:
0https://ej2.syncfusion.com/vue/documentation03, zIndex: 'Behind'
          }
        ]
      }],
      indicators: [{
        type: 'Stochastic', field: 'Close', seriesName: 'Apple Inc',
yAxisName: 'secondary', fill: '#6063ff',
        kPeriod: 2, dPeriod: 3, showZones: true, periodLine: { color:
'#f2ec2f' },
        period: 3, animation: { enable: false }, upperLine: { color:
'#e74c3d' }, lowerLine: { color: '#2ecd71' }
      }],
      tooltip: {
        enable: true, shared: true
      },
      animation: { enable: true },
      crosshair: { enable: true, lineType: 'Vertical' },
      chartArea: { border: { width: 0 } },
      zoomSettings:
      {
        enableSelectionZooming: true,
        mode: 'X',

```

```

        enablePan : true
      },
      legendSettings: { visible: false },
      title: "AAPL 2012-2017"
    };
  },
  provide: {
    chart: [CandleSeries, Category, Tooltip, DateTime, Zoom, Logarithmic,
    Crosshair, LineSeries, StochasticIndicator, StripLine]
  }
};
</script>
<style>
  #container {
    height: 350px;
  }
</style>

```

{% previewsample "page.domainurl/code-snippet/chart/technical-indicators/stochastic-cs1" %}

- Customization of StochasticIndicator

stroke, stroke-width, and color of upperLine can be customized by using [upperLine](#), the lowerLine can be customized by using [lowerLine](#) and the periodLine can be customized by using [periodLine](#) properties of indicator. To customize the period to find the properties.

#### APP.VUE

```

<template>
  <div id="app">
    <ejs-chart id="container" :title='title'
    :primaryXAxis='primaryXAxis' :primaryYAxis='primaryYAxis'
    :chartArea='chartArea' :legendSettings='legendSettings'
    :indicators='indicators' :crosshair='crosshair' :tooltip='tooltip'
    :zoomSettings='zoomSettings' :axes='axes' :rows='rows'>
      <e-series-collection>
        <e-series :dataSource='seriesData1' type='Candle' xName='x'
        yName='y' name='Apple Inc' width=2 low='low' high='high' close='close'
        open='open' volume='volume' bearFillColor='#2ecd71' bullFillColor='#e74c3d'
        :animation='animation'> </e-series>
      </e-series-collection>
    </ejs-chart>
  </div>
</template>
<script>
import Vue from "vue";
import { ChartPlugin, Category, CandleSeries, Tooltip, DateTime, Zoom,
Crosshair, LineSeries, Logarithmic, StripLine, StochasticIndicator } from
"@syncfusion/ej2-vue-charts";
Vue.use(ChartPlugin);
let series1: Object[] = [
  {x: new Date('2012-10-15'), open: 90.3357, high: 93.2557, low:
87.0885,close: 87.12,volume: 646996264},
  {x: new Date('2012-10-22'), open: 87.4885, high: 90.7685, low:
84.4285,close: 86.2857,volume: 866040680 },

```

```
{x: new Date('2012-10-29'), open: 84.9828, high: 86.1428, low: 82.1071, close: 82.4, volume: 367371310},
{x: new Date('2012-11-05'), open: 83.3593, high: 84.3914, low: 76.2457, close: 78.1514, volume: 919719846},
{x: new Date('2012-11-12'), open: 79.1643, high: 79.2143, low: 72.25, close: 75.3825, volume: 894382149},
{x: new Date('2012-11-19'), open: 77.2443, high: 81.7143, low: 77.1257, close: 81.6428, volume: 527416747},
{x: new Date('2012-11-26'), open: 82.2714, high: 84.8928, low: 81.7514, close: 83.6114, volume: 646467974},
{x: new Date('2012-12-03'), open: 84.8071, high: 84.9414, low: 74.09, close: 76.1785, volume: 980096264},
{x: new Date('2012-12-10'), open: 75, high: 78.5085, low: 72.2257, close: 72.8277, volume: 835016110},
{x: new Date('2012-12-17'), open: 72.7043, high: 76.4143, low: 71.6043, close: 74.19, volume: 726150329},
{x: new Date('2012-12-24'), open: 74.3357, high: 74.8928, low: 72.0943, close: 72.7984, volume: 321104733},
{x: new Date('2012-12-31'), open: 72.9328, high: 79.2857, low: 72.7143, close: 75.2857, volume: 540854882},
{x: new Date('2013-01-07'), open: 74.5714, high: 75.9843, low: 73.6, close: 74.3285, volume: 574594262},
{x: new Date('2013-01-14'), open: 71.8114, high: 72.9643, low: 69.0543, close: 71.4285, volume: 803105621},
{x: new Date('2013-01-21'), open: 72.08, high: 73.57, low: 62.1428, close: 62.84, volume: 971912560},
{x: new Date('2013-01-28'), open: 62.5464, high: 66.0857, low: 62.2657, close: 64.8028, volume: 656549587},
{x: new Date('2013-02-04'), open: 64.8443, high: 68.4014, low: 63.1428, close: 67.8543, volume: 743778993},
{x: new Date('2013-02-11'), open: 68.0714, high: 69.2771, low: 65.7028, close: 65.7371, volume: 585292366},
{x: new Date('2013-02-18'), open: 65.8714, high: 66.1043, low: 63.26, close: 64.4014, volume: 421766997},
{x: new Date('2013-02-25'), open: 64.8357, high: 65.0171, low: 61.4257, close: 61.4957, volume: 582741215},
{x: new Date('2013-03-04'), open: 61.1143, high: 62.2043, low: 59.8571, close: 61.6743, volume: 632856539},
{x: new Date('2013-03-11'), open: 61.3928, high: 63.4614, low: 60.7343, close: 63.38, volume: 572066981},
{x: new Date('2013-03-18'), open: 63.0643, high: 66.0143, low: 63.0286, close: 65.9871, volume: 552156035},
{x: new Date('2013-03-25'), open: 66.3843, high: 67.1357, low: 63.0886, close: 63.2371, volume: 390762517},
{x: new Date('2013-04-01'), open: 63.1286, high: 63.3854, low: 59.9543, close: 60.4571, volume: 505273732},
{x: new Date('2013-04-08'), open: 60.6928, high: 62.57, low: 60.3557, close: 61.4, volume: 387323550},
{x: new Date('2013-04-15'), open: 61, high: 61.1271, low: 55.0143, close: 55.79, volume: 709945604},
{x: new Date('2013-04-22'), open: 56.0914, high: 59.8241, low: 55.8964, close: 59.6007, volume: 787007506},
{x: new Date('2013-04-29'), open: 60.0643, high: 64.7471, low: 60, close: 64.2828, volume: 655020017},
{x: new Date('2013-05-06'), open: 65.1014, high: 66.5357, low: 64.3543, close: 64.71, volume: 545488533},
```

```

    {x: new Date('2013-05-13'), open: 64.5014, high: 65.4143, low:
59.8428,close: 61.8943,volume: 633706550},
    {x: new Date('2013-05-20'), open: 61.7014, high: 64.05, low:
61.4428,close: 63.5928,volume: 494379068},
    {x: new Date('2013-05-27'), open: 64.2714, high: 65.3, low:
62.7714,close: 64.2478,volume: 362907830},
    {x: new Date('2013-06-03'), open: 64.39, high: 64.9186, low:
61.8243,close: 63.1158,volume: 443249793},
    {x: new Date('2013-06-10'), open: 63.5328, high: 64.1541, low:
61.2143,close: 61.4357,volume: 389680092},
    {x: new Date('2013-06-17'), open: 61.6343, high: 62.2428, low:
58.3,close: 59.0714,volume: 400384818},
    {x: new Date('2013-06-24'), open: 58.2, high: 58.38, low: 55.5528,close:
56.6471,volume: 519314826},
    {x: new Date('2013-07-01'), open: 57.5271, high: 60.47, low:
57.3171,close: 59.6314,volume: 343878841},
    {x: new Date('2013-07-08'), open: 60.0157, high: 61.3986, low:
58.6257,close: 60.93,volume: 384106977},
    {x: new Date('2013-07-15'), open: 60.7157, high: 62.1243, low:
60.5957,close: 60.7071,volume: 286035513},
    {x: new Date('2013-07-22'), open: 61.3514, high: 63.5128, low:
59.8157,close: 62.9986,volume: 395816827},
    {x: new Date('2013-07-29'), open: 62.9714, high: 66.1214, low:
62.8857,close: 66.0771,volume: 339668858},
    {x: new Date('2013-08-12'), open: 65.2657, high: 72.0357, low:
65.2328,close: 71.7614,volume: 711563584},
    {x: new Date('2013-08-19'), open: 72.0485, high: 73.3914, low:
71.1714,close: 71.5743,volume: 417119660},
    {x: new Date('2013-08-26'), open: 71.5357, high: 72.8857, low:
69.4286,close: 69.6023,volume: 392805888},
    {x: new Date('2013-09-02'), open: 70.4428, high: 71.7485, low:
69.6214,close: 71.1743,volume: 317244380},
    {x: new Date('2013-09-09'), open: 72.1428, high: 72.56, low:
66.3857,close: 66.4143,volume: 669376320},
    {x: new Date('2013-09-16'), open: 65.8571, high: 68.3643, low:
63.8886,close: 66.7728,volume: 625142677},
    {x: new Date('2013-09-23'), open: 70.8714, high: 70.9871, low:
68.6743,close: 68.9643,volume: 475274537},
    {x: new Date('2013-09-30'), open: 68.1786, high: 70.3357, low:
67.773,close: 69.0043,volume: 368198906},
    {x: new Date('2013-10-07'), open: 69.5086, high: 70.5486, low:
68.3257,close: 70.4017,volume: 361437661},
    {x: new Date('2013-10-14'), open: 69.9757, high: 72.7514, low:
69.9071,close: 72.6985,volume: 342694379},
    {x: new Date('2013-10-21'), open: 73.11, high: 76.1757, low:
72.5757,close: 75.1368,volume: 490458997},
    {x: new Date('2013-10-28'), open: 75.5771, high: 77.0357, low:
73.5057,close: 74.29,volume: 508130174},
    {x: new Date('2013-11-04'), open: 74.4428, high: 75.555, low:
73.1971,close: 74.3657,volume: 318132218},
    {x: new Date('2013-11-11'), open: 74.2843, high: 75.6114, low:
73.4871,close: 74.9987,volume: 306711021},
    {x: new Date('2013-11-18'), open: 74.9985, high: 75.3128, low:
73.3814,close: 74.2571,volume: 282778778},
  ];
export default {
  data() {

```



```

return {
  seriesData1: series1,
  primaryXAxis: {
    valueType: 'DateTime',
    majorGridLines: { width: 0 },
    zoomFactor: 0.2, zoomPosition: 0.6,
    crosshairTooltip: { enable: true },
  },
  //Initializing Primary Y Axis
  primaryYAxis: {
    title: 'Price',
    labelFormat: '${value}',
    minimum: 80, maximum: 170,
    plotOffset: 25,
    interval: 30, rowIndex: 1, opposedPosition: true, lineStyle: {
width: 0 }
  },
  rows: [
    {
      height: '40%'
    }, {
      height: '60%'
    }
  ],
  axes: [{
    name: 'secondary',
    opposedPosition: true, rowIndex: 0,
    majorGridLines: { width: 0 }, lineStyle: { width: 0 }, minimum:
0, maximum: 120, interval: 60,
    majorTickLines: { width: 0 }, title: 'Stochastic', stripLines: [
      {
        start: 0, end: 120, text: '', color: 'black', visible:
true,
        opacity: 0.03, zIndex: 'Behind'
      }
    ]
  }],
  indicators: [{
    type: 'Stochastic', field: 'Close', seriesName: 'Apple Inc',
yAxisName: 'secondary', fill: '#6063ff',
    kPeriod: 2, dPeriod: 3, showZones: true, periodLine: { color:
'#f2ec2f' },
    period: 3, animation: { enable: false }, upperLine: { color:
'#e74c3d' }, lowerLine: { color: '#2ecd71' }
  }],
  tooltip: {
    enable: true, shared: true
  },
  animation: { enable: true },
  crosshair: { enable: true, lineType: 'Vertical' },
  chartArea: { border: { width: 0 } },
  zoomSettings:
  {
    enableSelectionZooming: true,
    mode: 'X',
    enablePan : true
  },
  legendSettings: { visible: false },

```

```

        title: "AAPL 2012-2017"
      };
    },
    provide: {
      chart: [CandleSeries, Category, Tooltip, DateTime, Zoom, Logarithmic,
Crosshair, LineSeries, StochasticIndicator, StripLine]
    }
  };
</script>
<style>
#container {
  height: 350px;
}
</style>

```

{% previewsample "page.domainurl/code-snippet/chart/technical-indicators/stochastic-cs2" %}

### Trend lines in Vue Chart component

Trendlines are used to show the direction and speed of price.

Trendlines can be generated for Cartesian type series (Line, Column, Scatter, Area, Candle, Hilo etc.)

except bar type series. You can add more than one trendline to a series.

Chart supports 6 types of trendlines.

#### Linear

A linear trendline is a best fit straight line that is used with simpler data sets. To render a linear trendline, use trendline [type](#) as `Linear` and inject `TrendLines` module using `provide`.

#### APP.VUE

```

<template>
  <div id="app">
    <ejs-chart id="container" :primaryXAxis='primaryXAxis'
:primaryYAxis='primaryYAxis'>
      <e-series-collection>
        <e-series :dataSource='seriesData' type='Scatter' xName='x'
yName='y'>
          <e-trendlines>
            <e-trendline :type='type'>
            </e-trendline>
          </e-trendlines>
        </e-series>
      </e-series-collection>
    </ejs-chart>
  </div>
</template>
<script>
import Vue from "vue";
import { ChartPlugin, ScatterSeries, Trendlines, LineSeries } from
"@syncfusion/ej2-vue-charts";
Vue.use(ChartPlugin);
let series1 = [];
let yValue = [7.66, 8.03, 8.41, 8.97, 8.77, 8.20, 8.16, 7.89, 8.68, 9.48,
10.11, 11.36, 12.34, 12.60, 12.95,
13.91, 16.21, 17.50, 22.72, 28.14, 31.26, 31.39, 32.43, 35.52, 36.36,

```

```

    41.33, 43.12, 45.00, 47.23, 48.62, 46.60, 45.28, 44.01, 45.17, 41.20,
    43.41, 48.32, 45.65, 46.61, 53.34, 58.53];
    let point1; let i; let j = 0;
    for (i = 1973; i <= 2013; i++) {
        point1 = { x: i, y: yValue[j] };
        series1.push(point1); j++;
    }
    export default {
        data() {
            return {
                seriesData: series1,
                primaryXAxis: {
                    title: 'Months',
                },
                primaryYAxis: {
                    title: 'Rupees against Dollars',
                    interval: 5
                },
                type: 'Linear'
            };
        },
        provide: {
            chart: [ScatterSeries, Trendlines, LineSeries]
        }
    };
</script>
<style>
    #container{
        height: 350px;
    }
</style>

```

{% previewsample "page.domainurl/code-snippet/chart/trendlines/linear-cs1" %}

### Exponential

An exponential trendline is a curved line that is most useful when data values rise or fall at increasingly higher rates. You cannot create an exponential trendline, if your data contains zero or negative values.

To render a exponential trendline, use trendline [type](#) as **Exponential** and inject **TrendLines** module using **provide**.

### APP.VUE

```

<template>
    <div id="app">
        <ejs-chart id="container" :title='title'
        :primaryXAxis='primaryXAxis' :primaryYAxis='primaryYAxis'>
            <e-series-collection>
                <e-series :dataSource='seriesData' type='Scatter' xName='x'
                yName='y'>
                    <e-trendlines>
                        <e-trendline :type='type'>
                        </e-trendline>
                    </e-trendlines>
                </e-series>
            </e-series-collection>

```

```

        </ejs-chart>
    </div>
</template>
<script>
import Vue from "vue";
import { ChartPlugin, ScatterSeries, Trendlines, SplineSeries, LineSeries }
from "@syncfusion/ej2-vue-charts";
Vue.use(ChartPlugin);
let series1 = [];
let yValue = [7.66, 8.03, 8.41, 8.97, 8.77, 8.20, 8.16, 7.89, 8.68, 9.48,
10.11, 11.36, 12.34, 12.60, 12.95,
13.91, 16.21, 17.50, 22.72, 28.14, 31.26, 31.39, 32.43, 35.52, 36.36,
41.33, 43.12, 45.00, 47.23, 48.62, 46.60, 45.28, 44.01, 45.17, 41.20,
43.41, 48.32, 45.65, 46.61, 53.34, 58.53];
let point1; let i; let j = 0;
for (i = 1973; i <= 2013; i++) {
    point1 = { x: i, y: yValue[j] };
    series1.push(point1); j++;
}
export default {
    data() {
        return {
            seriesData: series1,
            primaryXAxis: {
                title: 'Months',
            },
            primaryYAxis: {
                title: 'Rupees against Dollars',
                interval: 5
            },
            type: 'Exponential',
            title: 'Historical Indian Rupee Rate (INR USD)'
        };
    },
    provide: {
        chart: [ScatterSeries, Trendlines, SplineSeries, LineSeries]
    }
};
</script>
<style>
    #container{
        height: 350px;
    }
</style>

```

{% previewsample "page.domainurl/code-snippet/chart/trendlines/exponential-cs1" %}

### Logarithmic

A logarithmic trendline is a best-fit curved line that is most useful when the rate of change in the data increases or decreases quickly and then levels out.

A logarithmic trendline can use negative and/or positive values.

To render a logarithmic trendline, use trendline [type](#) as `Logarithmic` and inject `TrendLines` module using `provide`.

**APP.VUE**

```

<template>
  <div id="app">
    <ejs-chart id="container" :title='title'
:primaryXAxis='primaryXAxis' :primaryYAxis='primaryYAxis'>
      <e-series-collection>
        <e-series :dataSource='seriesData' type='Scatter' xName='x'
yName='y'>
          <e-trendlines>
            <e-trendline :type='type'>
            </e-trendline>
          </e-trendlines>
        </e-series>
      </e-series-collection>
    </ejs-chart>
  </div>
</template>
<script>
import Vue from "vue";
import { ChartPlugin, ScatterSeries, SplineSeries, Trendlines, LineSeries }
from "@syncfusion/ej2-vue-charts";
Vue.use(ChartPlugin);
let series1 = [];
let yValue = [7.66, 8.03, 8.41, 8.97, 8.77, 8.20, 8.16, 7.89, 8.68, 9.48,
10.11, 11.36, 12.34, 12.60, 12.95,
13.91, 16.21, 17.50, 22.72, 28.14, 31.26, 31.39, 32.43, 35.52, 36.36,
41.33, 43.12, 45.00, 47.23, 48.62, 46.60, 45.28, 44.01, 45.17, 41.20,
43.41, 48.32, 45.65, 46.61, 53.34, 58.53];
let point1; let i; let j = 0;
for (i = 1973; i <= 2013; i++) {
  point1 = { x: i, y: yValue[j] };
  series1.push(point1); j++;
}
export default {
  data() {
    return {
      seriesData: series1,
      primaryXAxis: {
        title: 'Months',
      },
      primaryYAxis: {
        title: 'Rupees against Dollars',
        interval: 5
      },
      type: 'Logarithmic',
      title: 'Historical Indian Rupee Rate (INR USD) '
    };
  },
  provide: {
    chart: [ScatterSeries, SplineSeries, Trendlines, LineSeries]
  }
};
</script>
<style>
#container{
  height: 350px;

```

```
}
</style>
```

{% previewsample "page.domainurl/code-snippet/chart/trendlines/logarithmic-cs1" %}

### Polynomial

A polynomial trendline is a curved line that is used when data fluctuates.

To render a polynomial trendline, use trendline [type](#) as **Polynomial** and inject **TrendLines** module using **provide**.

**polynomialOrder** used to define the polynomial value.

### APP.VUE

```
<template>
  <div id="app">
    <ejs-chart id="container" :title='title'
:primaryXAxis='primaryXAxis' :primaryYAxis='primaryYAxis'>
      <e-series-collection>
        <e-series :dataSource='seriesData' type='Scatter' xName='x'
yName='y'>
          <e-trendlines>
            <e-trendline :type='type'>
            </e-trendline>
          </e-trendlines>
        </e-series>
      </e-series-collection>
    </ejs-chart>
  </div>
</template>
<script>
import Vue from "vue";
import { ChartPlugin, ScatterSeries, SplineSeries, Trendlines, LineSeries }
from "@syncfusion/ej2-vue-charts";
Vue.use(ChartPlugin);
let series1 = [];
let yValue = [7.66, 8.03, 8.41, 8.97, 8.77, 8.20, 8.16, 7.89, 8.68, 9.48,
10.11, 11.36, 12.34, 12.60, 12.95,
13.91, 16.21, 17.50, 22.72, 28.14, 31.26, 31.39, 32.43, 35.52, 36.36,
41.33, 43.12, 45.00, 47.23, 48.62, 46.60, 45.28, 44.01, 45.17, 41.20,
43.41, 48.32, 45.65, 46.61, 53.34, 58.53];
let point1; let i; let j = 0;
for (i = 1973; i <= 2013; i++) {
  point1 = { x: i, y: yValue[j] };
  series1.push(point1); j++;
}
export default {
  data() {
    return {
      seriesData: series1,
      primaryXAxis: {
        title: 'Months',
      },
      primaryYAxis: {
        title: 'Rupees against Dollars',

```

```

        interval: 5
      },
      type: 'Polynomial',
      title: 'Historical Indian Rupee Rate (INR USD)'
    };
  },
  provide: {
    chart: [ScatterSeries, Trendlines, SplineSeries, LineSeries]
  }
};
</script>
<style>
  #container{
    height: 350px;
  }
</style>

```

{% previewsample "page.domainurl/code-snippet/chart/trendlines/polynomial-cs1" %}

### Power

A power trendline is a curved line that is best used with data sets that compare measurements that increase at a specific rate.

To render a power trendline, use trendline [type](#) as **Power** and inject

**TrendLines**.

### APP.VUE

```

<template>
  <div id="app">
    <ejs-chart id="container" :title='title'
:primaryXAxis='primaryXAxis' :primaryYAxis='primaryYAxis'>
      <e-series-collection>
        <e-series :dataSource='seriesData' type='Scatter' xName='x'
yName='y'>
          <e-trendlines>
            <e-trendline :type='type'>
            </e-trendline>
          </e-trendlines>
        </e-series>
      </e-series-collection>
    </ejs-chart>
  </div>
</template>
<script>
import Vue from "vue";
import { ChartPlugin, ScatterSeries, SplineSeries, Trendlines, LineSeries }
from "@syncfusion/ej2-vue-charts";
Vue.use(ChartPlugin);
export default {
  data() {
    return {
      seriesData: [
        { x: 1, y: 10 }, { x: 2, y: 50 }, { x: 3, y: 80 }, { x: 4, y: 110 },
        { x: 5, y: 180 }, { x: 6, y: 220 }, { x: 7, y: 300 }, { x: 8, y: 370
}, { x: 9, y: 490 }, { x: 10, y: 500 }

```

```

    ],
    primaryXAxis: {
      title: 'Months',
    },
    primaryYAxis: {
      title: 'Rupees against Dollars',
      interval: 100
    },
    type: 'Power',
    title: 'Historical Indian Rupee Rate (INR USD)'
  };
},
provide: {
  chart: [ScatterSeries, SplineSeries, Trendlines, LineSeries]
}
};
</script>
<style>
#container{
  height: 350px;
}
</style>

```

{% previewsample "page.domainurl/code-snippet/chart/trendlines/power-cs1" %}

### Moving Average

A moving average trendline smoothen out fluctuations in data to show a pattern or trend more clearly.

To render a moving average trendline, use trendline [type](#) as `MovingAverage` and inject `TrendLines` module using `provide`.

`period` property defines the period to find the moving average.

### APP.VUE

```

<template>
  <div id="app">
    <ejs-chart id="container" :title='title'
:primaryXAxis='primaryXAxis' :primaryYAxis='primaryYAxis'>
      <e-series-collection>
        <e-series :dataSource='seriesData' type='Scatter' xName='x'
yName='y'>
          <e-trendlines>
            <e-trendline :type='type'>
            </e-trendline>
          </e-trendlines>
        </e-series>
      </e-series-collection>
    </ejs-chart>
  </div>
</template>
<script>
import Vue from "vue";
import { ChartPlugin, ScatterSeries, Trendlines, LineSeries } from
"@syncfusion/ej2-vue-charts";
Vue.use(ChartPlugin);

```



```

let series1 = [];
let yValue = [7.66, 8.03, 8.41, 8.97, 8.77, 8.20, 8.16, 7.89, 8.68, 9.48,
10.11, 11.36, 12.34, 12.60, 12.95,
13.91, 16.21, 17.50, 22.72, 28.14, 31.26, 31.39, 32.43, 35.52, 36.36,
41.33, 43.12, 45.00, 47.23, 48.62, 46.60, 45.28, 44.01, 45.17, 41.20,
43.41, 48.32, 45.65, 46.61, 53.34, 58.53];
let point1; let i; let j = 0;
for (i = 1973; i <= 2013; i++) {
    point1 = { x: i, y: yValue[j] };
    series1.push(point1); j++;
}
export default {
    data() {
        return {
            seriesData: series1,
            primaryXAxis: {
                title: 'Months',
            },
            primaryYAxis: {
                title: 'Rupees against Dollars',
                interval: 5
            },
            type: 'MovingAverage',
            title: 'Historical Indian Rupee Rate (INR USD)'
        };
    },
    provide: {
        chart: [ScatterSeries, Trendlines, LineSeries]
    }
};
</script>
<style>
    #container{
        height: 350px;
    }
</style>

```

{% previewsample "page.domainurl/code-snippet/chart/trendlines/power-cs2" %}

- Customization of Trendlines

The [fill](#) and [width](#) properties are used to customize the appearance of the trendline.

#### APP.VUE

```

<template>
    <div id="app">
        <ejs-chart id="container" :title='title'
:primaryXAxis='primaryXAxis' :primaryYAxis='primaryYAxis'>
            <e-series-collection>
                <e-series :dataSource='seriesData' type='Scatter' xName='x'
yName='y' fill='#0066FF'>
                    <e-trendlines>
                        <e-trendline :type='type'>
                        </e-trendline>
                    </e-trendlines>
                </e-series>
            </e-series-collection>
        </ejs-chart>
    </div>

```

```

        </e-series>
      </e-series-collection>
    </ejs-chart>
  </div>
</template>
<script>
import Vue from "vue";
import { ChartPlugin, ScatterSeries, Trendlines, LineSeries } from
"@syncfusion/ej2-vue-charts";
Vue.use(ChartPlugin);
let series1 = [];
let yValue = [7.66, 8.03, 8.41, 8.97, 8.77, 8.20, 8.16, 7.89, 8.68, 9.48,
10.11, 11.36, 12.34, 12.60, 12.95,
13.91, 16.21, 17.50, 22.72, 28.14, 31.26, 31.39, 32.43, 35.52, 36.36,
41.33, 43.12, 45.00, 47.23, 48.62, 46.60, 45.28, 44.01, 45.17, 41.20,
43.41, 48.32, 45.65, 46.61, 53.34, 58.53];
let point1;
let i;
let j = 0;
for (i = 1973; i <= 2013; i++) {
  point1 = { x: i, y: yValue[j] };
  series1.push(point1); j++;
}
export default {
  data() {
    return {
      seriesData: series1,
      primaryXAxis: {
        title: 'Months',
      },
      primaryYAxis: {
        title: 'Rupees against Dollars',
        interval: 5
      },
      type: 'MovingAverage',
      title: 'Historical Indian Rupee Rate (INR USD)'
    };
  },
  provide: {
    chart: [ScatterSeries, Trendlines, LineSeries]
  }
};
</script>
<style>
#container{
  height: 350px;
}
</style>

```

{% previewsample "page.domainurl/code-snippet/chart/trendlines/power-cs3" %}

### Forecasting

Trendlines forecasting is the prediction of future/past situations.

Forecasting can be classified into two types as follows

## Forward Forecasting

## Backward Forecasting

- Forward Forecasting

The value set for forwardForecast is used to determine the distance moving towards the future trend.

**APP.VUE**

```

<template>
  <div id="app">
    <ejs-chart id="container" :title='title'
:primaryXAxis='primaryXAxis' :primaryYAxis='primaryYAxis'>
      <e-series-collection>
        <e-series :dataSource='seriesData' type='Scatter' xName='x'
yName='y'>
          <e-trendlines>
            <e-trendline :type='type'
:forwardForecast='forwardForecast'>
              </e-trendline>
            </e-trendlines>
          </e-series>
        </e-series-collection>
      </ejs-chart>
    </div>
  </template>
<script>
import Vue from "vue";
import { ChartPlugin, ScatterSeries, Trendlines, LineSeries } from
"@syncfusion/ej2-vue-charts";
Vue.use(ChartPlugin);
let series1 = [];
let yValue = [7.66, 8.03, 8.41, 8.97, 8.77, 8.20, 8.16, 7.89, 8.68, 9.48,
10.11, 11.36, 12.34, 12.60, 12.95,
13.91, 16.21, 17.50, 22.72, 28.14, 31.26, 31.39, 32.43, 35.52, 36.36,
41.33, 43.12, 45.00, 47.23, 48.62, 46.60, 45.28, 44.01, 45.17, 41.20,
43.41, 48.32, 45.65, 46.61, 53.34, 58.53];
let point1; let i; let j = 0;
for (i = 1973; i <= 2013; i++) {
  point1 = { x: i, y: yValue[j] };
  series1.push(point1); j++;
}
export default {
  data() {
    return {
      seriesData: series1,
      primaryXAxis: {
        title: 'Months'
      },
      primaryYAxis: {
        title: 'Rupees against Dollars',
        interval: 5
      },
      forwardForecast:5,
      type: 'MovingAverage',
      title: 'Historical Indian Rupee Rate (INR USD)'
    }
  }
}

```

```

    };
  },
  provide: {
    chart: [ScatterSeries, Trendlines, LineSeries]
  }
};
</script>
<style>
  #container{
    height: 350px;
  }
</style>

```

{% previewsample "page.domainurl/code-snippet/chart/trendlines/power-cs4" %}

- Backward Forecasting

The value set for the backwardForecast is used to determine the past trends.

### APP.VUE

```

<template>
  <div id="app">
    <ejs-chart id="container" :title='title'
:primaryXAxis='primaryXAxis' :primaryYAxis='primaryYAxis'>
      <e-series-collection>
        <e-series :dataSource='seriesData' type='Scatter' xName='x'
yName='y'>
          <e-trendlines>
            <e-trendline :type='type'
:backwardForecast='backwardForecast'>
              </e-trendline>
            </e-trendlines>
          </e-series>
        </e-series-collection>
      </ejs-chart>
    </div>
  </template>
<script>
import Vue from "vue";
import { ChartPlugin, ScatterSeries, Trendlines, LineSeries } from
"@syncfusion/ej2-vue-charts";
Vue.use(ChartPlugin);
let series1 = [];
let yValue = [7.66, 8.03, 8.41, 8.97, 8.77, 8.20, 8.16, 7.89, 8.68, 9.48,
10.11, 11.36, 12.34, 12.60, 12.95,
13.91, 16.21, 17.50, 22.72, 28.14, 31.26, 31.39, 32.43, 35.52, 36.36,
41.33, 43.12, 45.00, 47.23, 48.62, 46.60, 45.28, 44.01, 45.17, 41.20,
43.41, 48.32, 45.65, 46.61, 53.34, 58.53];
let point1; let i; let j = 0;
for (i = 1973; i <= 2013; i++) {
  point1 = { x: i, y: yValue[j] };
  series1.push(point1); j++;
}
export default {
  data() {

```

```

    return {
      seriesData: series1,
      primaryXAxis: {
        title: 'Months',
      },
      primaryYAxis: {
        title: 'Rupees against Dollars',
        interval: 5
      },
      backwardForecast: 5,
      type: 'MovingAverage',
      title: 'Historical Indian Rupee Rate (INR USD)'
    };
  },
  provide: {
    chart: [ScatterSeries, Trendlines, LineSeries]
  }
};
</script>
<style>
  #container{
    height: 350px;
  }
</style>

```

{% previewsample "page.domainurl/code-snippet/chart/trendlines/power-cs5" %}

Show or hide a trendline

You can show or hide the trendline by setting trendline **visible** property.

#### APP.VUE

```

<template>
  <div id="app">
    <ejs-chart id="container" :primaryXAxis='primaryXAxis'
:primaryYAxis='primaryYAxis'>
      <e-series-collection>
        <e-series :dataSource='seriesData' type='Scatter' xName='x'
yName='y'>
          <e-trendlines>
            <e-trendline :type='type' :visible=false>
            </e-trendline>
          </e-trendlines>
        </e-series>
      </e-series-collection>
    </ejs-chart>
  </div>
</template>
<script>
import Vue from "vue";
import { ChartPlugin, ScatterSeries, Trendlines, LineSeries } from
"@syncfusion/ej2-vue-charts";
Vue.use(ChartPlugin);
let series1 = [];
let yValue = [7.66, 8.03, 8.41, 8.97, 8.77, 8.20, 8.16, 7.89, 8.68, 9.48,
10.11, 11.36, 12.34, 12.60, 12.95,

```

```

    13.91, 16.21, 17.50, 22.72, 28.14, 31.26, 31.39, 32.43, 35.52, 36.36,
    41.33, 43.12, 45.00, 47.23, 48.62, 46.60, 45.28, 44.01, 45.17, 41.20,
    43.41, 48.32, 45.65, 46.61, 53.34, 58.53];
let point1; let i; let j = 0;
for (i = 1973; i <= 2013; i++) {
    point1 = { x: i, y: yValue[j] };
    series1.push(point1); j++;
}
export default {
  data() {
    return {
      seriesData: series1,
      primaryXAxis: {
        title: 'Months',
      },
      primaryYAxis: {
        title: 'Rupees against Dollars',
        interval: 5
      },
      type: 'Linear'
    };
  },
  provide: {
    chart: [ScatterSeries, Trendlines, LineSeries]
  }
};
</script>
<style>
#container{
  height: 350px;
}
</style>

```

{% previewsample "page.domainurl/code-snippet/chart/trendlines/linear-cs2" %}

## Data markers in Vue Chart component

Data markers are used to provide information about the data points in the series. You can add a shape to adorn

each data point.

<!-- markdownlint-disable MD036 -->

### Marker

<!-- markdownlint-disable MD036 -->

Markers can be added to points by enabling the [visible](#) option of the marker property. By default, distinct markers will be enabled for each series in the chart.

### APP.VUE

```

<template>
  <div id="app">
    <ejs-chart id="container" :title='title'
    :primaryXAxis='primaryXAxis'>
      <e-series-collection>

```

```

        <e-series :dataSource='seriesData' type='Line' xName='x'
yName='y' name='December 2007'
        :marker='marker'> </e-series>
        <e-series :dataSource='seriesData' type='Line' xName='x'
yName='y1' name='December 2008'
        :marker='marker'> </e-series>
    </e-series-collection>
</ejs-chart>
</div>
</template>
<script>
import Vue from "vue";
import { ChartPlugin, LineSeries, Category } from "@syncfusion/ej2-vue-
charts";
Vue.use(ChartPlugin);
export default {
    data() {
        return {
            seriesData: [
                { x: 'WW', y: 12, y1: 22, y2: 38.3, y3: 50 },
                { x: 'EU', y: 9.9, y1: 26, y2: 45.2, y3: 63.6 },
                { x: 'APAC', y: 4.4, y1: 9.3, y2: 18.2, y3: 20.9 },
                { x: 'LATAM', y: 6.4, y1: 28, y2: 46.7, y3: 65.1 },
                { x: 'MEA', y: 30, y1: 45.7, y2: 61.5, y3: 73 },
                { x: 'NA', y: 25.3, y1: 35.9, y2: 64, y3: 81.4 }
            ],
            primaryXAxis: {
                valueType: 'Category', interval: 1
            },
            marker: {
                visible: true,
            },
            title: "FB Penetration of Internet Audience"
        };
    },
    provide: {
        chart: [LineSeries, Category]
    }
};
</script>
<style>
    #container {
        height: 350px;
    }
</style>

```

{% previewsample "page.domainurl/code-snippet/chart/data-marker/marker-cs2" %}

### Shape

Markers can be assigned with different shapes such as Rectangle, Circle, Diamond, etc. using the [shape](#) property.

### APP.VUE

```

<template>
    <div id="app">

```

```

        <ejs-chart id="container" :title='title'
:primaryXAxis='primaryXAxis'>
            <e-series-collection>
                <e-series :dataSource='seriesData' type='Line' xName='x'
yName='y' name='December 2007'
                :marker='marker'> </e-series>
            </e-series-collection>
        </ejs-chart>
    </div>
</template>
<script>
import Vue from "vue";
import { ChartPlugin, LineSeries, Category } from "@syncfusion/ej2-vue-
charts";
Vue.use(ChartPlugin);
export default {
    data() {
        return {
            seriesData: [
                { x: 'WW', y: 12, y1: 22, y2: 38.3, y3: 50 },
                { x: 'EU', y: 9.9, y1: 26, y2: 45.2, y3: 63.6 },
                { x: 'APAC', y: 4.4, y1: 9.3, y2: 18.2, y3: 20.9 },
                { x: 'LATAM', y: 6.4, y1: 28, y2: 46.7, y3: 65.1 },
                { x: 'MEA', y: 30, y1: 45.7, y2: 61.5, y3: 73 },
                { x: 'NA', y: 25.3, y1: 35.9, y2: 64, y3: 81.4 }
            ],
            primaryXAxis: {
                valueType: 'Category', interval: 1
            },
            marker: {
                visible: true, width: 10, height: 10, shape: 'Diamond'
            },
            title: "FB Penetration of Internet Audience"
        };
    },
    provide: {
        chart: [LineSeries, Category]
    }
};
</script>
<style>
    #container {
        height: 350px;
    }
</style>

```

{% previewsample "page.domainurl/code-snippet/chart/data-marker/marker-cs3" %}

Note : To know more about the marker shape type refer the [shape](#).

### Images

Apart from the shapes, you can also add custom images to mark the data point using the

[imageUrl](#) property.

### APP.VUE



```

<template>
  <div id="app">
    <ejs-chart id="container" :title='title'
:primaryXAxis='primaryXAxis'>
      <e-series-collection>
        <e-series :dataSource='seriesData' type='Line' xName='x'
yName='y' name='December 2007'
          :marker='marker'> </e-series>
      </e-series-collection>
    </ejs-chart>
  </div>
</template>
<script>
import Vue from "vue";
import { ChartPlugin, LineSeries, Category } from "@syncfusion/ej2-vue-
charts";
Vue.use(ChartPlugin);
export default {
  data() {
    return {
      seriesData: [
        { x: 'WW', y: 12, y1: 22, y2: 38.3, y3: 50 },
        { x: 'EU', y: 9.9, y1: 26, y2: 45.2, y3: 63.6 },
        { x: 'APAC', y: 4.4, y1: 9.3, y2: 18.2, y3: 20.9 },
        { x: 'LATAM', y: 6.4, y1: 28, y2: 46.7, y3: 65.1 },
        { x: 'MEA', y: 30, y1: 45.7, y2: 61.5, y3: 73 },
        { x: 'NA', y: 25.3, y1: 35.9, y2: 64, y3: 81.4 }
      ],
      primaryXAxis: {
        valueType: 'Category', interval: 1
      },
      marker: {
        visible: true,
        width: 10, height: 10, shape: 'Image',
        imageUrl: './sun_annotation.png'
      },
      title: "FB Penetration of Internet Audience"
    };
  },
  provide: {
    chart: [LineSeries, Category]
  }
};
</script>
<style>
#container {
  height: 350px;
}
</style>

```

{% previewsample "page.domainurl/code-snippet/chart/data-marker/marker-cs4" %}

### Customization

Marker's color and border can be customized using `fill` and `border` properties.

### APP.VUE

```

<template>
  <div id="app">
    <ejs-chart id="container" :title='title'
:primaryXAxis='primaryXAxis'>
      <e-series-collection>
        <e-series :dataSource='seriesData' type='Line' xName='x'
yName='y' name='December 2007'
          :marker='marker'> </e-series>
      </e-series-collection>
    </ejs-chart>
  </div>
</template>
<script>
import Vue from "vue";
import { ChartPlugin, LineSeries, Category } from "@syncfusion/ej2-vue-
charts";
Vue.use(ChartPlugin);
export default {
  data() {
    return {
      seriesData: [
        { x: 'WW', y: 12, y1: 22, y2: 38.3, y3: 50 },
        { x: 'EU', y: 9.9, y1: 26, y2: 45.2, y3: 63.6 },
        { x: 'APAC', y: 4.4, y1: 9.3, y2: 18.2, y3: 20.9 },
        { x: 'LATAM', y: 6.4, y1: 28, y2: 46.7, y3: 65.1 },
        { x: 'MEA', y: 30, y1: 45.7, y2: 61.5, y3: 73 },
        { x: 'NA', y: 25.3, y1: 35.9, y2: 64, y3: 81.4 }
      ],
      primaryXAxis: {
        valueType: 'Category', interval: 1
      },
      marker: { visible: true, fill: 'Red', height: 10, width: 10,
        border:{width: 2, color: 'blue'} },
      title: "FB Penetration of Internet Audience"
    };
  },
  provide: {
    chart: [LineSeries, Category]
  }
};
</script>
<style>
  #container {
    height: 350px;
  }
</style>

```

{% previewsample "page.domainurl/code-snippet/chart/data-marker/marker-cs5" %}

### Customizing specific point

You can also customize the specific marker and label using [pointRender](#) event. The `pointRender` event allows you to change the shape, color and border for a point.

### APP.VUE

```

<template>

```

```

<div id="app">
  <ejs-chart id="container" :title='title'
:primaryXAxis='primaryXAxis' :pointRender='pointRender'>
    <e-series-collection>
      <e-series :dataSource='seriesData' type='Line' xName='x'
yName='y' name='December 2007'
      :marker='marker'> </e-series>
    </e-series-collection>
  </ejs-chart>
</div>
</template>
<script>
import Vue from "vue";
import { ChartPlugin, LineSeries, Category } from "@syncfusion/ej2-vue-
charts";
Vue.use(ChartPlugin);
export default {
  data() {
    return {
      seriesData: [
        { x: 'WW', y: 12, y1: 22, y2: 38.3, y3: 50 },
        { x: 'EU', y: 9.9, y1: 26, y2: 45.2, y3: 63.6 },
        { x: 'APAC', y: 4.4, y1: 9.3, y2: 18.2, y3: 20.9 },
        { x: 'LATAM', y: 6.4, y1: 28, y2: 46.7, y3: 65.1 },
        { x: 'MEA', y: 30, y1: 45.7, y2: 61.5, y3: 73 },
        { x: 'NA', y: 25.3, y1: 35.9, y2: 64, y3: 81.4 }
      ],
      primaryXAxis: {
        valueType: 'Category', interval: 1
      },
      marker: { visible: true, height: 10, width: 10 },
      title: "FB Penetration of Internet Audience"
    };
  },
  provide: {
    chart: [LineSeries, Category]
  },
  methods: {
    pointRender: function(args) {
      if(args.point.index === 3) {
        args.fill = 'red'
      }
    }
  }
};
</script>
<style>
#container {
  height: 350px;
}
</style>

```

{% previewsample "page.domainurl/code-snippet/chart/data-marker/marker-cs6" %}

Fill marker with series color

Marker can be filled with the series color by setting the [isFilled](#) property to **<b>true</b>**.

**APP.VUE**

```

<template>
  <div id="app">
    <ejs-chart id="container" :title='title'
:primaryXAxis='primaryXAxis' :pointRender='pointRender'>
      <e-series-collection>
        <e-series :dataSource='seriesData' type='Line' xName='x'
yName='y' name='December 2007'
          :marker='marker'> </e-series>
      </e-series-collection>
    </ejs-chart>
  </div>
</template>
<script>
import Vue from "vue";
import { ChartPlugin, LineSeries, Category } from "@syncfusion/ej2-vue-
charts";
Vue.use(ChartPlugin);
export default {
  data() {
    return {
      seriesData: [
        { x: 'WW', y: 12, y1: 22, y2: 38.3, y3: 50 },
        { x: 'EU', y: 9.9, y1: 26, y2: 45.2, y3: 63.6 },
        { x: 'APAC', y: 4.4, y1: 9.3, y2: 18.2, y3: 20.9 },
        { x: 'LATAM', y: 6.4, y1: 28, y2: 46.7, y3: 65.1 },
        { x: 'MEA', y: 30, y1: 45.7, y2: 61.5, y3: 73 },
        { x: 'NA', y: 25.3, y1: 35.9, y2: 64, y3: 81.4 }
      ],
      primaryXAxis: {
        valueType: 'Category', interval: 1
      },
      marker: { visible: true, height: 10, width: 10, isFilled: true},
      title: "FB Penetration of Internet Audience"
    };
  },
  provide: {
    chart: [LineSeries, Category]
  },
};
</script>
<style>
  #container {
    height: 350px;
  }
</style>

```

{% previewsample "page.domainurl/code-snippet/chart/data-marker/marker-cs7" %}

### Data labels in Vue Chart component

Data label can be added to a chart series by enabling the [visible](#)

option in the dataLabel. By default, the labels will arrange smartly without overlapping.

**APP.VUE**

```

<template>
  <div id="app">
    <ejs-chart id= "container" :title='title'
:primaryXAxis='primaryXAxis'>
      <e-series-collection>
        <e-series :dataSource='seriesData' type='Column' xName='x'
yName='y' name='Warmest' :marker='marker'></e-series>
      </e-series-collection>
    </ejs-chart>
  </div>
</template>
<script>
import Vue from "vue";
import { ChartPlugin, ColumnSeries, Category, DataLabel } from
"@syncfusion/ej2-vue-charts";
Vue.use(ChartPlugin);
export default {
  data() {
    return {
      seriesData: [
        { x: 'Jan', y: -7.1 }, { x: 'Feb', y: -3.7 },
        { x: 'Mar', y: 2 }, { x: 'Apr', y: 6.3 },
        { x: 'May', y: 13.3 }, { x: 'Jun', y: 18.0 },
        { x: 'Jul', y: 19.8 }, { x: 'Aug', y: 18.1 },
        { x: 'Sep', y: 13.1 }, { x: 'Oct', y: 4.1 },
        { x: 'Nov', y: -3.8 }, { x: 'Dec', y: -6.8 }
      ],
      primaryXAxis: {
        valueType: 'Category'
      },
      marker: {
        //Data label for chart series
        dataLabel: { visible: true }
      },
      title: "Alaska Weather Statistics - 2016"
    };
  },
  provide: {
    chart: [ColumnSeries, Category, DataLabel]
  }
};
</script>
<style>
#container {
  height: 350px;
}
</style>

```

{% previewsample "page.domainurl/code-snippet/chart/data-marker/datalabel-cs1" %}

Note: To use datalabel feature, we need to inject **DataLabel** into the **provide**.

#### Position

Using [position](#) property, you can place the label either on

**Top**, **Middle**, **Bottom** or **Outer** (outer is applicable for column and bar type series).

**APP.VUE**

```

<template>
  <div id="app">
    <ejs-chart id= "container" :title='title'
:primaryXAxis='primaryXAxis'>
      <e-series-collection>
        <e-series :dataSource='seriesData' type='Column' xName='x'
yName='y' name='Warmest' :marker='marker'> </e-series>
      </e-series-collection>
    </ejs-chart>
  </div>
</template>
<script>
import Vue from "vue";
import { ChartPlugin, ColumnSeries, Category, DataLabel } from
"@syncfusion/ej2-vue-charts";
Vue.use(ChartPlugin);
export default {
  data() {
    return {
      seriesData: [
        { x: 'Jan', y: -7.1 }, { x: 'Feb', y: -3.7 },
        { x: 'Mar', y: 2 }, { x: 'Apr', y: 6.3 },
        { x: 'May', y: 13.3 }, { x: 'Jun', y: 18.0 },
        { x: 'Jul', y: 19.8 }, { x: 'Aug', y: 18.1 },
        { x: 'Sep', y: 13.1 }, { x: 'Oct', y: 4.1 },
        { x: 'Nov', y: -3.8 }, { x: 'Dec', y: -6.8 }
      ],
      primaryXAxis: {
        valueType: 'Category'
      },
      marker: {
        dataLabel: { visible: true, position: 'Middle' }
      },
      title: "Alaska Weather Statistics - 2016"
    };
  },
  provide: {
    chart: [ColumnSeries, Category, DataLabel]
  }
};
</script>
<style>
#container {
  height: 350px;
}
</style>

```

{% previewsample "page.domainurl/code-snippet/chart/data-marker/datalabel-cs2" %}

Note: The position **Outer** is applicable for column and bar type series.

### Data Label Template

Label content can be formatted by using the template option. Inside the template, you can add the placeholder text ``${point.x}`` and ``${point.y}`` to display corresponding data points x & y value. Using [template](#) property, you can set data label template in chart.

### APP.VUE

```
<template>
  <div id="app">
    <ejs-chart id= "container" :title='title'
:primaryXAxis='primaryXAxis'>
      <e-series-collection>
        <e-series :dataSource='seriesData' type='Column' xName='x'
yName='y' name='Warmest' :marker='marker'> </e-series>
      </e-series-collection>
    </ejs-chart>
  </div>
</template>
<script>
import Vue from "vue";
import { ChartPlugin, ColumnSeries, Category, DataLabel } from
"@syncfusion/ej2-vue-charts";
let contentVue = Vue.component("contentTemplate", {
  template:
`<div><div>{{data.point.x}}</div><div>{{data.point.y}}</div></div>`,
  data() {
    return {
      data: {}
    };
  }
});
let contentTemplate = function() {
  return { template: contentVue };
};
Vue.use(ChartPlugin);
export default {
  data() {
    return {
      seriesData: [
        { x: 'Jan', y: -7.1 }, { x: 'Feb', y: -3.7 },
        { x: 'Mar', y: 2 }, { x: 'Apr', y: 6.3 },
        { x: 'May', y: 13.3 }, { x: 'Jun', y: 18.0 },
        { x: 'Jul', y: 19.8 }, { x: 'Aug', y: 18.1 },
        { x: 'Sep', y: 13.1 }, { x: 'Oct', y: 4.1 },
        { x: 'Nov', y: -3.8 }, { x: 'Dec', y: -6.8 }
      ],
      primaryXAxis: {
        valueType: 'Category'
      },
      marker: { dataLabel: { visible: true, position: 'Middle',
        template: contentTemplate } },
      title: "Alaska Weather Statistics - 2016"
    };
  },
  provide: {
    chart: [ColumnSeries, Category, DataLabel]
  }
}
```

```

    }
  };
</script>
<style>
  #container {
    height: 350px;
  }
</style>

```

{% previewsample "page.domainurl/code-snippet/chart/data-marker/datalabel-cs3" %}

### Text Mapping

Text from the data source can be mapped using `name` property.

### APP.VUE

```

<template>
  <div id="app">
    <ejs-chart id= "container" :title='title'
:primaryXAxis='primaryXAxis'>
      <e-series-collection>
        <e-series :dataSource='seriesData' type='Column' xName='x'
yName='y' name='Warmest' :marker='marker'> </e-series>
      </e-series-collection>
    </ejs-chart>
  </div>
</template>
<script>
import Vue from "vue";
import { ChartPlugin, ColumnSeries, Category, DataLabel } from
"@syncfusion/ej2-vue-charts";
Vue.use(ChartPlugin);
export default {
  data() {
    return {
      seriesData:[{ x: 'Jan', y: 12, text: 'January : 12°C' }, { x: 'Feb',
y: 8, text: 'February : 8°C' },
        { x: 'Mar', y: 11, text: 'March : 11°C' }, { x: 'Apr', y:
6, text: 'April : 6°C' }],
      primaryXAxis: {
        valueType: 'Category'
      },
      marker: { dataLabel: { visible: true, name: 'text' } },
      title: "Alaska Weather Statistics - 2016"
    };
  },
  provide: {
    chart: [ColumnSeries, Category, DataLabel]
  }
};
</script>
<style>
  #container {
    height: 350px;
  }
</style>

```



{% previewsample "page.domainurl/code-snippet/chart/data-marker/datalabel-cs4" %}

### Format

Data label for the chart can be formatted using [format](#) property. You can use the global formatting options, such as 'n', 'p', and 'c'.

### APP.VUE

```
<template>
  <div id="app">
    <ejs-chart id= "container" :title='title'
:primaryXAxis='primaryXAxis'>
      <e-series-collection>
        <e-series :dataSource='seriesData' type='Column' xName='x'
yName='y' name='Warmest' :marker='marker'> </e-series>
      </e-series-collection>
    </ejs-chart>
  </div>
</template>
<script>
import Vue from "vue";
import { ChartPlugin, ColumnSeries, Category, DataLabel } from
"@syncfusion/ej2-vue-charts";
Vue.use(ChartPlugin);
export default {
  data() {
    return {
      seriesData:[{ x: 'Jan', y: 12, text: 'January : 12°C' }, { x: 'Feb',
y: 8, text: 'February : 8°C' },
        { x: 'Mar', y: 11, text: 'March : 11°C' }, { x: 'Apr', y:
6, text: 'April : 6°C' }],
      primaryXAxis: {
        valueType: 'Category'
      },
      marker: { dataLabel: { visible: true, format: 'n2' } },
      title: "Alaska Weather Statistics - 2016"
    };
  },
  provide: {
    chart: [ColumnSeries, Category, DataLabel]
  }
};
</script>
<style>
  #container {
    height: 350px;
  }
</style>
```

{% previewsample "page.domainurl/code-snippet/chart/data-marker/datalabel-cs5" %}

Value	Format	Resultant Value	Description
1000	n1	1000.0	The number is rounded to 1 decimal place.

1000	n2	1000.00	The number is rounded to 2 decimal places.
1000	n3	1000.000	The number is rounded to 3 decimal place.
0.01	p1	1.0%	The number is converted to percentage with 1 decimal place.
0.01	p2	1.00%	The number is converted to percentage with 2 decimal place.
0.01	p3	1.000%	The number is converted to percentage with 3 decimal place.
1000	c1	\$1000.0	The currency symbol is appended to number and number is rounded to 1 decimal place.
1000	c2	\$1000.00	The currency symbol is appended to number and number is rounded to 2 decimal place.

### Margin

**margin** for data label can be applied to using **left**, **right**, **bottom** and **top** properties.

### APP.VUE

```
<template>
  <div id="app">
    <ejs-chart id= "container" :title='title'
:primaryXAxis='primaryXAxis'>
      <e-series-collection>
        <e-series :dataSource='seriesData' type='Column' xName='x'
yName='y' name='Warmest' :marker='marker'> </e-series>
      </e-series-collection>
    </ejs-chart>
  </div>
</template>
<script>
import Vue from "vue";
import { ChartPlugin, ColumnSeries, Category, DataLabel } from
"@syncfusion/ej2-vue-charts";
Vue.use(ChartPlugin);
export default {
  data() {
    return {
      seriesData: [
        { x: 'Jan', y: -7.1 }, { x: 'Feb', y: -3.7 },
        { x: 'Mar', y: 2 }, { x: 'Apr', y: 6.3 },
        { x: 'May', y: 13.3 }, { x: 'Jun', y: 18.0 },
        { x: 'Jul', y: 19.8 }, { x: 'Aug', y: 18.1 },
        { x: 'Sep', y: 13.1 }, { x: 'Oct', y: 4.1 },
        { x: 'Nov', y: -3.8 }, { x: 'Dec', y: -6.8 }
      ],
      primaryXAxis: {
        valueType: 'Category'
      },
      marker: { dataLabel: { visible: true, border:{width: 1, color :
'red'},
                    margin:{
                      left:5,
```

```

        right:5,
        top:5,
        bottom:5
    } }

    },
    title: "Alaska Weather Statistics - 2016"
  };
},
provide: {
  chart: [ColumnSeries, Category, DataLabel]
}
};
</script>
<style>
  #container {
    height: 350px;
  }
</style>

```

{% previewsample "page.domainurl/code-snippet/chart/data-marker/datalabel-cs6" %}

### DataLabel Rotation

Using `angle` property, you can rotate the data label by its given angle.

### APP.VUE

```

<template>
  <div id="app">
    <ejs-chart id= "container" :title='title'
    :primaryXAxis='primaryXAxis'>
      <e-series-collection>
        <e-series :dataSource='seriesData' type='Column' xName='x'
        yName='y' name='Warmest' :marker='marker'> </e-series>
      </e-series-collection>
    </ejs-chart>
  </div>
</template>
<script>
import Vue from "vue";
import { ChartPlugin, ColumnSeries, Category, DataLabel } from
"@syncfusion/ej2-vue-charts";
Vue.use(ChartPlugin);
export default {
  data() {
    return {
      seriesData: [
        { x: 'Jan', y: -7.1 }, { x: 'Feb', y: -3.7 },
        { x: 'Mar', y: 2 }, { x: 'Apr', y: 6.3 },
        { x: 'May', y: 13.3 }, { x: 'Jun', y: 18.0 },
        { x: 'Jul', y: 19.8 }, { x: 'Aug', y: 18.1 },
        { x: 'Sep', y: 13.1 }, { x: 'Oct', y: 4.1 },
        { x: 'Nov', y: -3.8 }, { x: 'Dec', y: -6.8 }
      ],
      primaryXAxis: {
        valueType: 'Category'
      },
    },
  },
};

```

```

        marker: { dataLabel: { visible: true, border: { width: 1, color :
'red' },
                    margin: {
                        left: 5,
                        right: 5,
                        top: 5,
                        bottom: 5
                    }, angle: 45, enableRotation: true }
        },
        title: "Alaska Weather Statistics - 2016"
    };
    },
    provide: {
        chart: [ColumnSeries, Category, DataLabel]
    }
};
</script>
<style>
    #container {
        height: 350px;
    }
</style>

```

{% previewsample "page.domainurl/code-snippet/chart/data-marker/marker-cs1" %}

### Customization

**stroke** and **border** of data label can be customized using **fill** and **border** properties. Rounded corners can be customized using **rx** and **ry** properties.

### APP.VUE

```

<template>
    <div id="app">
        <ejs-chart id= "container" :title='title'
:primaryXAxis='primaryXAxis'>
            <e-series-collection>
                <e-series :dataSource='seriesData' type='Column' xName='x'
yName='y' name='Warmest' :marker='marker'> </e-series>
            </e-series-collection>
        </ejs-chart>
    </div>
</template>
<script>
import Vue from "vue";
import { ChartPlugin, ColumnSeries, Category, DataLabel } from
"@syncfusion/ej2-vue-charts";
Vue.use(ChartPlugin);
export default {
    data() {
        return {
            seriesData: [
                { x: 'Jan', y: -7.1 }, { x: 'Feb', y: -3.7 },
                { x: 'Mar', y: 2 }, { x: 'Apr', y: 6.3 },
                { x: 'May', y: 13.3 }, { x: 'Jun', y: 18.0 },
                { x: 'Jul', y: 19.8 }, { x: 'Aug', y: 18.1 },
            ]
        }
    }
}

```

```

        { x: 'Sep', y: 13.1 }, { x: 'Oct', y: 4.1 },
        { x: 'Nov', y: -3.8 }, { x: 'Dec', y: -6.8 }
    ],
    primaryXAxis: {
        valueType: 'Category'
    },
    marker: { dataLabel: { visible: true,
        border: { width: 2, color: 'red' },
        rx: 10, ry: 10
    }
    },
    title: "Alaska Weather Statistics - 2016"
};
},
provide: {
    chart: [ColumnSeries, Category, DataLabel]
}
};
</script>
<style>
#container {
    height: 350px;
}
</style>

```

{% previewsample "page.domainurl/code-snippet/chart/data-marker/datalabel-cs7" %}

Note: `rx` and `ry` properties requires `border` values not to be null.

### Customizing Specific Point

You can also customize the specific marker and label using [pointRender](#) and [textRender](#) event.

`pointRender` event allows you to change the shape, color and border for a point, whereas the `textRender` event allows you to change the text for the point.

### APP.VUE

```

<template>
    <div id="app">
        <ejs-chart id="container" :title='title'
        :primaryXAxis='primaryXAxis' :pointRender='pointRender'
        :textRender='textRender'>
            <e-series-collection>
                <e-series :dataSource='seriesData' type='Column' xName='x'
                yName='y' name='Warmest' :marker='marker'> </e-series>
            </e-series-collection>
        </ejs-chart>
    </div>
</template>
<script>
import Vue from "vue";
import { ChartPlugin, ColumnSeries, Category, DataLabel } from
"@syncfusion/ej2-vue-charts";
Vue.use(ChartPlugin);
export default {
    data() {
        return {

```

```

seriesData: [
    { x: 'Jan', y: -7.1 }, { x: 'Feb', y: -3.7 },
    { x: 'Mar', y: 2 }, { x: 'Apr', y: 6.3 },
    { x: 'May', y: 13.3 }, { x: 'Jun', y: 18.0 },
    { x: 'Jul', y: 19.8 }, { x: 'Aug', y: 18.1 },
    { x: 'Sep', y: 13.1 }, { x: 'Oct', y: 4.1 },
    { x: 'Nov', y: -3.8 }, { x: 'Dec', y: -6.8 }
],
primaryXAxis: {
    valueType: 'Category'
},
marker:{ dataLabel: { visible: true } },
title: "Alaska Weather Statistics - 2016"
};
},
provide: {
    chart: [ColumnSeries, Category, DataLabel]
},
methods: {
    pointRender: function(args) {
        if (args.point.index === 6) {
            args.fill = 'red'
        }
    },
    textRender: function(args) {
        if (args.point.index === 6) {
            args.text = 'Maximum Temperature';
            args.color = 'red';
        }
    }
}
};
</script>
<style>
    #container {
        height: 350px;
    }
</style>

```

{% previewsample "page.domainurl/code-snippet/chart/data-marker/datalabel-cs8" %}

### Show percentage based on each series points

You can calculate the percentage value based on the sum for each series using the `seriesRender` and `textRender` events in the chart. In `seriesRender` calculate the sum of each series y values and In `textRender` calculate percentage value based on the sum value and modify the text.

### APP.VUE

```

<template>
  <div class="control-section">
    <div align="center">
      <ejs-chart
        style="display: block"
        :theme="theme"
        align="center"
        id="chartcontainer"

```

```

      :title="title"
      :primaryXAxis="primaryXAxis"
      :primaryYAxis="primaryYAxis"
      :chartArea="chartArea"
      :width="width"
      :tooltip="tooltip"
      :textRender="onTextRender"
      :seriesRender="onSeriesRender"
    >
    <e-series-collection>
      <e-series
        :dataSource="seriesData"
        type="Column"
        xName="x"
        yName="y"
        name="Gold"
        width="2"
        :marker="marker"
      >
    </e-series>
    <e-series
      :dataSource="seriesData1"
      type="Column"
      xName="x"
      yName="y"
      name="Silver"
      width="2"
      :marker="marker"
    >
    </e-series>
    <e-series
      :dataSource="seriesData2"
      type="Column"
      xName="x"
      yName="y"
      name="Bronze"
      width="2"
      :marker="marker"
    >
    </e-series>
  </e-series-collection>
</ejs-chart>
</div>
</div>
</template>
<style scoped>
</style>
<script>
import Vue from "vue";
import { Browser } from "@syncfusion/ej2-base";
import {
  ChartPlugin,
  ColumnSeries,
  Category,
  DataLabel,
  Tooltip,
  Legend,

```

```

} from "@syncfusion/ej2-vue-charts";
Vue.use(ChartPlugin);
let total = [];
export default {
  data: function () {
    return {
      seriesData: [
        { x: "USA", y: 46 },
        { x: "GBR", y: 27 },
        { x: "CHN", y: 26 },
      ],
      seriesData1: [
        { x: "USA", y: 37 },
        { x: "GBR", y: 23 },
        { x: "CHN", y: 18 },
      ],
      seriesData2: [
        { x: "USA", y: 38 },
        { x: "GBR", y: 17 },
        { x: "CHN", y: 26 },
      ],
      //Initializing Primary X Axis
      primaryXAxis: {
        valueType: "Category",
        interval: 1,
        majorGridLines: { width: 0 },
      },
      chartArea: { border: { width: 0 } },
      //Initializing Primary Y Axis
      primaryYAxis: {
        majorGridLines: { width: 0 },
        majorTickLines: { width: 0 },
        lineStyle: { width: 0 },
        labelStyle: { color: "transparent" },
      },
      width: Browser.isDevice ? "100%" : "60%",
      marker: {
        dataLabel: {
          visible: true,
          position: "Top",
          font: { fontWeight: "600", color: "#ffffff" },
        },
      },
      tooltip: {
        enable: true,
      },
      title: "Olympic Medal Counts - RIO",
    };
  },
  provide: {
    chart: [ColumnSeries, Legend, DataLabel, Category, Tooltip],
  },
  methods: {
    onSeriesRender: function (args) {
      for (let i = 0; i < args.data.length; i++) {
        if (!total[args.data[i].x]) total[args.data[i].x] = 0;
        total[args.data[i].x] += parseInt(args.data[i].y);
      }
    }
  }
}

```



```

    }
  },
  onTextRender: function (args) {
    let percentage = (parseInt(args.text) / total[args.point.x]) * 100;
    percentage = percentage % 1 === 0 ? percentage :
percentage.toFixed(2);
    args.text = percentage + "%";
  },
},
};
</script>

```

{% previewsample "page.domainurl/code-snippet/chart/data-marker/datalabel-cs9" %}

### Chart annotations in Vue Chart component

Annotations are used to mark the specific area of interest in the chart area with texts, shapes or images.

<!-- markdownlint-disable MD033 -->

You can add annotations to the chart by using the `<code>annotations</code> option. By using the content option of annotation object, you can specify`

the id of the element that needs to be displayed in the chart area.

### APP.VUE

```

<template>
  <div id="app">
    <ejs-chart id="container" :title='title'
:primaryXAxis='primaryXAxis' :primaryYAxis='primaryYAxis'
:annotations='annotations'>
      <e-series-collection>
        <e-series :dataSource='seriesData' type='Column'
xName='country' yName='gold' name='Gold'> </e-series>
      </e-series-collection>
    </ejs-chart>
  </div>
</template>
<script>
import Vue from "vue";
import { ChartPlugin, ColumnSeries, Category, ChartAnnotation } from
"@syncfusion/ej2-vue-charts";
Vue.use(ChartPlugin);
export default {
  data() {
    return {
      seriesData: [
        { country: "USA", gold: 50 },
        { country: "China", gold: 40 },
        { country: "Japan", gold: 70 },
        { country: "Australia", gold: 60 },
        { country: "France", gold: 50 },
        { country: "Germany", gold: 40 },
        { country: "Italy", gold: 40 },
        { country: "Sweden", gold: 30 }
      ],
      annotations:[{

```

```

        content: '70 Gold Medals',
        coordinateUnits: 'Point',
        x: 'France',
        y: 50
    }],
    primaryXAxis: {
        valueType: 'Category',
        title: 'Countries'
    },
    primaryYAxis:
    {
        minimum: 0, maximum: 80,
        interval: 20, title: 'Medals'
    },
    title: "Olympic Medals"
};
},
provide: {
    chart: [ColumnSeries, Category, ChartAnnotation]
},
};
</script>
<style>
    #container {
        height: 350px;
    }
</style>

```

{% previewsample "page.domainurl/code-snippet/chart/series/column-cs1" %}

Note: To use annotation feature in chart, we need to inject **ChartAnnotation** into the **provide**.

### Region

Annotations can be placed either with respect to **Series** or **Chart**. by default, it will placed with respect to **Chart**.

### APP.VUE

```

<template>
    <div id="app">
        <ejs-chart id="container" :title='title'
        :primaryXAxis='primaryXAxis' :primaryYAxis='primaryYAxis'
        :annotations='annotations'>
            <e-series-collection>
                <e-series :dataSource='seriesData' type='Column'
                xName='country' yName='gold' name='Gold'> </e-series>
            </e-series-collection>
        </ejs-chart>
    </div>
</template>
<script>
import Vue from "vue";
import { ChartPlugin, ColumnSeries, Category, ChartAnnotation } from
"@syncfusion/ej2-vue-charts";
let contentVue = Vue.component("contentTemplate", {
    template: '<div>Highest Medal Count</div>',

```

```

data() {
  return {
    data: {}
  };
}
});
let contentTemplate = function() {
  return { template: contentVue };
};
Vue.use(ChartPlugin);
export default {
  data() {
    return {
      seriesData: [
        { country: "USA", gold: 50 },
        { country: "China", gold: 40 },
        { country: "Japan", gold: 70 },
        { country: "Australia", gold: 60 },
        { country: "France", gold: 50 },
        { country: "Germany", gold: 40 },
        { country: "Italy", gold: 40 },
        { country: "Sweden", gold: 30 }
      ],
      annotations:[{
        content: contentTemplate,
        region: 'Series',
        coordinateUnits: 'Point',
        x: 'France',
        y: 50
      }],
      primaryXAxis: {
        valueType: 'Category',
        title: 'Countries'
      },
      primaryYAxis:
      {
        minimum: 0, maximum: 80,
        interval: 20, title: 'Medals'
      },
      title: "Olympic Medals"
    };
  },
  provide: {
    chart: [ColumnSeries, Category, ChartAnnotation]
  },
};
</script>
<style>
  #container {
    height: 350px;
  }
</style>

```

{% previewsample "page.domainurl/code-snippet/chart/series/column-cs2" %}

## Co-ordinate Units

Specified the coordinates units of the annotation either **Pixel** or **Point**.

### APP.VUE

```
<template>
  <div id="app">
    <ejs-chart id="container" :title='title'
:primaryXAxis='primaryXAxis' :primaryYAxis='primaryYAxis'
:annotations='annotations'>
      <e-series-collection>
        <e-series :dataSource='seriesData' type='Column'
xName='country' yName='gold' name='Gold'> </e-series>
      </e-series-collection>
    </ejs-chart>
  </div>
</template>
<script>
import Vue from "vue";
import { ChartPlugin, ColumnSeries, Category, ChartAnnotation } from
"@syncfusion/ej2-vue-charts";
let contentVue = Vue.component("contentTemplate", {
  template: '<div style="border: 1px solid black; padding: 5px 5px 5px 5px,
background:#f5f5f5">Annotation in Pixel</div>',
  data() {
    return {
      data: {}
    };
  }
});
let contentTemplate = function() {
  return { template: contentVue };
};
Vue.use(ChartPlugin);
export default {
  data() {
    return {
      seriesData: [
        { country: "USA", gold: 50 },
        { country: "China", gold: 40 },
        { country: "Japan", gold: 70 },
        { country: "Australia", gold: 60 },
        { country: "France", gold: 50 },
        { country: "Germany", gold: 40 },
        { country: "Italy", gold: 40 },
        { country: "Sweden", gold: 30 }
      ],
      annotations:[{
        content: contentTemplate,
        coordinateUnits: 'Pixel',
        x: 150,
        y: 50
      }],
      primaryXAxis: {
        valueType: 'Category',
        title: 'Countries'
      },
    },
  },
}
```

```

        primaryYAxis:
        {
            minimum: 0, maximum: 80,
            interval: 20, title: 'Medals'
        },
        title: "Olympic Medals"
    };
},
provide: {
    chart: [ColumnSeries, Category, ChartAnnotation]
},
};
</script>
<style>
    #container {
        height: 350px;
    }
</style>

```

{% previewsample "page.domainurl/code-snippet/chart/series/column-cs3" %}

### Alignment

Annotation provides **verticalAlignment** and **horizontalAlignment**. The **verticalAlignment** can be customized

via **Top**, **Bottom** or **Middle** and the **horizontalAlignment** can be customized via **Near**, **Far** or **Center**.

### APP.VUE

```

<template>
    <div id="app">
        <ejs-chart id="container" :title='title'
        :primaryXAxis='primaryXAxis' :primaryYAxis='primaryYAxis'
        :annotations='annotations'>
            <e-series-collection>
                <e-series :dataSource='seriesData' type='Column'
                xName='country' yName='gold' name='Gold'> </e-series>
            </e-series-collection>
        </ejs-chart>
    </div>
</template>
<script>
import Vue from "vue";
import { ChartPlugin, ColumnSeries, Category, ChartAnnotation } from
"@syncfusion/ej2-vue-charts";
let contentVue = Vue.component("contentTemplate", {
    template: '<div style="border: 1px solid black; padding: 5px 5px 5px 5px,
    background:#f5f5f5">Highest Medal Count</div>',
    data() {
        return {
            data: {}
        };
    }
});
let contentTemplate = function() {
    return { template: contentVue };
}

```

```

};
Vue.use(ChartPlugin);
export default {
  data() {
    return {
      seriesData: [
        { country: "USA", gold: 50 },
        { country: "China", gold: 40 },
        { country: "Japan", gold: 70 },
        { country: "Australia", gold: 60 },
        { country: "France", gold: 50 },
        { country: "Germany", gold: 40 },
        { country: "Italy", gold: 40 },
        { country: "Sweden", gold: 30 }
      ],
      annotations:[{
        content: contentTemplate,
        x: 'France',
        y: 50,
        verticalAlignment:'Top',
        horizontalAlignment:'Near'
      }],
      primaryXAxis: {
        valueType: 'Category',
        title: 'Countries'
      },
      primaryYAxis:
      {
        minimum: 0, maximum: 80,
        interval: 20, title: 'Medals'
      },
      title: "Olympic Medals"
    };
  },
  provide: {
    chart: [ColumnSeries, Category, ChartAnnotation]
  },
};
</script>
<style>
  #container {
    height: 350px;
  }
</style>

```

{% previewsample "page.domainurl/code-snippet/chart/series/column-cs4" %}

### Adding y-axis sub title through on annotation

By setting text div in the `content` option of annotation object you can add sub title to chart y-axis. Specified the `coordinate` value as `pixel` and customize x and y location of the text.

### APP.VUE

```

<template>
  <div id="app">

```

```

        <ejs-chart id="container" :title='title'
:primaryXAxis='primaryXAxis' :primaryYAxis='primaryYAxis'
:annotations='annotations'>
            <e-series-collection>
                <e-series :dataSource='seriesData' type='Column'
xName='country' yName='gold' name='Gold'> </e-series>
            </e-series-collection>
        </ejs-chart>
    </div>
</template>
<script>
import Vue from "vue";
import { ChartPlugin, ColumnSeries, Category, ChartAnnotation } from
"@syncfusion/ej2-vue-charts";
    let contentVue = Vue.component("contentTemplate", {
        template: '<div id="text" style="transform: rotate(-90deg);">Speed
Rate</div>',
        data() {
            return {
                data: {}
            };
        }
    });
let contentTemplate = function() {
    return { template: contentVue };
};
Vue.use(ChartPlugin);
export default {
    data() {
        return {
            seriesData: [
                { country: "USA", gold: 50 },
                { country: "China", gold: 40 },
                { country: "Japan", gold: 70 },
                { country: "Australia", gold: 60 },
                { country: "France", gold: 50 },
                { country: "Germany", gold: 40 },
                { country: "Italy", gold: 40 },
                { country: "Sweden", gold: 30 }
            ],
            annotations:[{
                content: contentTemplate,
                coordinateUnits: 'Pixel',
                x: 13,
                y: 180,
                region: 'Chart'
            }],
            primaryXAxis: {
                valueType: 'Category',
                title: 'Countries'
            },
            primaryYAxis:
            {
                minimum: 0, maximum: 80,
                interval: 20, title: '(m2/min)'
            },
            title: "Olympic Medals"
        }
    }
}

```

```

    };
  },
  provide: {
    chart: [ColumnSeries, Category, ChartAnnotation]
  },
};
</script>
<style>
  #container {
    height: 350px;
  }
</style>

```

{% previewsample "page.domainurl/code-snippet/chart/series/column-cs5" %}

### Legend in Vue Chart component

Legend provides information about the series rendered in the chart which can be customized using legend properties.

#### Position and Alignment

By using the [position](#) property, you can position the legend

at left, right, top or bottom of the chart. The legend is positioned at the bottom of the chart, by default.

### APP.VUE

```

<template>
  <div id="app">
    <ejs-chart id="container" :title='title'
:primaryXAxis='primaryXAxis' :legendSettings='legendSettings'>
      <e-series-collection>
        <e-series :dataSource='seriesData' type='Column'
xName='country' yName='gold' name='Gold'> </e-series>
        <e-series :dataSource='seriesData' type='Column'
xName='country' yName='silver' name='Silver'> </e-series>
        <e-series :dataSource='seriesData' type='Column'
xName='country' yName='bronze' name='Bronze'> </e-series>
      </e-series-collection>
    </ejs-chart>
  </div>
</template>
<script>
import Vue from "vue";
import { ChartPlugin, ColumnSeries, Category, Legend } from
"@syncfusion/ej2-vue-charts";
Vue.use(ChartPlugin);
export default {
  data() {
    return {
      seriesData: [
        { country: "USA", gold: 50, silver: 70, bronze: 45 },
        { country: "China", gold: 40, silver: 60, bronze: 55 },
        { country: "Japan", gold: 70, silver: 60, bronze: 50 },
        { country: "Australia", gold: 60, silver: 56, bronze: 40 },
        { country: "France", gold: 50, silver: 45, bronze: 35 },
        { country: "Germany", gold: 40, silver: 30, bronze: 22 },

```



```

        { country: "Italy", gold: 40, silver: 35, bronze: 37 },
        { country: "Sweden", gold: 30, silver: 25, bronze: 27 }
    ],
    primaryXAxis: {
        valueType: 'Category',
        title: 'Countries'
    },
    legendSettings: {
        visible: true,
        position: 'Top'
    },
    title: "Olympic Medals"
};
},
provide: {
    chart: [ColumnSeries, Category, Legend ]
}
};
</script>
<style>
    #container {
        height: 350px;
    }
</style>

```

{% previewsample "page.domainurl/code-snippet/chart/axis/category-cs31" %}

- Custom position helps you to position the legend anywhere in the chart using x, y coordinates.

### APP.VUE

```

<template>
    <div id="app">
        <ejs-chart id="container" :title='title'
        :primaryXAxis='primaryXAxis' :legendSettings='legendSettings'>
            <e-series-collection>
                <e-series :dataSource='seriesData' type='Column'
                xName='country' yName='gold' name='Gold'> </e-series>
                <e-series :dataSource='seriesData' type='Column'
                xName='country' yName='silver' name='Silver'> </e-series>
                <e-series :dataSource='seriesData' type='Column'
                xName='country' yName='bronze' name='Bronze'> </e-series>
            </e-series-collection>
        </ejs-chart>
    </div>
</template>
<script>
import Vue from "vue";
import { ChartPlugin, ColumnSeries, Category, Legend } from
"@syncfusion/ej2-vue-charts";
Vue.use(ChartPlugin);
export default {
    data() {
        return {
            seriesData: [

```

```

        { country: "USA", gold: 50, silver: 70, bronze: 45 },
        { country: "China", gold: 40, silver: 60, bronze: 55 },
        { country: "Japan", gold: 70, silver: 60, bronze: 50 },
        { country: "Australia", gold: 60, silver: 56, bronze: 40 },
        { country: "France", gold: 50, silver: 45, bronze: 35 },
        { country: "Germany", gold: 40, silver: 30, bronze: 22 },
        { country: "Italy", gold: 40, silver: 35, bronze: 37 },
        { country: "Sweden", gold: 30, silver: 25, bronze: 27 }
    ],
    primaryXAxis: {
        valueType: 'Category',
        title: 'Countries'
    },
    legendSettings: {
        visible: true,
        position: 'Custom',
        location: { x: 200, y: 40 }
    },
    title: "Olympic Medals"
};
},
provide: {
    chart: [ColumnSeries, Category, Legend ]
}
};
</script>
<style>
#container {
    height: 350px;
}
</style>

```

{% previewsample "page.domainurl/code-snippet/chart/axis/category-cs32" %}

### Legend Reverse

You can reverse the order of the legend items by using the [reverse](#) property. By default, legend for the first series in the collection will be placed first.

### APP.VUE

```

<template>
  <ejs-chart
    style="display: block"
    :theme="theme"
    align="center"
    id="container"
    :title="title"
    :primaryXAxis="primaryXAxis"
    :primaryYAxis="primaryYAxis"
    :chartArea="chartArea"
    :tooltip="tooltip"
    :legendSettings="legendSettings"
  >
    <e-series-collection>
      <e-series
        :dataSource="seriesData"

```

```

        type="Column"
        xName="x"
        yName="y"
        name="Gold"
        width="2"
      >
    </e-series>
    <e-series
      :dataSource="seriesData1"
      type="Column"
      xName="x"
      yName="y"
      name="Silver"
      width="2"
    >
    </e-series>
    <e-series
      :dataSource="seriesData2"
      type="Column"
      xName="x"
      yName="y"
      name="Bronze"
      width="2"
    >
    </e-series>
  </e-series-collection>
</ejs-chart>
</template>
<style scoped>
</style>
<script>
import Vue from "vue";
import { Browser } from "@syncfusion/ej2-base";
import {
  ChartPlugin,
  ColumnSeries,
  Category,
  DataLabel,
  Tooltip,
  Legend,
} from "@syncfusion/ej2-vue-charts";
Vue.use(ChartPlugin);
export default Vue.extend({
  data: function () {
    return {
      seriesData: [
        { x: "USA", y: 46 },
        { x: "GBR", y: 27 },
        { x: "CHN", y: 26 },
      ],
      seriesData1: [
        { x: "USA", y: 37 },
        { x: "GBR", y: 23 },
        { x: "CHN", y: 18 },
      ],
      seriesData2: [
        { x: "USA", y: 38 },

```

```

        { x: "GBR", y: 17 },
        { x: "CHN", y: 26 },
    ],
    //Initializing Primary X Axis
    primaryXAxis: {
        valueType: "Category",
        interval: 1,
        majorGridLines: { width: 0 },
    },
    chartArea: { border: { width: 0 } },
    //Initializing Primary Y Axis
    primaryYAxis: {
        majorGridLines: { width: 0 },
        majorTickLines: { width: 0 },
        lineStyle: { width: 0 },
        labelStyle: { color: "transparent" },
    },
    tooltip: {
        enable: true,
    },
    title: "Olympic Medal Counts - RIO",
    legendSettings: { visible: true, reverse: true },
    };
    },
    provide: {
        chart: [ColumnSeries, Legend, DataLabel, Category, Tooltip],
    },
    });
</script>

```

```
{% previewsample "page.domainurl/code-snippet/chart/axis/legend-reverse-cs1" %}
```

```
<!-- markdownlint-disable MD036 -->
```

### Legend Alignment

```
<!-- markdownlint-disable MD036 -->
```

You can align the legend as **center**, **far** or **near** to the chart using [alignment](#) property.

### APP.VUE

```

<template>
  <div id="app">
    <ejs-chart id="container" :title='title'
    :primaryXAxis='primaryXAxis' :legendSettings='legendSettings'>
      <e-series-collection>
        <e-series :dataSource='seriesData' type='Column'
        xName='country' yName='gold' name='Gold'> </e-series>
        <e-series :dataSource='seriesData' type='Column'
        xName='country' yName='silver' name='Silver'> </e-series>
        <e-series :dataSource='seriesData' type='Column'
        xName='country' yName='bronze' name='Bronze'> </e-series>
      </e-series-collection>
    </ejs-chart>
  </div>
</template>

```

```

<script>
import Vue from "vue";
import { ChartPlugin, ColumnSeries, Category, Legend } from
"@syncfusion/ej2-vue-charts";
Vue.use(ChartPlugin);
export default {
  data() {
    return {
      seriesData: [
        { country: "USA", gold: 50, silver: 70, bronze: 45 },
        { country: "China", gold: 40, silver: 60, bronze: 55 },
        { country: "Japan", gold: 70, silver: 60, bronze: 50 },
        { country: "Australia", gold: 60, silver: 56, bronze: 40 },
        { country: "France", gold: 50, silver: 45, bronze: 35 },
        { country: "Germany", gold: 40, silver: 30, bronze: 22 },
        { country: "Italy", gold: 40, silver: 35, bronze: 37 },
        { country: "Sweden", gold: 30, silver: 25, bronze: 27 }
      ],
      primaryXAxis: {
        valueType: 'Category',
        title: 'Countries'
      },
      legendSettings: {
        visible: true, position: 'Top', alignment: 'Near'
      },
      title: "Olympic Medals"
    };
  },
  provide: {
    chart: [ColumnSeries, Category, Legend ]
  }
};
</script>
<style>
#container {
  height: 350px;
}
</style>

```

{% previewsample "page.domainurl/code-snippet/chart/axis/category-cs33" %}

### Customization

To change the legend icon shape, you can use [legendShape](#) property

in the [series](#). By default legend icon shape is `seriesType`.

### APP.VUE

```

<template>
  <div id="app">
    <ejs-chart id="container" :title='title'
    :primaryXAxis='primaryXAxis' :legendSettings='legendSettings'>
      <e-series-collection>
        <e-series :dataSource='seriesData' type='Column'
        xName='country' yName='gold' name='Gold' legendShape='Circle'> </e-series>

```

```

        <e-series :dataSource='seriesData' type='Column'
xName='country' yName='silver' name='Silver' legendShape='SeriesType'> </e-
series>
        <e-series :dataSource='seriesData' type='Column'
xName='country' yName='bronze' name='Bronze' legendShape='Rectangle'> </e-
series>
    </e-series-collection>
</ejs-chart>
</div>
</template>
<script>
import Vue from "vue";
import { ChartPlugin, ColumnSeries, Category, Legend } from
"@syncfusion/ej2-vue-charts";
Vue.use(ChartPlugin);
export default {
  data() {
    return {
      seriesData: [
        { country: "USA", gold: 50, silver: 70, bronze: 45 },
        { country: "China", gold: 40, silver: 60, bronze: 55 },
        { country: "Japan", gold: 70, silver: 60, bronze: 50 },
        { country: "Australia", gold: 60, silver: 56, bronze: 40 },
        { country: "France", gold: 50, silver: 45, bronze: 35 },
        { country: "Germany", gold: 40, silver: 30, bronze: 22 },
        { country: "Italy", gold: 40, silver: 35, bronze: 37 },
        { country: "Sweden", gold: 30, silver: 25, bronze: 27 }
      ],
      primaryXAxis: {
        valueType: 'Category',
        title: 'Countries'
      },
      legendSettings: {
        visible: true
      },
      title: "Olympic Medals"
    };
  },
  provide: {
    chart: [ColumnSeries, Category, Legend ]
  }
};
</script>
<style>
#container {
  height: 350px;
}
</style>

```

{% previewsample "page.domainurl/code-snippet/chart/axis/category-cs34" %}

### Legend Size

By default, legend takes 20% - 25% of the chart's height horizontally, when it is placed on top or bottom position and 20% - 25% of the chart's width vertically, when placed on left or right position of the chart. You can change this default legend size by using the [width](#) and [height](#) property of the `legendSettings`.

**APP.VUE**

```

<template>
  <div id="app">
    <ejs-chart id="container" :title='title'
:primaryXAxis='primaryXAxis' :legendSettings='legendSettings'>
      <e-series-collection>
        <e-series :dataSource='seriesData' type='Column'
xName='country' yName='gold' name='Gold' legendShape='Circle'> </e-series>
        <e-series :dataSource='seriesData' type='Column'
xName='country' yName='silver' name='Silver' legendShape='SeriesType'> </e-
series>
        <e-series :dataSource='seriesData' type='Column'
xName='country' yName='bronze' name='Bronze' legendShape='Rectangle'> </e-
series>
      </e-series-collection>
    </ejs-chart>
  </div>
</template>
<script>
import Vue from "vue";
import { ChartPlugin, ColumnSeries, Category, Legend } from
"@syncfusion/ej2-vue-charts";
Vue.use(ChartPlugin);
export default {
  data() {
    return {
      seriesData: [
        { country: "USA", gold: 50, silver: 70, bronze: 45 },
        { country: "China", gold: 40, silver: 60, bronze: 55 },
        { country: "Japan", gold: 70, silver: 60, bronze: 50 },
        { country: "Australia", gold: 60, silver: 56, bronze: 40 },
        { country: "France", gold: 50, silver: 45, bronze: 35 },
        { country: "Germany", gold: 40, silver: 30, bronze: 22 },
        { country: "Italy", gold: 40, silver: 35, bronze: 37 },
        { country: "Sweden", gold: 30, silver: 25, bronze: 27 }
      ],
      primaryXAxis: {
        valueType: 'Category',
        title: 'Countries'
      },
      legendSettings: {
        visible: true, height: '100', width: '500'
      },
      title: "Olympic Medals"
    };
  },
  provide: {
    chart: [ColumnSeries, Category, Legend ]
  }
};
</script>
<style>
  #container {
    height: 350px;
  }
</style>

```

{% previewsample "page.domainurl/code-snippet/chart/axis/category-cs35" %}

### Legend Item Size

You can customize the size of the legend items by using the [shapeHeight](#) and [shapeWidth](#) property.

### APP.VUE

```
<template>
  <div id="app">
    <ejs-chart id="container" :title='title'
:primaryXAxis='primaryXAxis' :legendSettings='legendSettings'>
      <e-series-collection>
        <e-series :dataSource='seriesData' type='Column'
xName='country' yName='gold' name='Gold' legendShape='Circle'> </e-series>
        <e-series :dataSource='seriesData' type='Column'
xName='country' yName='silver' name='Silver' legendShape='Circle'> </e-
series>
        <e-series :dataSource='seriesData' type='Column'
xName='country' yName='bronze' name='Bronze' legendShape='Circle'> </e-
series>
      </e-series-collection>
    </ejs-chart>
  </div>
</template>
<script>
import Vue from "vue";
import { ChartPlugin, ColumnSeries, Category, Legend } from
"@syncfusion/ej2-vue-charts";
Vue.use(ChartPlugin);
export default {
  data() {
    return {
      seriesData: [
        { country: "USA", gold: 50, silver: 70, bronze: 45 },
        { country: "China", gold: 40, silver: 60, bronze: 55 },
        { country: "Japan", gold: 70, silver: 60, bronze: 50 },
        { country: "Australia", gold: 60, silver: 56, bronze: 40 },
        { country: "France", gold: 50, silver: 45, bronze: 35 },
        { country: "Germany", gold: 40, silver: 30, bronze: 22 },
        { country: "Italy", gold: 40, silver: 35, bronze: 37 },
        { country: "Sweden", gold: 30, silver: 25, bronze: 27 }
      ],
      primaryXAxis: {
        valueType: 'Category',
        title: 'Countries'
      },
      legendSettings: {
        visible: true, shapeHeight: 15, shapeWidth: 15
      },
      title: "Olympic Medals"
    };
  },
  provide: {
    chart: [ColumnSeries, Category, Legend ]
  }
};
```



```

</script>
<style>
  #container {
    height: 350px;
  }
</style>

```

{% previewsample "page.domainurl/code-snippet/chart/axis/category-cs36" %}

#### [Paging for Legend](#)

Paging will be enabled by default, when the legend items exceeds the legend bounds. You can view each legend

items by navigating between the pages using navigation buttons.

#### **APP.VUE**

```

<template>
  <div id="app">
    <ejs-chart id="container" :title='title'
:primaryXAxis='primaryXAxis' :primaryYAxis='primaryYAxis'
:legendSettings='legendSettings'>
      <e-series-collection>
        <e-series :dataSource='seriesData' type='Line' xName='x'
yName='y' name='December 2007' width=2 :marker='marker'> </e-series>
        <e-series :dataSource='seriesData' type='Line' xName='x'
yName='y1' name='December 2008' width=2 :marker='marker1'> </e-series>
        <e-series :dataSource='seriesData' type='Line' xName='x'
yName='y2' name='December 2009' width=2 :marker='marker2'> </e-series>
        <e-series :dataSource='seriesData' type='Line' xName='x'
yName='y3' name='December 2010' width=2 :marker='marker3'> </e-series>
      </e-series-collection>
    </ejs-chart>
  </div>
</template>
<script>
import Vue from "vue";
import { ChartPlugin, LineSeries, Category, Legend } from "@syncfusion/ej2-
vue-charts";
Vue.use(ChartPlugin);
export default {
  data() {
    return {
      seriesData: [
        { x: 'WW', y: 12, y1: 22, y2: 38.3, y3: 50 },
        { x: 'EU', y: 9.9, y1: 26, y2: 45.2, y3: 63.6 },
        { x: 'APAC', y: 4.4, y1: 9.3, y2: 18.2, y3: 20.9 },
        { x: 'LATAM', y: 6.4, y1: 28, y2: 46.7, y3: 65.1 },
        { x: 'MEA', y: 30, y1: 45.7, y2: 61.5, y3: 73 },
        { x: 'NA', y: 25.3, y1: 35.9, y2: 64, y3: 81.4 }
      ],
      primaryXAxis: {
        title: 'Countries',
        valueType: 'Category', interval: 1,
        labelIntersectAction : 'Rotate45'
      },
      primaryYAxis: {

```

```

        title: 'Penetration (%)',
        rangePadding: 'None', labelFormat: '{value}%',
        minimum: 0, maximum: 90
    },
    legendSettings: {
        padding: 10, shapePadding: 10,
        visible: true, border: {
            width: 2, color: 'grey'
        },
        width: '200'
    },
    marker: {
        visible: true,
        width: 10, height: 10,
        shape: 'Diamond'
    },
    marker1: {
        visible: true,
        width: 10, height: 10,
        shape: 'Pentagon'
    },
    marker2: {
        visible: true,
        width: 10, height: 10,
        shape: 'Triangle',
    },
    marker3: {
        visible: true,
        width: 10, height: 10,
        shape: 'Circle'
    }
    title: "FB Penetration of Internet Audience"
};
},
provide: {
    chart: [LineSeries, Category, Legend ]
}
};
</script>
<style>
    #container {
        height: 350px;
    }
</style>

```

{% previewsample "page.domainurl/code-snippet/chart/axis/legend-cs1" %}

### Legend Text Wrap

When the legend text exceeds the container, the text can be wrapped by using [textWrap](#) Property. End user can also wrap the legend text based on the [maximumLabelWidth](#) property.

### APP.VUE

```

<template>
    <div id="app">

```

```

<ejs-chart id="container" :title='title'
:primaryXAxis='primaryXAxis' :legendSettings='legendSettings'>
  <e-series-collection>
    <e-series :dataSource='seriesData' type='Column'
xName='country' yName='gold' name='Gold Medals'> </e-series>
    <e-series :dataSource='seriesData' type='Column'
xName='country' yName='silver' name='Silver Medals'> </e-series>
    <e-series :dataSource='seriesData' type='Column'
xName='country' yName='bronze' name='Bronze Medals'> </e-series>
  </e-series-collection>
</ejs-chart>
</div>
</template>
<script>
import Vue from "vue";
import { ChartPlugin, ColumnSeries, Category, Legend } from
"@syncfusion/ej2-vue-charts";
Vue.use(ChartPlugin);
export default {
  data() {
    return {
      seriesData: [
        { country: "USA", gold: 50, silver: 70, bronze: 45 },
        { country: "China", gold: 40, silver: 60, bronze: 55 },
        { country: "Japan", gold: 70, silver: 60, bronze: 50 },
        { country: "Australia", gold: 60, silver: 56, bronze: 40 },
        { country: "France", gold: 50, silver: 45, bronze: 35 },
        { country: "Germany", gold: 40, silver: 30, bronze: 22 },
        { country: "Italy", gold: 40, silver: 35, bronze: 37 },
        { country: "Sweden", gold: 30, silver: 25, bronze: 27 }
      ],
      primaryXAxis: {
        valueType: 'Category',
        title: 'Countries'
      },
      legendSettings: {
        visible: true, position: 'Right', textWrap: 'Wrap',
maximumLabelWidth: 50
      },
      title: "Olympic Medals"
    };
  },
  provide: {
    chart: [ColumnSeries, Category, Legend ]
  }
};
</script>
<style>
#container {
  height: 350px;
}
</style>

```

{% previewsample "page.domainurl/code-snippet/chart/axis/category-cs37" %}

Set the label color based on series color

You can set the legend label color based on series color by using chart's [loaded](#) event.

#### APP.VUE

```
<template>
  <div id="app">
    <ejs-chart :title='title' :primaryXAxis='primaryXAxis'
:legendSettings='legendSettings' :loaded='onChartLoaded'>
      <e-series-collection>
        <e-series :dataSource='seriesData' type='Column'
xName='country' yName='gold' name='Gold'> </e-series>
        <e-series :dataSource='seriesData' type='Column'
xName='country' yName='silver' name='Silver'> </e-series>
        <e-series :dataSource='seriesData' type='Column'
xName='country' yName='bronze' name='Bronze'> </e-series>
      </e-series-collection>
    </ejs-chart>
  </div>
</template>
<script>
import Vue from "vue";
import { ChartPlugin, ColumnSeries, Category, Legend } from
"@syncfusion/ej2-vue-charts";
Vue.use(ChartPlugin);
// declare the series colors
let colors = ['#00BDAE', '#404041', '#357CD2'];
export default {
  data() {
    return {
      seriesData: [
        { country: "USA", gold: 50, silver: 70, bronze: 45 },
        { country: "China", gold: 40, silver: 60, bronze: 55 },
        { country: "Japan", gold: 70, silver: 60, bronze: 50 },
        { country: "Australia", gold: 60, silver: 56, bronze: 40 },
        { country: "France", gold: 50, silver: 45, bronze: 35 },
        { country: "Germany", gold: 40, silver: 30, bronze: 22 },
        { country: "Italy", gold: 40, silver: 35, bronze: 37 },
        { country: "Sweden", gold: 30, silver: 25, bronze: 27 }
      ],
      primaryXAxis: {
        valueType: 'Category',
        title: 'Countries'
      },
      legendSettings: {
        visible: true,
        position: 'Top'
      },
      title: "Olympic Medals"
    };
  },
  provide: {
    chart: [ColumnSeries, Category, Legend]
  },
  methods: {
    onChartLoaded: function(args) {
      let chart = document.querySelector('.e-chart');;
```

```

    let legendTextCol =
chart.querySelectorAll('[id*="chart_legend_text_"]');
    for (let i = 0; i < legendTextCol.length; i++) {
        //set the color to legend label
        legendTextCol[i].setAttribute('fill', colors[i]);
    }
}
}};
</script>

```

{% previewsample "page.domainurl/code-snippet/chart/axis/category-cs38" %}

### Enable Animation

You can customize the animation while clicking legend by setting enableAnimation as true or false using enableAnimation property in chart.

### APP.VUE

```

<template>
  <div id="app">
    <ejs-chart id="container" :title='title'
:primaryXAxis='primaryXAxis' :enableAnimation='enableAnimation'
:legendSettings='legendSettings'>
      <e-series-collection>
        <e-series :dataSource='seriesData' type='Column'
xName='country' yName='gold' name='Gold' legendShape='Circle'> </e-series>
        <e-series :dataSource='seriesData' type='Column'
xName='country' yName='silver' name='Silver' legendShape='Circle'> </e-
series>
        <e-series :dataSource='seriesData' type='Column'
xName='country' yName='bronze' name='Bronze' legendShape='Circle'> </e-
series>
      </e-series-collection>
    </ejs-chart>
  </div>
</template>
<script>
import Vue from "vue";
import { ChartPlugin, ColumnSeries, Category, Legend } from
"@syncfusion/ej2-vue-charts";
Vue.use(ChartPlugin);
export default {
  data() {
    return {
      seriesData: [
        { country: "USA", gold: 50, silver: 70, bronze: 45 },
        { country: "China", gold: 40, silver: 60, bronze: 55 },
        { country: "Japan", gold: 70, silver: 60, bronze: 50 },
        { country: "Australia", gold: 60, silver: 56, bronze: 40 },
        { country: "France", gold: 50, silver: 45, bronze: 35 },
        { country: "Germany", gold: 40, silver: 30, bronze: 22 },
        { country: "Italy", gold: 40, silver: 35, bronze: 37 },
        { country: "Sweden", gold: 30, silver: 25, bronze: 27 }
      ],
      primaryXAxis: {
        valueType: 'Category',

```

```

        title: 'Countries'
      },
      legendSettings: {
        visible: true, toggleVisibility: true
      }
    },
    title: "Olympic Medals",
    enableAnimation: true
  };
};
provide: {
  chart: [ColumnSeries, Category, Legend ]
}
};
</script>
<style>
#container {
  height: 350px;
}
</style>

```

{% previewsample "page.domainurl/code-snippet/chart/axis/category-cs39" %}

### Collapsing Legend Item

By default, series name will be displayed as legend. To skip the legend for a particular series, you can give empty string to the series name.

### APP.VUE

```

<template>
  <div id="app">
    <ejs-chart id="container" :title='title'
    :primaryXAxis='primaryXAxis' :legendSettings='legendSettings'>
      <e-series-collection>
        <e-series :dataSource='seriesData' type='Column'
        xName='country' yName='gold' name='Gold' legendShape='Circle'> </e-series>
        <e-series :dataSource='seriesData' type='Column'
        xName='country' yName='silver' name='Silver' legendShape='Circle'> </e-
        series>
        <e-series :dataSource='seriesData' type='Column'
        xName='country' yName='bronze' name='Bronze' legendShape='Circle'> </e-
        series>
      </e-series-collection>
    </ejs-chart>
  </div>
</template>
<script>
import Vue from "vue";
import { ChartPlugin, ColumnSeries, Category, Legend } from
"@syncfusion/ej2-vue-charts";
Vue.use(ChartPlugin);
export default {
  data() {
    return {
      seriesData: [
        { country: "USA", gold: 50, silver: 70, bronze: 45 },
        { country: "China", gold: 40, silver: 60, bronze: 55 },

```

```

        { country: "Japan", gold: 70, silver: 60, bronze: 50 },
        { country: "Australia", gold: 60, silver: 56, bronze: 40 },
        { country: "France", gold: 50, silver: 45, bronze: 35 },
        { country: "Germany", gold: 40, silver: 30, bronze: 22 },
        { country: "Italy", gold: 40, silver: 35, bronze: 37 },
        { country: "Sweden", gold: 30, silver: 25, bronze: 27 }
    ],
    primaryXAxis: {
        valueType: 'Category',
        title: 'Countries'
    },
    legendSettings: {
        visible: true, toggleVisibility: true
    },
    title: "Olympic Medals"
};
},
provide: {
    chart: [ColumnSeries, Category, Legend ]
}
};
</script>
<style>
    #container {
        height: 350px;
    }
</style>

```

{% previewsample "page.domainurl/code-snippet/chart/axis/category-cs40" %}

### Legend Title

You can set title for legend using `title` property in `legendSettings`. You can also customize the `fontStyle`, `size`, `fontWeight`, `color`, `textAlignment`, `fontFamily`, `opacity` and `textOverflow` of legend title. `titlePosition` is used to set the legend position in `Top`, `Left` and `Right` position. `maximumTitleWidth` is used to set the width of the legend title. By default, it will be `100px`.

### APP.VUE

```

<template>
    <div id="app">
        <ejs-chart id="container" :title='title'
        :primaryXAxis='primaryXAxis' :legendSettings='legendSettings'>
            <e-series-collection>
                <e-series :dataSource='seriesData' type='Column'
                xName='country' yName='gold' name='Gold'> </e-series>
                <e-series :dataSource='seriesData' type='Column'
                xName='country' yName='silver' name='Silver'> </e-series>
                <e-series :dataSource='seriesData' type='Column'
                xName='country' yName='bronze' name='Bronze'> </e-series>
            </e-series-collection>
        </ejs-chart>
    </div>
</template>
<script>
import Vue from "vue";

```

```

import { ChartPlugin, ColumnSeries, Category, Legend } from
"@syncfusion/ej2-vue-charts";
Vue.use(ChartPlugin);
export default {
  data() {
    return {
      seriesData: [
        { country: "USA", gold: 50, silver: 70, bronze: 45 },
        { country: "China", gold: 40, silver: 60, bronze: 55 },
        { country: "Japan", gold: 70, silver: 60, bronze: 50 },
        { country: "Australia", gold: 60, silver: 56, bronze: 40 },
        { country: "France", gold: 50, silver: 45, bronze: 35 },
        { country: "Germany", gold: 40, silver: 30, bronze: 22 },
        { country: "Italy", gold: 40, silver: 35, bronze: 37 },
        { country: "Sweden", gold: 30, silver: 25, bronze: 27 }
      ],
      primaryXAxis: {
        valueType: 'Category',
      },
      legendSettings: {
        visible: true, title: 'Countries'
      },
      title: "Olympic Medals"
    };
  },
  provide: {
    chart: [ColumnSeries, Category, Legend ]
  }
};
</script>
<style>
  #container {
    height: 350px;
  }
</style>

```

{% previewsample "page.domainurl/code-snippet/chart/axis/category-cs41" %}

### Arrow Page Navigation

By default, the page number will be enabled while legend paging. Now, you can disable that page number and also you can get left and right arrows for page navigation. You have to set `false` value to `enablePages` to get this support.

### APP.VUE

```

<template>
  <div id="app">
    <ejs-chart id="container" :title='title'
:primaryXAxis='primaryXAxis' :primaryYAxis='primaryYAxis'
:legendSettings='legendSettings'>
      <e-series-collection>
        <e-series :dataSource='seriesData' type='Line' xName='x'
yName='y' name='December 2007' width=2 :marker='marker'> </e-series>
        <e-series :dataSource='seriesData' type='Line' xName='x'
yName='y1' name='December 2008' width=2 :marker='marker1'> </e-series>
      </e-series-collection>
    </div>
  </template>

```



```

        <e-series :dataSource='seriesData' type='Line' xName='x'
yName='y2' name='December 2009' width=2 :marker='marker2'> </e-series>
        <e-series :dataSource='seriesData' type='Line' xName='x'
yName='y3' name='December 2010' width=2 :marker='marker3'> </e-series>
    </e-series-collection>
</ejs-chart>
</div>
</template>
<script>
import Vue from "vue";
import { ChartPlugin, LineSeries, Category, Legend } from "@syncfusion/ej2-
vue-charts";
Vue.use(ChartPlugin);
export default {
    data() {
        return {
            seriesData: [
                { x: 'WW', y: 12, y1: 22, y2: 38.3, y3: 50 },
                { x: 'EU', y: 9.9, y1: 26, y2: 45.2, y3: 63.6 },
                { x: 'APAC', y: 4.4, y1: 9.3, y2: 18.2, y3: 20.9 },
                { x: 'LATAM', y: 6.4, y1: 28, y2: 46.7, y3: 65.1 },
                { x: 'MEA', y: 30, y1: 45.7, y2: 61.5, y3: 73 },
                { x: 'NA', y: 25.3, y1: 35.9, y2: 64, y3: 81.4 }
            ],
            primaryXAxis: {
                title: 'Countries',
                valueType: 'Category', interval: 1,
                labelIntersectAction : 'Rotate45'
            },
            primaryYAxis: {
                title: 'Penetration (%)',
                rangePadding: 'None', labelFormat: '{value}%',
                minimum: 0, maximum: 90
            },
            legendSettings: {
                width: '180',
                enablePages: false
            },
            marker: {
                visible: true,
                width: 10, height: 10,
                shape: 'Diamond'
            },
            marker1: {
                visible: true,
                width: 10, height: 10,
                shape: 'Pentagon'
            },
            marker2: {
                visible: true,
                width: 10, height: 10,
                shape: 'Triangle',
            },
            marker3: {
                visible: true,
                width: 10, height: 10,
                shape: 'Circle'
            }
        }
    }
}

```

```

    }
    title: "FB Penetration of Internet Audience"
  };
},
provide: {
  chart: [LineSeries, Category, Legend ]
}
};
</script>
<style>
#container {
  height: 350px;
}
</style>

```

{% previewsample "page.domainurl/code-snippet/chart/axis/legend-cs2" %}

### Legend Item Padding

The [itemPadding](#) property can be used to adjust the space between the legend items.

### APP.VUE

```

<template>
  <div id="app">
    <ejs-chart id="container" :title='title'
:primaryXAxis='primaryXAxis' :legendSettings='legendSettings'>
      <e-series-collection>
        <e-series :dataSource='seriesData' type='Column'
xName='country' yName='gold' name='Gold'> </e-series>
        <e-series :dataSource='seriesData' type='Column'
xName='country' yName='silver' name='Silver'> </e-series>
        <e-series :dataSource='seriesData' type='Column'
xName='country' yName='bronze' name='Bronze'> </e-series>
      </e-series-collection>
    </ejs-chart>
  </div>
</template>
<script>
import Vue from "vue";
import { ChartPlugin, ColumnSeries, Category, Legend } from
"@syncfusion/ej2-vue-charts";
Vue.use(ChartPlugin);
export default {
  data() {
    return {
      seriesData: [
        { country: "USA", gold: 50, silver: 70, bronze: 45 },
        { country: "China", gold: 40, silver: 60, bronze: 55 },
        { country: "Japan", gold: 70, silver: 60, bronze: 50 },
        { country: "Australia", gold: 60, silver: 56, bronze: 40 },
        { country: "France", gold: 50, silver: 45, bronze: 35 },
        { country: "Germany", gold: 40, silver: 30, bronze: 22 },
        { country: "Italy", gold: 40, silver: 35, bronze: 37 },
        { country: "Sweden", gold: 30, silver: 25, bronze: 27 }
      ],
      primaryXAxis: {

```

```

        valueType: 'Category',
        title: 'Countries'
    },
    legendSettings: {
        visible: true,
        position: 'Top',
        itemPadding: 30
    },
    title: "Olympic Medals"
  };
},
provide: {
  chart: [ColumnSeries, Category, Legend ]
}
};
</script>
<style>
  #container {
    height: 350px;
  }
</style>

```

{% previewsample "page.domainurl/code-snippet/chart/axis/category-cs42" %}

Note: To use legend feature, we need to inject **Legend** into the **Provide**.

### Tooltip in Vue Chart component

<!-- markdownlint-disable MD036 -->

Chart will display details about the points through tooltip, when the mouse is moved over the point.

#### Default tooltip

By default, tooltip is not visible. You can enable the tooltip by setting [enable](#) property to **true** and by injecting **Tooltip** into the **provide**.

#### APP.VUE

```

<template>
  <div id="app">
    <ejs-chart id="container" :title='title'
    :primaryXAxis='primaryXAxis' :tooltip='tooltip'>
      <e-series-collection>
        <e-series :dataSource='seriesData' type='StepLine' xName='x'
        yName='y' name='China' :marker='marker'> </e-series>
      </e-series-collection>
    </ejs-chart>
  </div>
</template>
<script>
import Vue from "vue";
import { ChartPlugin, StepLineSeries, DateTime, Tooltip } from
"@syncfusion/ej2-vue-charts";
Vue.use(ChartPlugin);
export default {
  data() {
    return {

```

```

seriesData:[
    { x: new Date(1975, 0, 1), y: 16, y1: 10, y2: 4.5 },
    { x: new Date(1980, 0, 1), y: 12.5, y1: 7.5, y2: 5 },
    { x: new Date(1985, 0, 1), y: 19, y1: 11, y2: 6.5 },
    { x: new Date(1990, 0, 1), y: 14.4, y1: 7, y2: 4.4 },
    { x: new Date(1995, 0, 1), y: 11.5, y1: 8, y2: 5 },
    { x: new Date(2000, 0, 1), y: 14, y1: 6, y2: 1.5 },
    { x: new Date(2005, 0, 1), y: 10, y1: 3.5, y2: 2.5 },
    { x: new Date(2010, 0, 1), y: 16, y1: 7, y2: 3.7 }
],
primaryXAxis: {
    valueType: 'DateTime'
},
marker: {
    visible: true, width: 10, height: 10
},
tooltip: {enable: true},
title: "Unemployment Rates 1975-2010"
};
},
provide: {
    chart: [StepLineSeries, DateTime, Tooltip]
},
};
</script>
<style>
#container{
    height: 350px;
}
</style>

```

{% previewsample "page.domainurl/code-snippet/chart/user-interaction/tooltip-cs1" %}

<!-- markdownlint-disable MD013 -->

### Fixed tooltip

By default, tooltip track the mouse movement, but you can set a fixed position for the tooltip by using the [location](#) property.

### APP.VUE

```

<template>
  <div id="app">
    <ejs-chart id="container" :title='title'
:primaryXAxis='primaryXAxis' :tooltip='tooltip'>
      <e-series-collection>
        <e-series :dataSource='seriesData' type='StepLine' xName='x'
yName='y' name='China' :marker='marker'> </e-series>
      </e-series-collection>
    </ejs-chart>
  </div>
</template>
<script>
import Vue from "vue";
import { ChartPlugin, StepLineSeries, DateTime, Tooltip } from
"@syncfusion/ej2-vue-charts";

```

```

Vue.use(ChartPlugin);
export default {
  data() {
    return {
      seriesData:[
        { x: new Date(1975, 0, 1), y: 16, y1: 10, y2: 4.5 },
        { x: new Date(1980, 0, 1), y: 12.5, y1: 7.5, y2: 5 },
        { x: new Date(1985, 0, 1), y: 19, y1: 11, y2: 6.5 },
        { x: new Date(1990, 0, 1), y: 14.4, y1: 7, y2: 4.4 },
        { x: new Date(1995, 0, 1), y: 11.5, y1: 8, y2: 5 },
        { x: new Date(2000, 0, 1), y: 14, y1: 6, y2: 1.5 },
        { x: new Date(2005, 0, 1), y: 10, y1: 3.5, y2: 2.5 },
        { x: new Date(2010, 0, 1), y: 16, y1: 7, y2: 3.7 }
      ],
      primaryXAxis: {
        valueType: 'DateTime'
      },
      marker: {
        visible: true, width: 10, height: 10
      },
      tooltip: { enable: true, location: { x: 120, y: 20 } },
      title: "Unemployment Rates 1975-2010"
    };
  },
  provide: {
    chart: [StepLineSeries, DateTime, Tooltip]
  },
};
</script>
<style>
  #container{
    height: 350px;
  }
</style>

```

{% previewsample "page.domainurl/code-snippet/chart/user-interaction/tooltip-cs7" %}

### Format the tooltip

<!-- markdownlint-disable MD013 -->

By default, tooltip shows information of x and y value in points. In addition to that, you can show more information in tooltip. For example the format `${series.name} ${point.x}` shows series name and point x value.

### APP.VUE

```

<template>
  <div id="app">
    <ejs-chart id="container" :title='title'
    :primaryXAxis='primaryXAxis' :tooltip='tooltip'>
      <e-series-collection>
        <e-series :dataSource='seriesData' type='StepLine' xName='x'
        yName='y' name='China' :marker='marker'> </e-series>
      </e-series-collection>
    </div>
  </template>

```

```

        </ejs-chart>
    </div>
</template>
<script>
import Vue from "vue";
import { ChartPlugin, StepLineSeries, DateTime, Tooltip } from
"@syncfusion/ej2-vue-charts";
Vue.use(ChartPlugin);
export default {
  data() {
    return {
      seriesData:[
        { x: new Date(1975, 0, 1), y: 16, y1: 10, y2: 4.5 },
        { x: new Date(1980, 0, 1), y: 12.5, y1: 7.5, y2: 5 },
        { x: new Date(1985, 0, 1), y: 19, y1: 11, y2: 6.5 },
        { x: new Date(1990, 0, 1), y: 14.4, y1: 7, y2: 4.4 },
        { x: new Date(1995, 0, 1), y: 11.5, y1: 8, y2: 5 },
        { x: new Date(2000, 0, 1), y: 14, y1: 6, y2: 1.5 },
        { x: new Date(2005, 0, 1), y: 10, y1: 3.5, y2: 2.5 },
        { x: new Date(2010, 0, 1), y: 16, y1: 7, y2: 3.7 }
      ],
      primaryXAxis: {
        valueType: 'DateTime'
      },
      marker: {
        visible: true, width: 10, height: 10
      },
      tooltip: { enable: true, header: 'Unemployment', format:
'<b>${point.x} : ${point.y}</b>',
        title: "Unemployment Rates 1975-2010"
      };
    },
    provide: {
      chart: [StepLineSeries, DateTime, Tooltip]
    },
  };
</script>
<style>
#container{
  height: 350px;
}
</style>

```

{% previewsample "page.domainurl/code-snippet/chart/user-interaction/tooltip-cs2" %}

<!-- markdownlint-disable MD013 -->

### Individual series format

<!-- markdownlint-disable MD013 -->

You can format the each series tooltip separately using series [tooltipFormat](#) property.

Note: If series [tooltipFormat](#) is given, it shows the tooltip for that series in that format, or else it will take tooltip format.

### APP.VUE

```

<template>
  <div id="app">
    <ejs-chart id="container" :title='title'
:primaryXAxis='primaryXAxis' :tooltip='tooltip'>
      <e-series-collection>
        <e-series :dataSource='seriesData1' type='Spline' xName='x'
yName='y' name='Max Temp' :marker='marker' tooltipFormat='{point.x}'> </e-
series>
        <e-series :dataSource='seriesData2' type='Spline' xName='x'
yName='y' name='Avg Temp' :marker='marker' tooltipFormat='{point.y}'> </e-
series>
        <e-series :dataSource='seriesData3' type='Spline' xName='x'
yName='y' name='Min Temp' :marker='marker'> </e-series>
      </e-series-collection>
    </ejs-chart>
  </div>
</template>
<script>
import Vue from "vue";
import { ChartPlugin, SplineSeries, Category, Tooltip } from
"@syncfusion/ej2-vue-charts";
Vue.use(ChartPlugin);
export default {
  data() {
    return {
      seriesData1:[
        { x: 'Sun', y: 15 }, { x: 'Mon', y: 22 },
        { x: 'Tue', y: 32 },
        { x: 'Wed', y: 31 },
        { x: 'Thu', y: 29 }, { x: 'Fri', y: 24 },
        { x: 'Sat', y: 18 },
      ],
      seriesData2:[
        { x: 'Sun', y: 10 }, { x: 'Mon', y: 18 },
        { x: 'Tue', y: 28 },
        { x: 'Wed', y: 28 },
        { x: 'Thu', y: 26 }, { x: 'Fri', y: 20 },
        { x: 'Sat', y: 15 }
      ],
      seriesData3:[
        { x: 'Sun', y: 2 }, { x: 'Mon', y: 12 },
        { x: 'Tue', y: 22 },
        { x: 'Wed', y: 23 },
        { x: 'Thu', y: 19 }, { x: 'Fri', y: 13 },
        { x: 'Sat', y: 8 },
      ],
      primaryXAxis: {
        valueType: 'Category'
      },
      marker: {
        visible: true, width: 10, height: 10
      },
      tooltip: { enable: true, header: 'Unemployment', format:
'<b>${point.x} : ${point.y}</b>',
        title: "NC Weather Report - 2016"
      };
    };
  },

```

```

    provide: {
      chart: [SplineSeries, Category, Tooltip]
    },
  };
</script>
<style>
  #container{
    height: 350px;
  }
</style>

```

{% previewsample "page.domainurl/code-snippet/chart/user-interaction/tooltip-cs3" %}

<!-- markdownlint-disable MD013 -->

### Tooltip template

Any HTML elements can be displayed in the tooltip by using the [template](#) property of the tooltip. You can use the  $\{x\}$  and  $\{y\}$  as place holders in the HTML element to display the x and y values of the corresponding data point.

### APP.VUE

```

<template>
  <div id="app">
    <ejs-chart id="container" :title='title'
    :primaryXAxis='primaryXAxis' :tooltip='tooltip'>
      <e-series-collection>
        <e-series :dataSource='seriesData' type='StepLine' xName='x'
        yName='y' name='China' :marker='marker'> </e-series>
      </e-series-collection>
    </ejs-chart>
  </div>
</template>
<script>
import Vue from "vue";
import { ChartPlugin, StepLineSeries, DateTime, Tooltip } from
"@syncfusion/ej2-vue-charts";
var TemplateVue = Vue.component('contentTemplate', {
  template: `<div>
    <table style="width:100%; border: 1px solid black;">
      <tr><th colspan="2">
bgcolor="#00FFFF">Unemployment</th></tr>
      <tr><td bgcolor="#00FFFF">{{data.x}}:</td><td
bgcolor="#00FFFF">{{data.y}}</td></tr>
    </table></div> `,
  data() {
    return {
      data: {}
    };
  }
});
let contentTemplate = function() {
  return { template: TemplateVue };
};
Vue.use(ChartPlugin);
export default {

```



```

data() {
  return {
    seriesData:[
      { x: new Date(1975, 0, 1), y: 16, y1: 10, y2: 4.5 },
      { x: new Date(1980, 0, 1), y: 12.5, y1: 7.5, y2: 5 },
      { x: new Date(1985, 0, 1), y: 19, y1: 11, y2: 6.5 },
      { x: new Date(1990, 0, 1), y: 14.4, y1: 7, y2: 4.4 },
      { x: new Date(1995, 0, 1), y: 11.5, y1: 8, y2: 5 },
      { x: new Date(2000, 0, 1), y: 14, y1: 6, y2: 1.5 },
      { x: new Date(2005, 0, 1), y: 10, y1: 3.5, y2: 2.5 },
      { x: new Date(2010, 0, 1), y: 16, y1: 7, y2: 3.7 }
    ],
    primaryXAxis: {
      valueType: 'DateTime'
    },
    marker: {
      visible: true, width: 10, height: 10
    },
    tooltip: { enable: true, template: contentTemplate },
    title: "Unemployment Rates 1975-2010"
  };
},
provide: {
  chart: [StepLineSeries, DateTime, Tooltip]
},
};
</script>
<style>
#container{
  height: 350px;
}
</style>

```

{% previewsample "page.domainurl/code-snippet/chart/user-interaction/tooltip-cs4" %}

### Customize the appearance of tooltip

The [fill](#) and [border](#) properties are used to customize the background color and border of the tooltip respectively. The [textStyle](#) property in the tooltip is used to customize the font of the tooltip text. The [highlightColor](#) property is used to customize the point color while hovering for tooltip.

### APP.VUE

```

<template>
  <div id="app">
    <ejs-chart
      id="container"
      :title="title"
      :primaryXAxis="primaryXAxis"
      :tooltip="tooltip"
      :highlightColor="highlightColor"
    >
      <e-series-collection>
        <e-series
          :dataSource="seriesData"
          type="Column"
          xName="x"

```

```

        yName="y"
        name="China"
        :marker="marker"
      >
    </e-series>
  </e-series-collection>
</ejs-chart>
</div>
</template>
<script>
import Vue from "vue";
import {
  ChartPlugin,
  ColumnSeries,
  DateTime,
  Tooltip,
} from "@syncfusion/ej2-vue-charts";
Vue.use(ChartPlugin);
export default {
  data() {
    return {
      seriesData: [
        { x: new Date(1975, 0, 1), y: 16, y1: 10, y2: 4.5 },
        { x: new Date(1980, 0, 1), y: 12.5, y1: 7.5, y2: 5 },
        { x: new Date(1985, 0, 1), y: 19, y1: 11, y2: 6.5 },
        { x: new Date(1990, 0, 1), y: 14.4, y1: 7, y2: 4.4 },
        { x: new Date(1995, 0, 1), y: 11.5, y1: 8, y2: 5 },
        { x: new Date(2000, 0, 1), y: 14, y1: 6, y2: 1.5 },
        { x: new Date(2005, 0, 1), y: 10, y1: 3.5, y2: 2.5 },
        { x: new Date(2010, 0, 1), y: 16, y1: 7, y2: 3.7 },
      ],
      primaryXAxis: {
        valueType: "DateTime",
      },
      marker: {
        visible: true,
        width: 10,
        height: 10,
      },
      tooltip: {
        enable: true,
        format: "${series.name} ${point.x} : ${point.y}",
        fill: "#7bb4eb",
        border: {
          width: 2,
          color: "grey",
        },
      },
      title: "Unemployment Rates 1975-2010",
      highlightColor: "red",
    };
  },
  provide: {
    chart: [ColumnSeries, DateTime, Tooltip],
  },
};
</script>

```

```
<style>
#container {
  height: 350px;
}
</style>
```

{% previewsample "page.domainurl/code-snippet/chart/user-interaction/tooltip-cs5" %}

### Tooltip mapping name

By default, tooltip shows information of x and y value in points. You can show more information from data source in tooltip by using the [tooltipMappingName](#) property of the tooltip. You can use the ``${point.tooltip}`` as place holders to display the specified tooltip content.

### APP.VUE

```
<template>
  <div id="app">
    <ejs-chart id="container" :title='title'
      :primaryXAxis='primaryXAxis' :tooltip='tooltip'>
      <e-series-collection>
        <e-series :dataSource='seriesData' type='Column' xName='x'
          yName='y' tooltipMappingName='text' :marker='marker'> </e-series>
      </e-series-collection>
    </ejs-chart>
  </div>
</template>
<script>
import Vue from "vue";
import { ChartPlugin, ColumnSeries, Category, Tooltip } from
"@syncfusion/ej2-vue-charts";
Vue.use(ChartPlugin);
export default {
  data() {
    return {
      seriesData:[
        { x: 'Germany', y: 72, text: 'GER: 72'},
        { x: 'Russia', y: 103.1, text: 'RUS: 103.1'},
        { x: 'Brazil', y: 139.1, text: 'BRZ: 139.1'},
        { x: 'India', y: 462.1, text: 'IND: 462.1'},
        { x: 'China', y: 721.4, text: 'CHN: 721.4'},
        { x: 'United States Of America', y: 286.9, text: 'USA:
286.9'},
        { x: 'Great Britain', y: 115.1, text: 'GBR: 115.1'},
        { x: 'Nigeria', y: 97.2, text: 'NGR: 97.2'},
      ],
      primaryXAxis: {
        valueType: 'Category'
      },
      marker: {
        visible: true, width: 10, height: 10
      },
      tooltip: {enable: true, format: `${point.tooltip}`},
      title: 'Internet Users in Million - 2016',
    };
  },
  provide: {
```

```

    chart: [ColumnSeries, Category, Tooltip]
  },
};
</script>
<style>
#container{
  height: 350px;
}
</style>

```

{% previewsample "page.domainurl/code-snippet/chart/user-interaction/tooltip-cs6" %}

## Zooming in Vue Chart component

### Enable zooming

Chart can be zoomed in three ways.

- Selection - By setting [enableSelectionZooming](#) property to true in `zoomSettings`, you can zoom the chart by using the rubber band selection.
- Mousewheel - By setting [enableMouseWheelZooming](#) property to true in `zoomSettings`, you can zoom in and zoom out the chart by scrolling the mouse wheel.
- Pinch - By setting [enablePinchZooming](#) property to true in `zoomSettings`, you can zoom the chart through pinch gesture in touch enabled devices.

Pinch zooming is supported only in browsers that support multi-touch gestures. Currently IE11, Chrome and Opera

browsers support multi-touch in desktop devices.

### APP.VUE

```

<template>
  <div id="app">
    <ejs-chart id="container" :title='title'
:primaryXAxis='primaryXAxis' :zoomSettings='zoom' :legendSettings='legend'>
      <e-series-collection>
        <e-series :dataSource='seriesData1' type='Area' xName='x'
yName='y' name='Product X' :border='border' :animation='animation'
opacity=0.3> </e-series>
      </e-series-collection>
    </ejs-chart>
  </div>
</template>
<script>
import Vue from "vue";
import { ChartPlugin, AreaSeries, DateTime, Zoom } from "@syncfusion/ej2-
vue-charts";
Vue.use(ChartPlugin);
let series1: Object[] = [];
let point1: Object;
let value: number = 40;
let i: number;
for (i = 1; i < 500; i++) {
  if (Math.random() > .5) {
    value += Math.random();

```

```

    } else {
      value -= Math.random();
    }
    point1 = { x: new Date(1950, i + 2, i), y: value.toFixed(1) };
    series1.push(point1);
  }
  export default {
    data() {
      return {
        seriesData1: series1,
        primaryXAxis: {
          valueType: 'DateTime',
          labelFormat: 'yMMM'
        },
        zoom: {
          enableMouseWheelZooming: true,
          enablePinchZooming: true,
          enableSelectionZooming: true
        },
        title: 'Sales History of Product X',
        legend: { visible: false },
        border: { width: 0.5, color: '#00bdae' },
        animation: { enable: false }
      };
    },
    provide: {
      chart: [AreaSeries, DateTime, Zoom]
    },
  };
</script>
<style>
  #container {
    height: 350px;
  }
</style>

```

{% previewsample "page.domainurl/code-snippet/chart/user-interaction/zoom-cs2" %}

Note: To use zooming feature, we need to inject **Zoom** into the **provide**.

After zooming the chart, a zooming toolbar will appear with **zoom**, **zoomin**, **zoomout**, **pan** and **reset** buttons.

Selecting the Pan option will allow to pan the chart and selecting the Reset option will reset the zoomed chart.

### Modes

The [mode](#) property in zoomSettings specifies whether the chart is

allowed to scale along the horizontal axis or vertical axis. The default value of the mode is XY (both axis).

There are three types of mode.

- X - Allows us to zoom the chart horizontally.
- Y - Allows us to zoom the chart vertically.

- XY - Allows us to zoom the chart both vertically and horizontally.

**APP.VUE**

```

<template>
  <div id="app">
    <ejs-chart id="container" :title='title'
:primaryXAxis='primaryXAxis' :zoomSettings='zoom' :legendSettings='legend'>
      <e-series-collection>
        <e-series :dataSource='seriesData1' type='Area' xName='x'
yName='y' name='Product X' :border='border' :animation='animation'
          opacity=0.3> </e-series>
      </e-series-collection>
    </ejs-chart>
  </div>
</template>
<script>
import Vue from "vue";
import { ChartPlugin, AreaSeries, DateTime, Zoom } from "@syncfusion/ej2-
vue-charts";
Vue.use(ChartPlugin);
let series1: Object[] = [];
let point1: Object;
let value: number = 40;
let i: number;
for (i = 1; i < 500; i++) {
  if (Math.random() > .5) {
    value += Math.random();
  } else {
    value -= Math.random();
  }
  point1 = { x: new Date(1950, i + 2, i), y: value.toFixed(1) };
  series1.push(point1);
}
export default {
  data() {
    return {
      seriesData1: series1,
      primaryXAxis: {
        valueType: 'DateTime',
        labelFormat: 'yMMM'
      },
      zoom: {
        enableSelectionZooming: true,
        mode: 'X'
      },
      title: 'Sales History of Product X',
      legend: { visible: false },
      border: { width: 0.5, color: '#00bdae' },
      animation: { enable: false }
    };
  },
  provide: {
    chart: [AreaSeries, DateTime, Zoom]
  },

```

```
};
</script>
<style>
  #container {
    height: 350px;
  }
</style>
```

```
{% previewsample "page.domainurl/code-snippet/chart/user-interaction/zoom-cs3" %}
```

### Toolbar

By default, zoomin, zoomout, pan and reset buttons will be displayed for zoomed chart. You can customize to show the desired options in the toolbar using the [toolbarItems](#) property. Also using the [showToolbar](#) property, you can show toolkit for zooming and panning the chart during initial rendering itself.

### APP.VUE

```
<template>
  <div id="app">
    <ejs-chart id="container" :title='title'
:primaryXAxis='primaryXAxis' :zoomSettings='zoom'>
      <e-series-collection>
        <e-series :dataSource='seriesData1' type='Area' xName='x'
yName='y' name='Product X' opacity=0.3> </e-series>
      </e-series-collection>
    </ejs-chart>
  </div>
</template>
<script>
import Vue from "vue";
import { ChartPlugin, AreaSeries, DateTime, Zoom } from "@syncfusion/ej2-
vue-charts";
Vue.use(ChartPlugin);
let series1: Object[] = [];
let point1: Object;
let value: number = 40;
let i: number;
for (i = 1; i < 500; i++) {
  if (Math.random() > .5) {
    value += Math.random();
  } else {
    value -= Math.random();
  }
  point1 = { x: new Date(1950, i + 2, i), y: value.toFixed(1) };
  series1.push(point1);
}
export default {
  data() {
    return {
      seriesData1: series1,
      primaryXAxis: {
        valueType: 'DateTime',
        labelFormat: 'yMMM'
      },
      zoom:

```

```

        {
            enableSelectionZooming: true,
            toolbarItems: ['Zoom', 'Pan', 'Reset'],
            showToolbar: true
        },
        title: 'Sales History of Product X'
    };
},
provide: {
    chart: [AreaSeries, DateTime, Zoom]
},
};
</script>
<style>
    #container {
        height: 350px;
    }
</style>

```

{% previewsample "page.domainurl/code-snippet/chart/user-interaction/zoom-cs4" %}

### Enable pan

Using [enablePan](#) property you can able to pan the zoomed chart without help of toolbar items.

### APP.VUE

```

<template>
    <div id="app">
        <ejs-chart id="container" :title='title'
        :primaryXAxis='primaryXAxis' :zoomSettings='zoom'>
            <e-series-collection>
                <e-series :dataSource='seriesData1' type='Area' xName='x'
                yName='y' name='Product X' opacity=0.3> </e-series>
            </e-series-collection>
        </ejs-chart>
    </div>
</template>
<script>
import Vue from "vue";
import { ChartPlugin, AreaSeries, DateTime, Zoom } from "@syncfusion/ej2-
vue-charts";
Vue.use(ChartPlugin);
let series1: Object[] = [];
let point1: Object;
let value: number = 40;
let i: number;
for (i = 1; i < 500; i++) {
    if (Math.random() > .5) {
        value += Math.random();
    } else {
        value -= Math.random();
    }
    point1 = { x: new Date(1950, i + 2, i), y: value.toFixed(1) };
    series1.push(point1);
}
export default {

```



```

data() {
  return {
    seriesData1: series1,
    primaryXAxis: {
      valueType: 'DateTime',
      labelFormat: 'yMMM',
      zoomFactor: 0.2, zoomPosition: 0.6
    },
    zoom: {
      enableSelectionZooming: true,
      enablePan: true
    },
    title: 'Sales History of Product X'
  };
},
provide: {
  chart: [AreaSeries, DateTime, Zoom]
},
};
</script>
<style>
  #container {
    height: 350px;
  }
</style>

```

{% previewsample "page.domainurl/code-snippet/chart/user-interaction/zoom-cs5" %}

#### Enable scrollbar

Using the [enableScrollbar](#) property, you can add a scrollbar to a zoomed chart. This scrollbar allows you to zoom or pan the chart. The appearance of the scrollbar can be customized using properties in [scrollbarSettings](#). For example, you can use [trackColor](#) and [trackRadius](#) properties to customize the track of the scrollbar, and [scrollbarRadius](#) and [scrollbarColor](#) properties to customize the scroller. The ability to zoom through the scrollbar can be enabled or disabled using the [enableZoom](#) property in [scrollbarSettings](#). Additionally, you can change the color of the grip and height of the scrollbar using the [gripColor](#) and [height](#) properties.

#### APP.VUE

```

<template>
  <div id="app">
    <ejs-chart id="container" :title='title'
    :primaryXAxis='primaryXAxis' :zoomSettings='zoom'>
      <e-series-collection>
        <e-series :dataSource='seriesData1' type='Area' xName='x'
        yName='y' name='Product X' opacity=0.3> </e-series>
      </e-series-collection>
    </ejs-chart>
  </div>
</template>
<script>
import Vue from "vue";
import { ChartPlugin, AreaSeries, DateTime, Zoom, ScrollBar } from
"@syncfusion/ej2-vue-charts";

```

```

Vue.use(ChartPlugin);
let series1: Object[] = [];
let point1: Object;
let value: number = 40;
let i: number;
for (i = 1; i < 500; i++) {
  if (Math.random() > .5) {
    value += Math.random();
  } else {
    value -= Math.random();
  }
  point1 = { x: new Date(1950, i + 2, i), y: value.toFixed(1) };
  series1.push(point1);
}
export default {
  data() {
    return {
      seriesData1: series1,
      primaryXAxis: {
        valueType: 'DateTime',
        labelFormat: 'yMMM',
        zoomFactor: 0.2, zoomPosition: 0.6,
        scrollbarSettings: {
          enable: true,
          enableZoom: false,
          height: 14,
          trackRadius: 8,
          scrollbarRadius: 8,
          gripColor: 'transparent',
          trackColor: 'yellow',
          scrollbarColor: 'red'
        }
      },
      zoom: {
        enableSelectionZooming: true,
        enableScrollbar: true
      },
      title: 'Sales History of Product X'
    };
  },
  provide: {
    chart: [AreaSeries, DateTime, Zoom, ScrollBar]
  },
};
</script>
<style>
#container {
  height: 350px;
}
</style>

```

{% previewsample "page.domainurl/code-snippet/chart/user-interaction/zoom-cs6" %}

Auto interval on zooming

By using [enableAutoIntervalOnZooming](#) property,

the axis interval will get calculated automatically with respect to the zoomed range.

### APP.VUE

```
<template>
  <div id="app">
    <ejs-chart id="container" :title='title'
:primaryXAxis='primaryXAxis' :zoomSettings='zoom'>
      <e-series-collection>
        <e-series :dataSource='seriesData1' type='Area' xName='x'
yName='y' name='Product X' opacity=0.3> </e-series>
      </e-series-collection>
    </ejs-chart>
  </div>
</template>
<script>
import Vue from "vue";
import { ChartPlugin, AreaSeries, DateTime, Zoom } from "@syncfusion/ej2-
vue-charts";
Vue.use(ChartPlugin);
let series1: Object[] = [];
let point1: Object;
let value: number = 40;
let i: number;
for (i = 1; i < 500; i++) {
  if (Math.random() > .5) {
    value += Math.random();
  } else {
    value -= Math.random();
  }
  point1 = { x: new Date(1950, i + 2, i), y: value.toFixed(1) };
  series1.push(point1);
}
export default {
  data() {
    return {
      seriesData1: series1,
      primaryXAxis: {
        valueType: 'DateTime',
        labelFormat: 'yMMM',
        enableAutoIntervalOnZooming: true
      },
      zoom: {
        enableSelectionZooming: true,
      },
      title: 'Sales History of Product X'
    };
  },
  provide: {
    chart: [AreaSeries, DateTime, Zoom]
  },
};
</script>
<style>
#container {
  height: 350px;
```

```
}
</style>
```

```
{% previewsample "page.domainurl/code-snippet/chart/user-interaction/zoom-cs7" %}
```

## Data editing in Vue Chart component

### Enable Data Editing

We can use the data editing through inject the **DataEditing** module in the chart provider. It provides drag and drop support to the rendered points. Now, we can change the location or value of the point based on its **y** value. To enable the data editing, set the **enable** property to true in the drag settings of the series. Also, we can set color using **fill** property and set the data editing minimum and maximum range using **minY** and **maxY** properties.

### APP.VUE

```
<template>
  <div id="app">
    <ejs-chart id="container" :title='title'
:primaryXAxis='primaryXAxis' :primaryYAxis="primaryYAxis"
:legendSettings='legend' :chartArea="chartArea">
      <e-series-collection>
        <e-series :dataSource='columnData' type='Column' xName='x'
yName='y' name='Product A' :marker="marker" :dragSettings="dragSettings">
        </e-series>
        <e-series :dataSource='lineData' type='Line' xName='x'
yName='y' name='Product B' :marker="marker" :dragSettings="dragSettings">
        </e-series>
      </e-series-collection>
    </ejs-chart>
  </div>
</template>
<script>
import Vue from "vue";
import { ChartPlugin, LineSeries, ColumnSeries, DateTime, Tooltip, Legend,
DataEditing } from "@syncfusion/ej2-vue-charts";
Vue.use(ChartPlugin);
export default {
  data() {
    return {
      columnData: [
        { x: new Date(2005, 0, 1), y: 21 }, { x: new Date(2006, 0,
1), y: 24 },
        { x: new Date(2007, 0, 1), y: 36 }, { x: new Date(2008, 0,
1), y: 38 },
        { x: new Date(2009, 0, 1), y: 54 }, { x: new Date(2010, 0,
1), y: 57 },
        { x: new Date(2011, 0, 1), y: 70 }
      ],
      lineData: [
        { x: new Date(2005, 0, 1), y: 21 }, { x: new Date(2006, 0,
1), y: 24 },
        { x: new Date(2007, 0, 1), y: 36 }, { x: new Date(2008, 0,
1), y: 38 },
        { x: new Date(2009, 0, 1), y: 54 }, { x: new Date(2010, 0,
1), y: 57 },
```

```

        { x: new Date(2011, 0, 1), y: 70 }
    ],
    primaryXAxis: {
        valueType: 'DateTime',
        labelFormat: 'y',
        intervalType: 'Years',
        edgeLabelPlacement: 'Shift',
        majorGridLines: { width: 0 }
    },
    primaryYAxis:
    {
        labelFormat: '{value}%',
        rangePadding: 'None',
        minimum: 0,
        maximum: 100,
        interval: 20,
        lineStyle: { width: 0 },
        majorTickLines: { width: 0 },
        minorTickLines: { width: 0 }
    },
    chartArea: {
        border: {
            width: 0
        }
    },
    title: 'Sales History of Product X',
    legend: { visible: false },
    marker: {
        visible: true, width: 10, height: 10
    },
    dragSettings: {
        enable: true
    }
};
},
provide: {
    chart: [LineSeries, ColumnSeries, DateTime, Tooltip, Legend,
DataEditing]
},
};
</script>
<style>
    #container {
        height: 350px;
    }
</style>

```

{% previewsample "page.domainurl/code-snippet/chart/user-interaction/data-editing-cs1" %}

### Cross hair and track ball in Vue Chart component

Crosshair has a vertical and horizontal line to view the value of the axis at mouse or touch position.

Crosshair lines can be enabled by using [enable](#) property in the `crosshair`.

### APP.VUE

```

<template>
  <div id="app">
    <ejs-chart id= "container" :title='title'
:primaryXAxis='primaryXAxis' :crosshair='crosshair'>
      <e-series-collection>
        <e-series :dataSource='seriesData1' type='Line' xName='x'
yName='y' name='Temperature'> </e-series>
      </e-series-collection>
    </ejs-chart>
  </div>
</template>
<script>
import Vue from "vue";
import { ChartPlugin, LineSeries, Legend, Crosshair, DateTime } from
"@syncfusion/ej2-vue-charts";
Vue.use(ChartPlugin);
let series1: Object[] = [];
let point1: Object;
let value: number = 40;
let i: number;
for (i = 1; i < 500; i++) {
  if (Math.random() > .5) {
    value += Math.random();
  } else {
    value -= Math.random();
  }
  point1 = { x: new Date(1950, i + 2, i), y: value.toFixed(1) };
  series1.push(point1);
}
export default {
  data() {
    return {
      seriesData1: series1,
      primaryXAxis: {
        valueType: 'DateTime',
        labelFormat: 'yMMM'
      },
      title: "Weather Condition",
      crosshair: { enable: true }
    };
  },
  provide: {
    chart: [LineSeries, Crosshair, DateTime]
  },
};
</script>
<style>
#container {
  height: 350px;
}
</style>

```

{% previewsample "page.domainurl/code-snippet/chart/user-interaction/crosshair-cs1" %}

### Tooltip for axis

Tooltip label for an axis can be enabled by using [enable](#) property of `crosshairTooltip` in the corresponding axis.

### APP.VUE

```
<template>
  <div id="app">
    <ejs-chart id= "container" :title='title'
:primaryXAxis='primaryXAxis' :primaryYAxis='primaryYAxis'
:crosshair='crosshair'>
      <e-series-collection>
        <e-series :dataSource='seriesData1' type='Line' xName='x'
yName='y' name='Temperature'> </e-series>
      </e-series-collection>
    </ejs-chart>
  </div>
</template>
<script>
import Vue from "vue";
import { ChartPlugin, LineSeries, Legend, Crosshair, DateTime } from
"@syncfusion/ej2-vue-charts";
Vue.use(ChartPlugin);
let series1: Object[] = [];
let point1: Object;
let value: number = 40;
let i: number;
for (i = 1; i < 500; i++) {
  if (Math.random() > .5) {
    value += Math.random();
  } else {
    value -= Math.random();
  }
  point1 = { x: new Date(1950, i + 2, i), y: value.toFixed(1) };
  series1.push(point1);
}
export default {
  data() {
    return {
      seriesData1: series1,
      primaryXAxis: {
        valueType: 'DateTime',
        crosshairTooltip: { enable: true },
        labelFormat: 'yMMM'
      },
      primaryYAxis: {
        crosshairTooltip: { enable: true }
      },
      title: "Weather Condition",
      crosshair: { enable: true }
    };
  },
  provide: {
    chart: [LineSeries, Crosshair, DateTime]
  }
};
</script>
```

```
<style>
  #container {
    height: 350px;
  }
</style>
```

{% previewsample "page.domainurl/code-snippet/chart/user-interaction/crosshair-cs2" %}

### Customization

The [fill](#) and [textStyle](#)

property of the `crosshairTooltip` is used to customize the background color and font style of the crosshair label respectively. Color and width of the crosshair line can be customized by using the [line](#) property in the crosshair.

### APP.VUE

```
<template>
  <div id="app">
    <ejs-chart id= "container" :title='title'
:primaryXAxis='primaryXAxis' :primaryYAxis='primaryYAxis'
:crosshair='crosshair'>
      <e-series-collection>
        <e-series :dataSource='seriesData1' type='Line' xName='x'
yName='y' name='Temperature'> </e-series>
      </e-series-collection>
    </ejs-chart>
  </div>
</template>
<script>
import Vue from "vue";
import { ChartPlugin, LineSeries, Legend, Crosshair, DateTime } from
"@syncfusion/ej2-vue-charts";
Vue.use(ChartPlugin);
let series1: Object[] = [];
let point1: Object;
let value: number = 40;
let i: number;
for (i = 1; i < 500; i++) {
  if (Math.random() > .5) {
    value += Math.random();
  } else {
    value -= Math.random();
  }
  point1 = { x: new Date(1950, i + 2, i), y: value.toFixed(1) };
  series1.push(point1);
}
export default {
  data() {
    return {
      seriesData1: series1,
      primaryXAxis: {
        valueType: 'DateTime',
        crosshairTooltip: { enable: true, fill: 'green' },
        labelFormat: 'yMMM'
      },
    },
  },
}
```



```

        primaryYAxis: {
            crosshairTooltip: { enable: true, fill: 'green' },
        },
        title: "Weather Condition",
        crosshair: { enable: true, line: {width: 2, color: 'green'}, fill:
'green' }
    };
},
provide: {
    chart: [LineSeries, Crosshair, DateTime]
}
};
</script>
<style>
    #container {
        height: 350px;
    }
</style>

```

{% previewsample "page.domainurl/code-snippet/chart/user-interaction/crosshair-cs3" %}

Note: To use crosshair feature, we need to inject **Crosshair** into the **provide**.

### Trackball

Trackball is used to track a data point closest to the mouse or touch position. Trackball marker indicates the

closest point and trackball tooltip displays the information about the point. To use trackball feature, we need to inject **Crosshair** and **Tooltip** into the **provide**.

Trackball can be enabled by setting the [enable](#) property of the crosshair to true and [shared](#) property in **tooltip** to true in chart.

### APP.VUE

```

<template>
    <div id="app">
        <ejs-chart id= "container" :title='title'
:primaryXAxis='primaryXAxis' :crosshair='crosshair' :tooltip='tooltip'>
            <e-series-collection>
                <e-series :dataSource='seriesData' type='Line' xName='x'
yName='y' name='John' :marker='marker'> </e-series>
                <e-series :dataSource='seriesData' type='Line' xName='x'
yName='y1' name='Andrew' :marker='marker'> </e-series>
                <e-series :dataSource='seriesData' type='Line' xName='x'
yName='y2' name='Thomas' :marker='marker'> </e-series>
                <e-series :dataSource='seriesData' type='Line' xName='x'
yName='y3' name='Mark' :marker='marker'> </e-series>
                <e-series :dataSource='seriesData' type='Line' xName='x'
yName='y4' name='William' :marker='marker'> </e-series>
            </e-series-collection>
        </ejs-chart>
    </div>
</template>
<script>
import Vue from "vue";

```

```

import { ChartPlugin, LineSeries, Legend, Crosshair, DateTime, Tooltip }
from "@syncfusion/ej2-vue-charts";
Vue.use(ChartPlugin);
export default {
  data() {
    return {
      seriesData: [
        { x: new Date(2000, 2, 11), y: 15, y1: 39, y2: 60, y3: 75, y4: 85 },
        { x: new Date(2000, 9, 14), y: 20, y1: 30, y2: 55, y3: 75, y4: 83 },
        { x: new Date(2001, 2, 11), y: 25, y1: 28, y2: 48, y3: 68, y4: 85 },
        { x: new Date(2001, 9, 16), y: 21, y1: 35, y2: 57, y3: 75, y4: 87 },
        { x: new Date(2002, 2, 7), y: 13, y1: 39, y2: 62, y3: 71, y4: 82 },
        { x: new Date(2002, 9, 7), y: 18, y1: 41, y2: 64, y3: 69, y4: 74 },
        { x: new Date(2003, 2, 11), y: 24, y1: 45, y2: 57, y3: 81, y4: 73 },
        { x: new Date(2003, 9, 14), y: 23, y1: 48, y2: 53, y3: 84, y4: 75 },
        { x: new Date(2004, 2, 6), y: 19, y1: 54, y2: 63, y3: 85, y4: 73 },
        { x: new Date(2004, 9, 6), y: 31, y1: 55, y2: 50, y3: 87, y4: 60 },
        { x: new Date(2005, 2, 11), y: 39, y1: 57, y2: 66, y3: 75, y4: 48 },
        { x: new Date(2005, 9, 11), y: 50, y1: 60, y2: 65, y3: 70, y4: 55 },
        { x: new Date(2006, 2, 11), y: 24, y1: 60, y2: 79, y3: 85, y4: 40 }
      ],
      primaryXAxis: {
        title: 'Years',
        minimum: new Date(2000, 1, 1), maximum: new Date(2006, 2, 11),
        intervalType: 'Years',
        valueType: 'DateTime',
      },
      title: "Average Sales per Person",
      crosshair: { enable: true, lineType: 'Vertical' },
      tooltip: { enable: true, shared: true, format: '${series.name} : ${point.x} : ${point.y}' },
      marker: { visible: true }
    };
  },
  provide: {
    chart: [LineSeries, Crosshair, DateTime, Tooltip]
  },
};
</script>
<style>
#container {
  height: 350px;
}
</style>

```

```
{% previewsample "page.domainurl/code-snippet/chart/user-interaction/trackball-cs1" %}
```

## Synchronized Charts in Vue Chart component

### Tooltip synchronization

The tooltip can be synchronized across multiple charts using the [showTooltip](#) and [hideTooltip](#) methods. When we hover over a data point in one chart, we call the `showTooltip` method for the other charts to display related information in other connected charts simultaneously.

In the `showTooltip` method, specify the following parameters programmatically to enable tooltip for a particular chart:

- `x` - Data point x-value or x-coordinate value.
- `y` - Data point y-value or y-coordinate value.

### APP.VUE

```
<template>
  <div class="control-section">
    <div class="row">
      <div class="col" id="container1">
        <ejs-chart style='display:block' align='center'
id='chartcontainer1' :title='title1'
:primaryXAxis='primaryXAxis'
:primaryYAxis='primaryYAxis1' :tooltip='tooltip1' :titleStyle='titleStyle'
:chartMouseLeave='chartMouseLeave1'
:chartMouseMove='chartMouseMove1' :chartMouseUp='chartMouseUp1'
ref="chart1">
          <e-series-collection>
            <e-series :dataSource='seriesData' type='Line'
xName='USD' yName='EUR' width = 2
:emptyPointSetting='emptyPointSettings'> </e-
series>
          </e-series-collection>
        </ejs-chart>
      </div>
      <div class="col" id="container2">
        <ejs-chart style='display:block' align='center'
id='chartcontainer2' :title='title2'
:primaryXAxis='primaryXAxis'
:primaryYAxis='primaryYAxis2' :tooltip='tooltip2' :titleStyle='titleStyle'
:chartMouseLeave='chartMouseLeave2'
:chartMouseMove='chartMouseMove2' :chartMouseUp='chartMouseUp2'
ref="chart2">
          <e-series-collection>
            <e-series :dataSource='seriesData' type='Area'
xName='USD' yName='INR' opacity=0.6
:border='border'>
            </e-series>
          </e-series-collection>
        </ejs-chart>
      </div>
    </div>
  </div>
```

```

</template>
<script>
import Vue from "vue";
import { ChartPlugin, AreaSeries, LineSeries, DateTime, Tooltip } from
"@syncfusion/ej2-vue-charts";
import { synchronizedData } from './dataSource.js';
import { Browser } from '@syncfusion/ej2-base';
Vue.use(ChartPlugin);
export default {
  data() {
    return {
      seriesData: synchronizedData,
      primaryXAxis: {
        minimum: new Date(2023, 1, 18),
        maximum: new Date(2023, 7, 18),
        valueType: 'DateTime',
        labelFormat: 'MMM d',
        lineStyle: { width: 0 },
        majorGridLines: { width: 0 },
        interval: Browser.isDevice ? 2 : 1,
        edgeLabelPlacement: Browser.isDevice ? 'None' : 'Shift',
        labelRotation: Browser.isDevice ? -45 : 0
      },
      primaryYAxis1: {
        labelFormat: 'n2',
        majorTickLines: { width: 0 },
        lineStyle: { width: 0 },
        minimum: 0.86,
        maximum: 0.96,
        interval: 0.025
      },
      primaryYAxis2: {
        labelFormat: 'n1',
        majorTickLines: { width: 0 },
        lineStyle: { width: 0 },
        minimum: 79,
        maximum: 85,
        interval: 1.5
      },
      border: { width: 2 },
      tooltip1: {
        enable: true,
        shared: true,
        header: '',
        enableMarker: false,
        format: '<b>€${point.y}</b> <br>${point.x} 2023',
        fadeOutDuration: Browser.isDevice ? 2500 : 1000
      },
      tooltip2: {
        enable: true,
        shared: true,
        header: '',
        enableMarker: false,
        format: '<b>₹${point.y}</b> <br>${point.x} 2023',
        fadeOutDuration: Browser.isDevice ? 2500 : 1000
      },
      emptyPointSettings: { mode: 'Drop' },
    }
  }
}

```

```

        titleStyle: { textAlignment: 'Near' },
        title1: "US to EURO",
        title2: "US to INR"
    };
},
provide: {
    chart: [AreaSeries, LineSeries, DateTime, Tooltip]
},
methods: {
    chartMouseLeave1: function () {
        this.$refs.chart2.ej2Instances.hideTooltip();
    },
    chartMouseUp1: function () {
        if (Browser.isDevice || this.$refs.chart1.ej2Instances.startMove) {
            this.$refs.chart2.ej2Instances.hideTooltip();
        }
    },
    chartMouseMove1: function (args) {
        if ((!Browser.isDevice && !this.$refs.chart1.ej2Instances.isTouch &&
        !this.$refs.chart1.ej2Instances.isChartDrag) ||
        this.$refs.chart1.ej2Instances.startMove) {
            this.$refs.chart2.ej2Instances.startMove =
            this.$refs.chart1.ej2Instances.startMove;
            this.$refs.chart2.ej2Instances.showTooltip(args.x, args.y);
        }
    },
    chartMouseMove2: function (args) {
        if ((!Browser.isDevice && !this.$refs.chart2.ej2Instances.isTouch &&
        !this.$refs.chart2.ej2Instances.isChartDrag) ||
        this.$refs.chart2.ej2Instances.startMove) {
            this.$refs.chart1.ej2Instances.startMove =
            this.$refs.chart2.ej2Instances.startMove;
            this.$refs.chart1.ej2Instances.showTooltip(args.x, args.y);
        }
    },
    chartMouseLeave2: function () {
        this.$refs.chart1.ej2Instances.hideTooltip();
    },
    chartMouseUp2: function () {
        if (Browser.isDevice || this.$refs.chart2.ej2Instances.startMove) {
            this.$refs.chart1.ej2Instances.hideTooltip();
        }
    }
}
};
</script>
<style>
    #container {
        height: 350px;
    }
    #control-container {
        padding: 1px !important;
    }
    .row {
        display: flex;
    }
    .col {

```

```

width: 50%;
margin: 10px;
height: 270px;
}
</style>

```

{% previewsample "page.domainurl/code-snippet/chart/user-interaction/synchronized-cs1" %}

### Crosshair synchronization

The crosshair can be synchronized across multiple charts using the [showCrosshair](#) and [hideCrosshair](#) methods. When we hover over one chart, we call the [showCrosshair](#) method for the other charts to align with data points in other connected charts, simplifying data comparison and analysis.

In the [showCrosshair](#) method, specify the following parameters programmatically to enable crosshair for a particular chart:

- **x** - Specifies the x-value of the point or x-coordinate.
- **y** - Specifies the y-value of the point or y-coordinate.

### APP.VUE

```

<template>
  <div class="control-section">
    <div class="row">
      <div class="col" id="container1">
        <ejs-chart style='display:block' align='center'
id='chartcontainer1' :title='title1'
:primaryXAxis='primaryXAxis'
:primaryYAxis='primaryYAxis1' :crosshair='crosshair'
:titleStyle='titleStyle'
:chartMouseLeave='chartMouseLeave1'
:chartMouseMove='chartMouseMove1' :chartMouseUp='chartMouseUp1'
ref="chart1">
          <e-series-collection>
            <e-series :dataSource='seriesData' type='Spline'
xName='USD' yName='EUR' width = 2
:emptyPointSetting='emptyPointSettings'> </e-
series>
          </e-series-collection>
        </ejs-chart>
      </div>
      <div class="col" id="container2">
        <ejs-chart style='display:block' align='center'
id='chartcontainer2' :title='title2'
:primaryXAxis='primaryXAxis'
:primaryYAxis='primaryYAxis2' :crosshair='crosshair'
:titleStyle='titleStyle'
:chartMouseLeave='chartMouseLeave2'
:chartMouseMove='chartMouseMove2' :chartMouseUp='chartMouseUp2'
ref="chart2">
          <e-series-collection>
            <e-series :dataSource='seriesData' type='Area'
xName='USD' yName='INR' opacity=0.6
:border='border'>

```

```

        </e-series>
      </e-series-collection>
    </ejs-chart>
  </div>
</div>
</div>
</template>
<script>
import Vue from "vue";
import { ChartPlugin, AreaSeries, SplineSeries, DateTime, Crosshair } from
"@syncfusion/ej2-vue-charts";
import { synchronizedData } from '../dataSource.js';
import { Browser } from "@syncfusion/ej2-base";
Vue.use(ChartPlugin);
export default {
  data() {
    return {
      seriesData: synchronizedData,
      primaryXAxis: {
        minimum: new Date(2023, 1, 18),
        maximum: new Date(2023, 7, 18),
        valueType: 'DateTime',
        labelFormat: 'MMM d',
        lineStyle: { width: 0 },
        majorGridLines: { width: 0 },
        interval: Browser.isDevice ? 2 : 1,
        edgeLabelPlacement: Browser.isDevice ? 'None' : 'Shift',
        labelRotation: Browser.isDevice ? -45 : 0,
        crosshairTooltip: { enable: true }
      },
      primaryYAxis1: {
        labelFormat: 'n2',
        majorTickLines: { width: 0 },
        lineStyle: { width: 0 },
        minimum: 0.86,
        maximum: 0.96,
        interval: 0.025
      },
      primaryYAxis2: {
        labelFormat: 'n1',
        majorTickLines: { width: 0 },
        lineStyle: { width: 0 },
        minimum: 79,
        maximum: 85,
        interval: 1.5
      },
      border: { width: 2 },
      crosshair: { enable: true, lineType: 'Vertical', dashArray: '2,2' },
      emptyPointSettings: { mode: 'Drop' },
      titleStyle: { textAlignment: 'Near' },
      title1: "US to EURO",
      title2: "US to INR"
    };
  },
  provide: {
    chart: [AreaSeries, SplineSeries, DateTime, Crosshair]
  },

```

```

methods: {
  chartMouseLeave1: function () {
    this.$refs.chart2.ej2Instances.hideCrosshair();
  },
  chartMouseUp1: function () {
    if (Browser.isDevice || this.$refs.chart1.ej2Instances.startMove) {
      this.$refs.chart2.ej2Instances.hideCrosshair();
    }
  },
  chartMouseMove1: function (args) {
    if ((!Browser.isDevice && !this.$refs.chart1.ej2Instances.isTouch &&
    !this.$refs.chart1.ej2Instances.isChartDrag) ||
    this.$refs.chart1.ej2Instances.startMove) {
      this.$refs.chart2.ej2Instances.startMove =
      this.$refs.chart1.ej2Instances.startMove;
      this.$refs.chart2.ej2Instances.showCrosshair(args.x, args.y);
    }
  },
  chartMouseMove2: function (args) {
    if ((!Browser.isDevice && !this.$refs.chart2.ej2Instances.isTouch &&
    !this.$refs.chart2.ej2Instances.isChartDrag) ||
    this.$refs.chart2.ej2Instances.startMove) {
      this.$refs.chart1.ej2Instances.startMove =
      this.$refs.chart2.ej2Instances.startMove;
      this.$refs.chart1.ej2Instances.showCrosshair(args.x, args.y);
    }
  },
  chartMouseLeave2: function () {
    this.$refs.chart1.ej2Instances.hideCrosshair();
  },
  chartMouseUp2: function () {
    if (Browser.isDevice || this.$refs.chart2.ej2Instances.startMove) {
      this.$refs.chart1.ej2Instances.hideCrosshair();
    }
  }
}
};
</script>
<style>
  #container {
    height: 350px;
  }
  #control-container {
    padding: 1px !important;
  }
  .row {
    display: flex;
  }
  .col {
    width: 50%;
    margin: 10px;
    height: 270px;
  }
</style>

```

{% previewsample "page.domainurl/code-snippet/chart/user-interaction/synchronized-cs2" %}



### Zooming synchronization

You can maintain constant zoom levels across multiple charts using the [zoomComplete](#) event. In the [zoomComplete](#) event, obtain the [zoomFactor](#) and [zoomPosition](#) values of the particular chart, and then apply those values to the other charts.

### APP.VUE

```
<template>
  <div class="control-section">
    <div class="row">
      <div class="col" id="container1">
        <ejs-chart style='display:block' align='center'
id='chartcontainer1' :title='title1'
:primaryXAxis='primaryXAxis'
:primaryYAxis='primaryYAxis1'
:zoomSettings='zoomSettings' :titleStyle='titleStyle'
:zoomComplete='zoomComplete' ref="chart1">
          <e-series-collection>
            <e-series :dataSource='seriesData' type='Line'
xName='USD' yName='EUR' width = 2
:emptyPointSetting='emptyPointSettings'> </e-
series>
          </e-series-collection>
        </ejs-chart>
      </div>
      <div class="col" id="container2">
        <ejs-chart style='display:block' align='center'
id='chartcontainer2' :title='title2'
:primaryXAxis='primaryXAxis'
:primaryYAxis='primaryYAxis2'
:zoomSettings='zoomSettings' :titleStyle='titleStyle'
:zoomComplete='zoomComplete' ref="chart2">
          <e-series-collection>
            <e-series :dataSource='seriesData' type='SplineArea'
xName='USD' yName='INR' opacity=0.6
:border='border'>
          </e-series>
        </e-series-collection>
      </ejs-chart>
    </div>
  </div>
</template>
<script>
import Vue from "vue";
import { ChartPlugin, LineSeries, SplineAreaSeries, DateTime, Zoom } from
"@syncfusion/ej2-vue-charts";
import { synchronizedData } from './dataSource.js';
import { Browser } from '@syncfusion/ej2-base';
Vue.use(ChartPlugin);
let zoomFactor = 0;
let zoomPosition = 0;
export default {
  data() {
    return {
      seriesData: synchronizedData,
```

```

primaryXAxis: {
  minimum: new Date(2023, 1, 18),
  maximum: new Date(2023, 7, 18),
  valueType: 'DateTime',
  labelFormat: 'MMM d',
  lineStyle: { width: 0 },
  majorGridLines: { width: 0 },
  interval: Browser.isDevice ? 2 : 1,
  edgeLabelPlacement: Browser.isDevice ? 'None' : 'Shift',
  labelRotation: Browser.isDevice ? -45 : 0
},
primaryYAxis1: {
  labelFormat: 'n2',
  majorTickLines: { width: 0 },
  lineStyle: { width: 0 },
  minimum: 0.86,
  maximum: 0.96,
  interval: 0.025
},
primaryYAxis2: {
  labelFormat: 'n1',
  majorTickLines: { width: 0 },
  lineStyle: { width: 0 },
  minimum: 79,
  maximum: 85,
  interval: 1.5
},
border: { width: 2 },
zoomSettings: {
  enableMouseWheelZooming: true,
  enablePinchZooming: true,
  enableScrollbar: false,
  enableDeferredZooming: false,
  enableSelectionZooming: true,
  enablePan: true,
  mode: 'X',
  toolbarItems: ['Pan', 'Reset']
},
emptyPointSettings: { mode: 'Drop' },
titleStyle: { textAlignment: 'Near' },
title1: "US to EURO",
title2: "US to INR",
charts: []
};
},
provide: {
  chart: [LineSeries, SplineAreaSeries, DateTime, Zoom]
},
methods: {
  zoomComplete: function (args) {
    if (args.axis.name === 'primaryXAxis') {
      zoomFactor = args.currentZoomFactor;
      zoomPosition = args.currentZoomPosition;
      this.zoomCompleteFunction(args);
    }
  },
  zoomCompleteFunction: function (args) {

```

```

        for (var i = 0; i < this.charts.length; i++) {
            if (args.axis.series[0].chart.element.id !==
this.charts[i].element.id) {
                this.charts[i].primaryXAxis.zoomFactor = zoomFactor;
                this.charts[i].primaryXAxis.zoomPosition = zoomPosition;
                this.charts[i].zoomModule.isZoomed =
args.axis.series[0].chart.zoomModule.isZoomed;
                this.charts[i].zoomModule.isPanning =
args.axis.series[0].chart.zoomModule.isPanning;
            }
        },
        mounted() {
            this.charts = [this.$refs.chart1.ej2Instances,
this.$refs.chart2.ej2Instances];
        }
    };
</script>
<style>
    #container {
        height: 350px;
    }
    #control-container {
        padding: 1px !important;
    }
    .row {
        display: flex;
    }
    .col {
        width: 50%;
        margin: 10px;
        height: 270px;
    }
</style>

```

{% previewsample "page.domainurl/code-snippet/chart/user-interaction/synchronized-cs3" %}

### Selection synchronization

You can select the data across multiple charts using the [selectionComplete](#) event. In the `selectionComplete` event, obtain the selected values of the particular chart, and then apply those values to the other charts.

### APP.VUE

```

<template>
    <div class="control-section">
        <div class="row">
            <div class="col" id="container1">
                <ejs-chart style='display:block' align='center'
id='chartcontainer1' :title='title1'
:primaryXAxis='primaryXAxis'
:primaryYAxis='primaryYAxis1'
:zoomSettings='zoomSettings' :titleStyle='titleStyle'
:zoomComplete='zoomComplete' :selectionComplete='selectionComplete'
ref="chart1">

```

```

        <e-series-collection>
            <e-series :dataSource='seriesData' type='Line'
xName='USD' yName='EUR' width=2
                :emptyPointSetting='emptyPointSettings'> </e-
series>
        </e-series-collection>
    </ejs-chart>
</div>
<div class="col" id="container2">
    <ejs-chart style='display:block' align='center'
id='chartcontainer2' :title='title2'
        :primaryXAxis='primaryXAxis'
:primaryYAxis='primaryYAxis2'
        :zoomSettings='zoomSettings' :titleStyle='titleStyle'
:zoomComplete='zoomComplete' :selectionComplete='selectionComplete'
ref="chart2">
        <e-series-collection>
            <e-series :dataSource='seriesData' type='Spline'
xName='USD' yName='INR' width=2
                :border='border'>
            </e-series>
        </e-series-collection>
    </ejs-chart>
</div>
</div>
</template>
<script>
import Vue from "vue";
import { ChartPlugin, SplineSeries, LineSeries, DateTime, Zoom, Selection }
from "@syncfusion/ej2-vue-charts";
import { synchronizedData } from './dataSource.js';
import { Browser } from '@syncfusion/ej2-base';
Vue.use(ChartPlugin);
let zoomFactor = 0;
let zoomPosition = 0;
let count = 0;
export default {
    data() {
        return {
            seriesData: synchronizedData,
            primaryXAxis: {
                minimum: new Date(2023, 1, 18),
                maximum: new Date(2023, 7, 18),
                valueType: 'DateTime',
                labelFormat: 'MMM d',
                lineStyle: { width: 0 },
                majorGridLines: { width: 0 },
                interval: Browser.isDevice ? 2 : 1,
                edgeLabelPlacement: Browser.isDevice ? 'None' : 'Shift',
                labelRotation: Browser.isDevice ? -45 : 0
            },
            primaryYAxis1: {
                labelFormat: 'n2',
                majorTickLines: { width: 0 },
                lineStyle: { width: 0 },
                minimum: 0.86,

```

```

        maximum: 0.96,
        interval: 0.025
    },
    primaryYAxis2: {
        labelFormat: 'n1',
        majorTickLines: { width: 0 },
        lineStyle: { width: 0 },
        minimum: 79,
        maximum: 85,
        interval: 1.5
    },
    border: { width: 2 },
    zoomSettings: {
        enableSelectionZooming: true,
        mode: 'X'
    },
    emptyPointSettings: { mode: 'Drop' },
    selectionPattern: 'Box',
    titleStyle: { textAlignment: 'Near' },
    selectionMode: 'Point',
    title1: "US to EURO",
    title2: "US to INR",
    charts: []
};
},
provide: {
    chart: [SplineSeries, LineSeries, DateTime, Zoom, Selection]
},
methods: {
    zoomComplete: function (args) {
        if (args.axis.name === 'primaryXAxis') {
            zoomFactor = args.currentZoomFactor;
            zoomPosition = args.currentZoomPosition;
            this.zoomCompleteFunction(args);
        }
    },
    zoomCompleteFunction: function (args) {
        for (var i = 0; i < this.charts.length; i++) {
            if (args.axis.series[0].chart.element.id !==
this.charts[i].element.id) {
                this.charts[i].primaryXAxis.zoomFactor = zoomFactor;
                this.charts[i].primaryXAxis.zoomPosition = zoomPosition;
                this.charts[i].zoomModule.isZoomed =
args.axis.series[0].chart.zoomModule.isZoomed;
                this.charts[i].zoomModule.isPanning =
args.axis.series[0].chart.zoomModule.isPanning;
            }
        }
    },
    selectionComplete: function (args) {
        if (count == 0) {
            for (var j = 0; j < args.selectedDataValues.length; j++) {
                args.selectedDataValues[j].point =
args.selectedDataValues[j].pointIndex;
                args.selectedDataValues[j].series =
args.selectedDataValues[j].seriesIndex;
            }
        }
    }
}

```

```

        for (var i = 0; i < this.charts.length; i++) {
            if (args.chart.element.id !== this.charts[i].element.id) {
                this.charts[i].selectedDataIndexes =
args.selectedDataValues;
                count += 1;
                this.charts[i].dataBind();
            }
        }
        count = 0;
    },
    },
    mounted() {
        this.charts = [this.$refs.chart1.ej2Instances,
this.$refs.chart2.ej2Instances];
    }
};
</script>
<style>
    #container {
        height: 350px;
    }
    #control-container {
        padding: 1px !important;
    }
    .row {
        display: flex;
    }
    .col {
        width: 50%;
        margin: 10px;
        height: 270px;
    }
</style>

```

{% previewsample "page.domainurl/code-snippet/chart/user-interaction/synchronized-cs4" %}

<!-- markdownlint-disable MD036 -->

### Selection in Vue Chart component

Chart provides selection support for the series and its data points on mouse click.

When Mouse is clicked on the data points, the corresponding series legend will also be selected.

We have different type of selection mode for selecting the data. They are,

- None
- Point
- Series
- Cluster
- DragXY
- DragX
- DragY

## Point

You can select a point, by setting `selectionMode` to point.

## APP.VUE

```

<template>
  <div id="app">
    <ejs-chart id="container" :title='title'
:primaryXAxis='primaryXAxis' selectionMode='Point'>
      <e-series-collection>
        <e-series :dataSource='seriesData' type='Column'
xName='country' yName='gold' name='Gold'> </e-series>
        <e-series :dataSource='seriesData' type='Column'
xName='country' yName='silver' name='Silver'> </e-series>
        <e-series :dataSource='seriesData' type='Column'
xName='country' yName='bronze' name='Bronze'> </e-series>
      </e-series-collection>
    </ejs-chart>
  </div>
</template>
<script>
import Vue from "vue";
import { ChartPlugin, ColumnSeries, Category, Legend, Selection } from
"@syncfusion/ej2-vue-charts";
Vue.use(ChartPlugin);
export default {
  data() {
    return {
      seriesData: [
        { country: "USA", gold: 50, silver: 70, bronze: 45 },
        { country: "China", gold: 40, silver: 60, bronze: 55 },
        { country: "Japan", gold: 70, silver: 60, bronze: 50 },
        { country: "Australia", gold: 60, silver: 56, bronze: 40 },
        { country: "France", gold: 50, silver: 45, bronze: 35 },
        { country: "Germany", gold: 40, silver: 30, bronze: 22 },
        { country: "Italy", gold: 40, silver: 35, bronze: 37 },
        { country: "Sweden", gold: 30, silver: 25, bronze: 27 }
      ],
      primaryXAxis: {
        valueType: 'Category',
        title: 'Countries'
      },
      title: "Olympic Medals"
    };
  },
  provide: {
    chart: [ColumnSeries, Category, Legend, Selection ]
  }
};
</script>
<style>
  #container {
    height: 350px;
  }
</style>

```

{% previewsample "page.domainurl/code-snippet/chart/user-interaction/selection-cs1" %}

Note: To use select feature, we need to Inject `Selection` into the `provide`.

### Series

You can select a series, by setting `selectionMode` to series.

### APP.VUE

```
<template>
  <div id="app">
    <ejs-chart id="container" :title='title'
:primaryXAxis='primaryXAxis' selectionMode='Series'>
      <e-series-collection>
        <e-series :dataSource='seriesData' type='Column'
xName='country' yName='gold' name='Gold'> </e-series>
        <e-series :dataSource='seriesData' type='Column'
xName='country' yName='silver' name='Silver'> </e-series>
        <e-series :dataSource='seriesData' type='Column'
xName='country' yName='bronze' name='Bronze'> </e-series>
      </e-series-collection>
    </ejs-chart>
  </div>
</template>
<script>
import Vue from "vue";
import { ChartPlugin, ColumnSeries, Category, Legend, Selection } from
"@syncfusion/ej2-vue-charts";
Vue.use(ChartPlugin);
export default {
  data() {
    return {
      seriesData: [
        { country: "USA", gold: 50, silver: 70, bronze: 45 },
        { country: "China", gold: 40, silver: 60, bronze: 55 },
        { country: "Japan", gold: 70, silver: 60, bronze: 50 },
        { country: "Australia", gold: 60, silver: 56, bronze: 40 },
        { country: "France", gold: 50, silver: 45, bronze: 35 },
        { country: "Germany", gold: 40, silver: 30, bronze: 22 },
        { country: "Italy", gold: 40, silver: 35, bronze: 37 },
        { country: "Sweden", gold: 30, silver: 25, bronze: 27 }
      ],
      primaryXAxis: {
        valueType: 'Category',
        title: 'Countries'
      },
      title: "Olympic Medals"
    };
  },
  provide: {
    chart: [ColumnSeries, Category, Legend, Selection]
  }
};
</script>
<style>
#container {
  height: 350px;
```



```
}
</style>
```

```
{% previewsample "page.domainurl/code-snippet/chart/user-interaction/selection-cs2" %}
```

### Cluster

You can select the points that corresponds to the same index in all the series, by setting `selectionMode` to

`cluster`.

### APP.VUE

```
<template>
  <div id="app">
    <ejs-chart id="container" :title='title'
:primaryXAxis='primaryXAxis' selectionMode='Cluster'>
      <e-series-collection>
        <e-series :dataSource='seriesData' type='Column'
xName='country' yName='gold' name='Gold'> </e-series>
        <e-series :dataSource='seriesData' type='Column'
xName='country' yName='silver' name='Silver'> </e-series>
        <e-series :dataSource='seriesData' type='Column'
xName='country' yName='bronze' name='Bronze'> </e-series>
      </e-series-collection>
    </ejs-chart>
  </div>
</template>
<script>
import Vue from "vue";
import { ChartPlugin, ColumnSeries, Category, Legend, Selection } from
"@syncfusion/ej2-vue-charts";
Vue.use(ChartPlugin);
export default {
  data() {
    return {
      seriesData: [
        { country: "USA", gold: 50, silver: 70, bronze: 45 },
        { country: "China", gold: 40, silver: 60, bronze: 55 },
        { country: "Japan", gold: 70, silver: 60, bronze: 50 },
        { country: "Australia", gold: 60, silver: 56, bronze: 40 },
        { country: "France", gold: 50, silver: 45, bronze: 35 },
        { country: "Germany", gold: 40, silver: 30, bronze: 22 },
        { country: "Italy", gold: 40, silver: 35, bronze: 37 },
        { country: "Sweden", gold: 30, silver: 25, bronze: 27 }
      ],
      primaryXAxis: {
        valueType: 'Category',
        title: 'Countries'
      },
      title: "Olympic Medals"
    };
  },
  provide: {
    chart: [ColumnSeries, Category, Legend, Selection]
  }
}
```

```
};
</script>
<style>
  #container {
    height: 350px;
  }
</style>
```

{% previewsample "page.domainurl/code-snippet/chart/user-interaction/selection-cs3" %}

### Rectangular selection

#### DragXY, DragX and DragY

To fetch the collection of data under a particular region, you have to set `selectionMode` as `DragXY`.

- DragXY - Allows us to select data with respect to horizontal and vertical axis.
- DragX - Allows us to select data with respect to horizontal axis.
- DragY - Allows us to select data with respect to vertical axis.

The selected data's are returned as an array collection in the [dragComplete](#)

event.

#### APP.VUE

```
<template>
  <div id="app">
    <ejs-chart id="container" :title='title'
:primaryXAxis='primaryXAxis' :primaryYAxis='primaryYAxis'
selectionMode='DragXY'>
      <e-series-collection>
        <e-series :dataSource='seriesData1' type='Scatter' xName='x'
yName='y' name='Male' opacity=0.7> </e-series>
        <e-series :dataSource='seriesData2' type='Scatter' xName='x'
yName='y' name='Female' opacity=0.7> </e-series>
      </e-series-collection>
    </ejs-chart>
  </div>
</template>
<script>
import Vue from "vue";
import { ChartPlugin, ScatterSeries, Legend, Selection } from
"@syncfusion/ej2-vue-charts";
Vue.use(ChartPlugin);
let series1: Object[] = [];
let series2: Object[] = [];
let point1: Object;
let value: number = 80;
let value1: number = 70;
let i: number;
for (i = 1; i < 50; i++) {
  if (Math.random() > 0.5) {
    value += Math.random();
  } else {
    value -= Math.random();
  }
}
```

```

    value = value < 60 ? 60 : value > 90 ? 90 : value;
    point1 = { x: 120 + (i / 2), y: value.toFixed(1) };
    series1.push(point1);
  }
  for (i = 1; i < 50; i++) {
    if (Math.random() > 0.5) {
      value1 += Math.random();
    } else {
      value1 -= Math.random();
    }
    value1 = value1 < 60 ? 60 : value1 > 90 ? 90 : value1;
    point1 = { x: 120 + (i / 2), y: value1.toFixed(1) };
    series2.push(point1);
  }
  export default {
    data() {
      return {
        seriesData1: series1,
        seriesData2: series2,
        primaryXAxis: {
          title: 'Height (cm)',
          minimum: 120, maximum: 180,
          edgeLabelPlacement: 'Shift',
          labelFormat: '{value}cm'
        },
        primaryYAxis: {
          title: 'Weight (kg)',
          minimum: 60, maximum: 90,
          labelFormat: '{value}kg',
          rangePadding: 'None'
        },
        title: 'Height Vs Weight'
      };
    },
    provide: {
      chart: [ScatterSeries, Legend, Selection]
    },
  };
</script>
<style>
  #container {
    height: 350px;
  }
</style>

```

{% previewsample "page.domainurl/code-snippet/chart/user-interaction/drag-cs1" %}

### Lasso selection

To select a region by drawing freehand shapes to fetch a collection of data use `selectionMode` as `Lasso`. You can also select multiple regions on the chart through this mode.

### APP.VUE

```

<template>
  <div id="app">

```

```

    <ejs-chart id="container" :title='title'
:primaryXAxis='primaryXAxis' :primaryYAxis='primaryYAxis'
selectionMode='Lasso'>
      <e-series-collection>
        <e-series :dataSource='seriesData1' type='Scatter' xName='x'
yName='y' name='Male' opacity=0.7> </e-series>
        <e-series :dataSource='seriesData2' type='Scatter' xName='x'
yName='y' name='Female' opacity=0.7> </e-series>
      </e-series-collection>
    </ejs-chart>
  </div>
</template>
<script>
import Vue from "vue";
import { ChartPlugin, ScatterSeries, Legend, Selection } from
"@syncfusion/ej2-vue-charts";
Vue.use(ChartPlugin);
let series1: Object[] = [];
let series2: Object[] = [];
let point1: Object;
let value: number = 80;
let value1: number = 70;
let i: number;
for (i = 1; i < 50; i++) {
  if (Math.random() > 0.5) {
    value += Math.random();
  } else {
    value -= Math.random();
  }
  value = value < 60 ? 60 : value > 90 ? 90 : value;
  point1 = { x: 120 + (i / 2), y: value.toFixed(1) };
  series1.push(point1);
}
for (i = 1; i < 50; i++) {
  if (Math.random() > 0.5) {
    value1 += Math.random();
  } else {
    value1 -= Math.random();
  }
  value1 = value1 < 60 ? 60 : value1 > 90 ? 90 : value1;
  point1 = { x: 120 + (i / 2), y: value1.toFixed(1) };
  series2.push(point1);
}
export default {
  data() {
    return {
      seriesData1: series1,
      seriesData2: series2,
      primaryXAxis: {
        title: 'Height (cm)',
        minimum: 120, maximum: 180,
        edgeLabelPlacement: 'Shift',
        labelFormat: '{value}cm'
      },
      primaryYAxis: {
        title: 'Weight (kg)',
        minimum: 60, maximum: 90,

```

```

        labelFormat: '{value}kg',
        rangePadding: 'None'
    },
    title: 'Height Vs Weight'
};
},
provide: {
    chart: [ScatterSeries, Legend, Selection]
},
};
</script>
<style>
    #container {
        height: 350px;
    }
</style>

```

{% previewsample "page.domainurl/code-snippet/chart/user-interaction/drag-cs2" %}

### Multi-region selection

To select multiple region on the chart, set the `allowMultiSelection` property to true.

### APP.VUE

```

<template>
    <div id="app">
        <ejs-chart id="container" :title='title'
        :primaryXAxis='primaryXAxis' :primaryYAxis='primaryYAxis'
        selectionMode='DragXY' allowMultiSelection='true'>
            <e-series-collection>
                <e-series :dataSource='seriesData1' type='Scatter' xName='x'
                yName='y' name='Male' opacity=0.7> </e-series>
                <e-series :dataSource='seriesData2' type='Scatter' xName='x'
                yName='y' name='Female' opacity=0.7> </e-series>
            </e-series-collection>
        </ejs-chart>
    </div>
</template>
<script>
import Vue from "vue";
import { ChartPlugin, ScatterSeries, Legend, Selection } from
"@syncfusion/ej2-vue-charts";
Vue.use(ChartPlugin);
let series1: Object[] = [];
let series2: Object[] = [];
let point1: Object;
let value: number = 80;
let value1: number = 70;
let i: number;
for (i = 1; i < 50; i++) {
    if (Math.random() > 0.5) {
        value += Math.random();
    } else {
        value -= Math.random();
    }
    value = value < 60 ? 60 : value > 90 ? 90 : value;
}

```

```

    point1 = { x: 120 + (i / 2), y: value.toFixed(1) };
    series1.push(point1);
  }
  for (i = 1; i < 50; i++) {
    if (Math.random() > 0.5) {
      value1 += Math.random();
    } else {
      value1 -= Math.random();
    }
    value1 = value1 < 60 ? 60 : value1 > 90 ? 90 : value1;
    point1 = { x: 120 + (i / 2), y: value1.toFixed(1) };
    series2.push(point1);
  }
  export default {
    data() {
      return {
        seriesData1: series1,
        seriesData2: series2,
        primaryXAxis: {
          title: 'Height (cm)',
          minimum: 120, maximum: 180,
          edgeLabelPlacement: 'Shift',
          labelFormat: '{value}cm'
        },
        primaryYAxis: {
          title: 'Weight (kg)',
          minimum: 60, maximum: 90,
          labelFormat: '{value}kg',
          rangePadding: 'None'
        },
        title: 'Height Vs Weight'
      };
    },
    provide: {
      chart: [ScatterSeries, Legend, Selection]
    },
  };
</script>
<style>
  #container {
    height: 350px;
  }
</style>

```

{% previewsample "page.domainurl/code-snippet/chart/user-interaction/drag-cs3" %}

### Selection type

You can select multiple points or series, by enabling the [isMultiSelect](#) property.

### APP.VUE

```

<template>
  <div id="app">
    <ejs-chart id="container" :title='title'
    :primaryXAxis='primaryXAxis' selectionMode='Point' isMultiSelect='true'>
      <e-series-collection>

```

```

        <e-series :dataSource='seriesData' type='Column'
xName='country' yName='gold' name='Gold'> </e-series>
        <e-series :dataSource='seriesData' type='Column'
xName='country' yName='silver' name='Silver'> </e-series>
        <e-series :dataSource='seriesData' type='Column'
xName='country' yName='bronze' name='Bronze'> </e-series>
    </e-series-collection>
  </ejs-chart>
</div>
</template>
<script>
import Vue from "vue";
import { ChartPlugin, ColumnSeries, Category, Legend, Selection } from
"@syncfusion/ej2-vue-charts";
Vue.use(ChartPlugin);
export default {
  data() {
    return {
      seriesData: [
        { country: "USA", gold: 50, silver: 70, bronze: 45 },
        { country: "China", gold: 40, silver: 60, bronze: 55 },
        { country: "Japan", gold: 70, silver: 60, bronze: 50 },
        { country: "Australia", gold: 60, silver: 56, bronze: 40 },
        { country: "France", gold: 50, silver: 45, bronze: 35 },
        { country: "Germany", gold: 40, silver: 30, bronze: 22 },
        { country: "Italy", gold: 40, silver: 35, bronze: 37 },
        { country: "Sweden", gold: 30, silver: 25, bronze: 27 }
      ],
      primaryXAxis: {
        valueType: 'Category',
        title: 'Countries'
      },
      title: "Olympic Medals"
    };
  },
  provide: {
    chart: [ColumnSeries, Category, Legend, Selection]
  }
};
</script>
<style>
#container {
  height: 350px;
}
</style>

```

{% previewsample "page.domainurl/code-snippet/chart/user-interaction/selection-cs4" %}

### Selection on load

You can able to select a point or series programmatically on a chart using [selectedDataIndexes](#) property.

### APP.VUE

```

<template>
  <div id="app">

```

```

    <ejs-chart id="container" :title='title'
:primaryXAxis='primaryXAxis' selectionMode='Point' isMultiSelect='true'
:selectedDataIndexes='selectedData'>
      <e-series-collection>
        <e-series :dataSource='seriesData' type='Column'
xName='country' yName='gold' name='Gold' :animation='animation'
selectionStyle='chartSelection1'> </e-series>
        <e-series :dataSource='seriesData' type='Column'
xName='country' yName='silver' name='Silver' :animation='animation'
selectionStyle='chartSelection1'> </e-series>
        <e-series :dataSource='seriesData' type='Column'
xName='country' yName='bronze' name='Bronze' :animation='animation'
selectionStyle='chartSelection1'> </e-series>
      </e-series-collection>
    </ejs-chart>
  </div>
</template>
<script>
import Vue from "vue";
import { ChartPlugin, ColumnSeries, Category, Legend, Selection } from
"@syncfusion/ej2-vue-charts";
Vue.use(ChartPlugin);
export default {
  data() {
    return {
      seriesData: [
        { country: "USA", gold: 50, silver: 70, bronze: 45 },
        { country: "China", gold: 40, silver: 60, bronze: 55 },
        { country: "Japan", gold: 70, silver: 60, bronze: 50 },
        { country: "Australia", gold: 60, silver: 56, bronze: 40 },
        { country: "France", gold: 50, silver: 45, bronze: 35 },
        { country: "Germany", gold: 40, silver: 30, bronze: 22 },
        { country: "Italy", gold: 40, silver: 35, bronze: 37 },
        { country: "Sweden", gold: 30, silver: 25, bronze: 27 }
      ],
      primaryXAxis: {
        valueType: 'Category',
        title: 'Countries'
      },
      selectedData:[
        { series: 0, point: 1}, { series: 2, point: 3}
      ],
      animation:{ enable: false},
      title: "Olympic Medals"
    };
  },
  provide: {
    chart: [ColumnSeries, Category, Legend, Selection]
  }
};
</script>
<style>
#container {
  height: 350px;
}
.chartSelection1 {
  fill: red

```



```

}
.chartSelection2 {
  fill: green
}
.chartSelection3 {
  fill: blue
}
</style>

```

{% previewsample "page.domainurl/code-snippet/chart/user-interaction/selection-cs5" %}

### Selection through on legend

You can able to select a point or series through on legend using [toggleVisibility](#) property. Also, use [enableHighlight](#) property for highlighting the series through legend.

### APP.VUE

```

<template>
  <div id="app">
    <ejs-chart id="container" :title='title'
:primaryXAxis='primaryXAxis' :legendSettings='legendSettings'
selectionMode='Point'>
      <e-series-collection>
        <e-series :dataSource='seriesData' type='Column'
xName='country' yName='gold' name='Gold' :animation='animation'> </e-series>
        <e-series :dataSource='seriesData' type='Column'
xName='country' yName='silver' name='Silver' :animation='animation'> </e-
series>
        <e-series :dataSource='seriesData' type='Column'
xName='country' yName='bronze' name='Bronze' :animation='animation'> </e-
series>
      </e-series-collection>
    </ejs-chart>
  </div>
</template>
<script>
import Vue from "vue";
import { ChartPlugin, ColumnSeries, Category, Legend, Selection, Highlight }
from "@syncfusion/ej2-vue-charts";
Vue.use(ChartPlugin);
export default {
  data() {
    return {
      seriesData: [
        { country: "USA", gold: 50, silver: 70, bronze: 45 },
        { country: "China", gold: 40, silver: 60, bronze: 55 },
        { country: "Japan", gold: 70, silver: 60, bronze: 50 },
        { country: "Australia", gold: 60, silver: 56, bronze: 40 },
        { country: "France", gold: 50, silver: 45, bronze: 35 },
        { country: "Germany", gold: 40, silver: 30, bronze: 22 },
        { country: "Italy", gold: 40, silver: 35, bronze: 37 },
        { country: "Sweden", gold: 30, silver: 25, bronze: 27 }
      ],
      primaryXAxis: {
        valueType: 'Category',
        title: 'Countries'
      }
    }
  }
}

```

```

    },
    animation: { enable: false },
    legendSettings: { visible: true, toggleVisibility: false,
enableHighlight: true }
    title: "Olympic Medals"
  };
},
provide: {
  chart: [ColumnSeries, Category, Legend, Selection, Highlight]
}
};
</script>
<style>
  #container {
    height: 350px;
  }
</style>

```

{% previewsample "page.domainurl/code-snippet/chart/user-interaction/selection-cs6" %}

### Customization for selection

You can apply custom style to selected points or series with [selectionStyle](#) property.

### APP.VUE

```

<template>
  <div id="app">
    <ejs-chart id="container" :title='title'
:primaryXAxis='primaryXAxis' selectionMode='Point' isMultiSelect='true'>
      <e-series-collection>
        <e-series :dataSource='seriesData' type='Column'
xName='country' yName='gold' name='Gold' selectionStyle='chartSelection1'>
        </e-series>
        <e-series :dataSource='seriesData' type='Column'
xName='country' yName='silver' name='Silver'
selectionStyle='chartSelection2'> </e-series>
        <e-series :dataSource='seriesData' type='Column'
xName='country' yName='bronze' name='Bronze'
selectionStyle='chartSelection3'> </e-series>
      </e-series-collection>
    </ejs-chart>
  </div>
</template>
<script>
import Vue from "vue";
import { ChartPlugin, ColumnSeries, Category, Legend, Selection } from
"@syncfusion/ej2-vue-charts";
Vue.use(ChartPlugin);
export default {
  data() {
    return {
      seriesData: [
        { country: "USA", gold: 50, silver: 70, bronze: 45 },
        { country: "China", gold: 40, silver: 60, bronze: 55 },
        { country: "Japan", gold: 70, silver: 60, bronze: 50 },
        { country: "Australia", gold: 60, silver: 56, bronze: 40 },

```

```

        { country: "France", gold: 50, silver: 45, bronze: 35 },
        { country: "Germany", gold: 40, silver: 30, bronze: 22 },
        { country: "Italy", gold: 40, silver: 35, bronze: 37 },
        { country: "Sweden", gold: 30, silver: 25, bronze: 27 }
    ],
    primaryXAxis: {
        valueType: 'Category',
        title: 'Countries'
    },
    animation: { enable: false },
    legendSettings: { visible: true, toggleVisibility: false }
    title: "Olympic Medals"
};
},
provide: {
    chart: [ColumnSeries, Category, Legend, Selection]
}
};
</script>
<style>
    #container {
        height: 350px;
    }
    .chartSelection1 {
        fill: red
    }
    .chartSelection2 {
        fill: green
    }
    .chartSelection3 {
        fill: blue
    }
</style>

```

{% previewsample "page.domainurl/code-snippet/chart/user-interaction/selection-cs7" %}

## Chart print in Vue Chart component

### Print

The rendered chart can be printed directly from the browser by calling the public method print. You can pass array of ID of elements or element to this method. By default it take element of the chart.

### APP.VUE

```

<template>
    <div id="app">
        <ejs-button id='print' @click.native='print'>Print</ejs-button>
        <ejs-chart ref="chart" :title='title' :primaryXAxis='primaryXAxis'
:primaryYAxis='primaryYAxis' style="float: left">
            <e-series-collection>
                <e-series :dataSource='seriesData' type='Polar' xName='x'
yName='y' drawType='Area'> </e-series>
            </e-series-collection>
        </ejs-chart>
    </div>
</template>
<script>

```

```

import Vue from "vue";
import { ChartPlugin, PolarSeries, Category, AreaSeries } from
"@syncfusion/ej2-vue-charts";
import { ButtonPlugin } from "@syncfusion/ej2-vue-buttons";
Vue.use(ChartPlugin);
Vue.use(ButtonPlugin);
export default {
  data() {
    return {
      seriesData:[
        { x: 'Jan', y: -1 }, { x: 'Feb', y: -1 }, { x: 'Mar', y: 2 },
        { x: 'Apr', y: 8 }, { x: 'May', y: 13 }, { x: 'Jun', y: 18 },
        { x: 'Jul', y: 21 }, { x: 'Aug', y: 20 }, { x: 'Sep', y: 16 },
        { x: 'Oct', y: 10 }, { x: 'Nov', y: 4 }, { x: 'Dec', y: 0 }
      ],
      primaryXAxis: {
        valueType: 'Category'
      },
      primaryYAxis: {
        minimum: -5, maximum: 35, interval: 10,
        title: 'Temperature in Celsius',
        labelFormat: '{value}C'
      },
      title: "Climate Graph-2012"
    };
  },
  provide: {
    chart: [PolarSeries, Category, AreaSeries]
  },
  methods: {
    print: function() {
      this.$refs.chart.print();
    }
  }
};
</script>

```

{% previewsample "page.domainurl/code-snippet/chart/series/polar-cs1" %}

## Export

The rendered chart can be exported to **JPEG**, **PNG**, **SVG**, **PDF**, **XLSX**, or **CSV** format using the export method in chart. The input parameters for this method are **type** for format and **fileName** for result.

The optional parameters for this method are,

- **orientation** - either portrait or landscape mode during PDF export,
- **controls** - pass collections of controls for multiple export,
- **width** - width of chart export, and
- **height** - height of chart export.

## APP.VUE

```
<template>
```

```

<ejs-chart ref="chart" :title='title' :primaryXAxis='primaryXAxis'
:primaryYAxis='primaryYAxis' :loaded='loaded'>
  <e-series-collection>
    <e-series :dataSource='seriesData' type='Polar' xName='x'
yName='y' drawType='Area'> </e-series>
  </e-series-collection>
</ejs-chart>
</div>
</template>
<script>
import Vue from "vue";
import { ChartPlugin, PolarSeries, Category, AreaSeries, Export } from
"@syncfusion/ej2-vue-charts";
Vue.use(ChartPlugin);
export default {
  data() {
    return {
      seriesData:[
        { x: 'Jan', y: -1 }, { x: 'Feb', y: -1 }, { x: 'Mar', y: 2 },
        { x: 'Apr', y: 8 }, { x: 'May', y: 13 }, { x: 'Jun', y: 18 },
        { x: 'Jul', y: 21 }, { x: 'Aug', y: 20 }, { x: 'Sep', y: 16 },
        { x: 'Oct', y: 10 }, { x: 'Nov', y: 4 }, { x: 'Dec', y: 0 }
      ],
      primaryXAxis: {
        valueType: 'Category'
      },
      primaryYAxis: {
        minimum: -5, maximum: 35, interval: 10,
        title: 'Temperature in Celsius',
        labelFormat: '{value}C'
      },
      title: "Climate Graph-2012"
    };
  },
  provide: {
    chart: [PolarSeries, Category, AreaSeries, Export]
  },
  methods: {
    loaded: function(args) {
      args.chart.exportModule.export('PNG', 'export');
    }
  }
};
</script>

```

{% previewsample "page.domainurl/code-snippet/chart/series/polar-cs2" %}

#### [Adding header and footer in PDF export](#)

In the export method, specify the following parameters to add a header and footer text to the exported PDF document:

- **header** - Specify the text that should appear at the top of the exported PDF document.
- **footer** - Specify the text that should appear at the bottom of the exported PDF document.

**APP.VUE**

```

<template>
  <div id="app">
    <ejs-button cssClass="e-flat" iconCss='e-icons e-export-icon'
    isPrimary=true v-on:click.native='onClick' id="exportBtn">EXPORT</ejs-
    button>
    <ejs-chart ref="chart" id="container" :title='title'
    :primaryXAxis='primaryXAxis' :primaryYAxis='primaryYAxis'>
      <e-series-collection>
        <e-series :dataSource='seriesData' type='Column' xName='x'
        yName='y' width=2> </e-series>
      </e-series-collection>
    </ejs-chart>
  </div>
</template>
<script>
import Vue from "vue";
import { ChartPlugin, ColumnSeries, Category, Export } from
"@syncfusion/ej2-vue-charts";
import { ButtonPlugin } from "@syncfusion/ej2-vue-buttons";
Vue.use(ChartPlugin);
Vue.use(ButtonPlugin);
export default {
  data: function() {
    return {
      seriesData: [
        { x: 'John', y: 10000 },
        { x: 'Jake', y: 12000 },
        { x: 'Peter', y: 18000 },
        { x: 'James', y: 11000 },
        { x: 'Mary', y: 9700 }
      ],
      primaryXAxis: {
        title: 'Manager',
        valueType: 'Category',
        majorGridLines: { width: 0 }
      },
      primaryYAxis: {
        title: 'Sales',
        minimum: 0,
        maximum: 20000,
        majorGridLines: { width: 0 }
      },
      title: "Sales Comparision"
    };
  },
  provide: {
    chart: [ColumnSeries, Category, Export]
  },
  methods: {
    onClick: function(args) {
      const header = {
        content: 'Chart Header',
        fontSize: 15
      };
      const footer = {

```

```

        content: 'Chart Footer',
        fontSize: 15
    };
    let chart1 = document.getElementById("container").ej2_instances[0];
    chart1.exportModule.export('PDF', 'Chart', 1, [chart1], null, null,
    true, header, footer);
    },
    }
};
</script>
<style>
    #container {
        height: 350px;
    }
    .e-export-icon::before {
        content: '\e728';
    }
</style>

```

{% previewsample "page.domainurl/code-snippet/chart/series/polar-cs14" %}

*Exporting charts into separate page during the PDF export*

During PDF export, set the `exportToMultiplePage` parameter to **true** to export each chart as a separate page.

#### APP.VUE

```

<template>
    <div id="app">
        <ejs-button cssClass="e-flat" iconCss='e-icons e-export-icon'
        isPrimary=true v-on:click.native='onClick' id="exportBtn">EXPORT</ejs-
        button>
        <ejs-chart ref="chart" id="container" :title='title1'
        :primaryXAxis='primaryXAxis1' :primaryYAxis='primaryYAxis1'>
            <e-series-collection>
                <e-series :dataSource='seriesData1' type='Line' xName='x'
                yName='y' width=2 name='Germany' :marker='marker'> </e-series>
                <e-series :dataSource='seriesData2' type='Line' xName='x'
                yName='y' width=2 name='England' :marker='marker'> </e-series>
            </e-series-collection>
        </ejs-chart>
        <ejs-chart ref="chart1" id="container1" :title='title'
        :primaryXAxis='primaryXAxis' :primaryYAxis='primaryYAxis'>
            <e-series-collection>
                <e-series :dataSource='seriesData' type='Column' xName='x'
                yName='y' width=2> </e-series>
            </e-series-collection>
        </ejs-chart>
        <ejs-accumulationchart id="container2" :title='title2'
        enableSmartLabels='enableSmartLabels' :legendSettings='legendSettings'>
            <e-accumulation-series-collection>
                <e-accumulation-series :dataSource='seriesData3' xName='x'
                yName='y' :dataLabel='datalabel' radius='70%' :startAngle='startAngle'
                :endAngle='endAngle'> </e-accumulation-series>
            </e-accumulation-series-collection>
        </ejs-accumulationchart>
    </div>

```

```

    </div>
  </template>
  <script>
import Vue from "vue";
import { ChartPlugin, ColumnSeries, LineSeries, Category, DateTime, Export }
from "@syncfusion/ej2-vue-charts";
import { AccumulationChartPlugin, PieSeries, AccumulationDataLabel,
AccumulationLegend } from "@syncfusion/ej2-vue-charts";
import { ButtonPlugin } from "@syncfusion/ej2-vue-buttons";
Vue.use(ChartPlugin);
Vue.use(ButtonPlugin);
Vue.use(AccumulationChartPlugin);
export default {
  data: function() {
    return {
      seriesData: [
        { x: 'John', y: 10000 },
        { x: 'Jake', y: 12000 },
        { x: 'Peter', y: 18000 },
        { x: 'James', y: 11000 },
        { x: 'Mary', y: 9700 }
      ],
      seriesData1: [
        { x: new Date(2005, 0, 1), y: 21 }, { x: new Date(2006, 0, 1),
y: 24 },
        { x: new Date(2007, 0, 1), y: 36 }, { x: new Date(2008, 0, 1),
y: 38 },
        { x: new Date(2009, 0, 1), y: 54 }, { x: new Date(2010, 0, 1),
y: 57 },
        { x: new Date(2011, 0, 1), y: 70 }
      ],
      seriesData2: [
        { x: new Date(2005, 0, 1), y: 28 }, { x: new Date(2006, 0, 1),
y: 44 },
        { x: new Date(2007, 0, 1), y: 48 }, { x: new Date(2008, 0, 1),
y: 50 },
        { x: new Date(2009, 0, 1), y: 66 }, { x: new Date(2010, 0, 1),
y: 78 },
        { x: new Date(2011, 0, 1), y: 84 }
      ],
      seriesData3: [
        { x: 'Labour', y: 18, text: '18%' }, { x: 'Legal', y: 8, text:
'8%' },
        { x: 'Production', y: 15, text: '15%' }, { x: 'License', y: 11,
text: '11%' },
        { x: 'Facilities', y: 18, text: '18%' }, { x: 'Taxes', y: 14,
text: '14%' },
        { x: 'Insurance', y: 16, text: '16%' }
      ],
      primaryXAxis: {
        title: 'Manager',
        valueType: 'Category',
        majorGridLines: { width: 0 }
      },
      primaryYAxis: {
        title: 'Sales',
        minimum: 0,

```



```

        maximum: 20000,
        majorGridLines: { width: 0 }
    },
    marker: {
        visible: true,
        width: 10,
        height: 10
    },
    primaryXAxis1: {
        valueType: 'DateTime',
        labelFormat: 'Y',
        intervalType: 'Years',
        edgeLabelPlacement: 'Shift',
        majorGridLines: { width: 0 }
    },
    primaryYAxis1: {
        labelFormat: '{value}%',
        rangePadding: 'None',
        minimum: 0,
        maximum: 100,
        interval: 20,
        lineStyle: { width: 0 },
        majorTickLines: { width: 0 },
        minorTickLines: { width: 0 }
    },
    title: "Sales Comparision",
    title1: "Medal Count",
    title2: "Project Cost Breakdown",
    enableSmartLabels: true,
    datalabel: { visible: true, name: 'text', position: 'Inside', font:
{ fontWeight: 600, color: '#ffffff' } },
    legendSettings: {
        visible: true
    },
    startAngle: '0',
    endAngle: '360'
    };
},
provide: {
    chart: [ColumnSeries, Category, DateTime, Export, LineSeries],
    accumulationchart: [PieSeries, AccumulationDataLabel,
AccumulationLegend]
},
methods: {
    onClick: function(args) {
        let chart1 = document.getElementById("container").ej2_instances[0];
        let chart2 = document.getElementById("container1").ej2_instances[0];
        let chart3 = document.getElementById("container2").ej2_instances[0];
        chart1.exportModule.export('PDF', 'Chart', null, [chart1, chart2,
chart3], null, null, true, undefined, undefined, true);
    },
}
};
</script>
<style>
#container {
    height: 350px;

```

```

}
.e-export-icon::before {
  content: '\e728';
}
</style>

```

{% previewsample "page.domainurl/code-snippet/chart/series/polar-cs15" %}

### Multiple chart export

You can export the multiple charts in single page by passing the multiple chart objects in the export method of chart. To export multiple charts in a single page, follow the given steps:

Initially, render more than one chart to export, and then add button to export the multiple charts. In button click, call the export method in charts, and then pass the multiple chart objects in the export method.

### APP.VUE

```

<template>
  <div id="app">
    <ejs-button cssClass="e-flat" iconCss='e-icons e-export-icon'
isPrimary=true v-on:click.native='onClick' id="exportBtn">EXPORT</ejs-
button>
    <ejs-chart ref="chart" id="container":title='title'
:primaryXAxis='primaryXAxis'>
      <e-series-collection>
        <e-series :dataSource='seriesData' type='Column'
xName='country' yName='gold' name='Gold'> </e-series>
      </e-series-collection>
    </ejs-chart>
    <ejs-chart ref="chart1" id="container1":title='title'
:primaryXAxis='primaryXAxis'>
      <e-series-collection>
        <e-series :dataSource='seriesData' type='Column'
xName='country' yName='gold' name='Gold'> </e-series>
      </e-series-collection>
    </ejs-chart>
  </div>
</template>
<script>
import Vue from "vue";
import { ChartPlugin, ColumnSeries, Category, Export } from
"@syncfusion/ej2-vue-charts";
import { ButtonPlugin } from "@syncfusion/ej2-vue-buttons";
Vue.use(ChartPlugin);
Vue.use(ButtonPlugin);
export default {
  data: function() {
    return {
      seriesData: [
        { country: "USA", gold: 50 },
        { country: "China", gold: 40 },
        { country: "Japan", gold: 70 },
        { country: "Australia", gold: 60 },
        { country: "France", gold: 50 },
        { country: "Germany", gold: 40 },

```

```

        { country: "Italy", gold: 40 },
        { country: "Sweden", gold: 30 }
      ],
      primaryXAxis: {
        valueType: 'Category',
        title: 'Countries'
      },
      title: "Olympic Medals"
    };
  },
  provide: {
    chart: [ColumnSeries, Category, Export]
  },
  methods: {
    onClick: function(args) {
      let chart1 = document.getElementById("container").ej2_instances[0];
      let chart2 = document.getElementById("container1").ej2_instances[0];
      chart1.exportModule.export('PNG', 'Chart', null, [chart1, chart2]);
    },
  }
};
</script>
<style>
  #container {
    height: 350px;
  }
  .e-export-icon::before {
    content: '\e728';
  }
</style>

```

{% previewsample "page.domainurl/code-snippet/chart/series/column-cs12" %}

### Exporting chart using base64 string

The chart can be exported as an image in the form of a basering by utilizing HTML canvas. This process involves rendering the chart onto a canvas element and then converting the canvas content to a base64 string.

### APP.VUE

```

<template>
  <div id="app">
    <ejs-chart style='display:block' align='center' id='chartcontainer'
      :primaryXAxis='primaryXAxis' :primaryYAxis='primaryYAxis'
      :chartArea='chartArea'>
      <e-series-collection>
        <e-series :dataSource='seriesData' type='Column' xName='x'
yName='y'> </e-series>
      </e-series-collection>
    </ejs-chart>
    <ejs-button id='togglebtn' @click='print' cssClass="e-flat"
      :iconCss='iconCss'
        isPrimary="true">Print</ejs-button>
  </div>
</template>
<script>

```

```

import Vue from "vue";
import { ChartComponent, ChartPlugin, SeriesDirective,
SeriesCollectionDirective, ColumnSeries, Legend, Tooltip, DateTime, Category,
Highlight } from "@syncfusion/ej2-vue-charts";
Vue.use(ChartPlugin);
export default {
  data: function() {
    return {
      seriesData: [
        { x: 'DEU', y: 35.5 }, { x: 'CHN', y: 18.3 }, { x: 'ITA', y: 17.6 }, {
x: 'JPN', y: 13.6 },
        { x: 'US', y: 12 }, { x: 'ESP', y: 5.6 }, { x: 'FRA', y: 4.6 }, { x:
'AUS', y: 3.3 },
        { x: 'BEL', y: 3 }, { x: 'UK', y: 2.9 }
      ],
      //Initializing Primary X Axis
      primaryXAxis: {
        valueType: 'Category',
        majorGridLines: { width: 0 },
        majorTickLines: { width: 0 },
        minorTickLines: { width: 0 }
      },
      //Initializing Primary Y Axis
      primaryYAxis: {
        minimum: 0,
        maximum: 40,
        interval: 10,
        lineStyle: {width : 0},
        minorTickLines: {width: 0},
        majorTickLines: {width : 0},
      },
      chartArea: {
        border: {
          width: 0
        }
      },
    };
  },
  provide: {
    chart: [ColumnSeries, Legend, Tooltip, Category, DateTime, Highlight]
  },
  methods: {
    print: function (args) {
      var svg = document.querySelector("#chartcontainer_svg");
      var svgData = new XMLSerializer().serializeToString(svg);
      var canvas = document.createElement("canvas");
      document.body.appendChild(canvas);
      var svgSize = svg.getBoundingClientRect();
      canvas.width = svgSize.width;
      canvas.height = svgSize.height;
      var ctx = canvas.getContext("2d");
      var img = document.createElement("img");
      img.setAttribute("src", "data:image/svg+xml;base64," +
btoa(svgData));
      img.onload = function() {
        ctx.drawImage(img, 0, 0);
      };
    }
  }
};

```

```

        var imagedata = canvas.toDataURL("image/png");
        console.log(imagedata); // printed base64 in console
        canvas.remove();
    };
},
}
};
</script>
<style>
#container {
    height: 350px;
}
.e-export-icon::before {
    content: '\e728';
}
</style>

```

{% previewsample "page.domainurl/code-snippet/chart/export-cs1" %}

## Chart appearance in Vue Chart component

### Custom color palette

You can customize the default color of series or points by providing a custom color palette of your choice by

using the [palettes](#) property.

### APP.VUE

```

<template>
  <div id="app">
    <ejs-chart id="container" :title='title'
:primaryXAxis='primaryXAxis' :primaryYAxis='primaryYAxis'
:palettes='palettes'>
      <e-series-collection>
        <e-series :dataSource='seriesData' type='Column'
xName='country' yName='gold' name='Gold'> </e-series>
        <e-series :dataSource='seriesData' type='Column'
xName='country' yName='silver' name='Silver'> </e-series>
        <e-series :dataSource='seriesData' type='Column'
xName='country' yName='bronze' name='Bronze'> </e-series>
      </e-series-collection>
    </ejs-chart>
  </div>
</template>
<script>
import Vue from "vue";
import { ChartPlugin, ColumnSeries, Category } from "@syncfusion/ej2-vue-
charts";
Vue.use(ChartPlugin);
export default {
  data() {
    return {
      seriesData: [
        { country: "USA", gold: 50, silver: 70, bronze: 45 },
        { country: "China", gold: 40, silver: 60, bronze: 55 },
        { country: "Japan", gold: 70, silver: 60, bronze: 50 },

```

```

        { country: "Australia", gold: 60, silver: 56, bronze: 40 },
        { country: "France", gold: 50, silver: 45, bronze: 35 },
        { country: "Germany", gold: 40, silver: 30, bronze: 22 },
        { country: "Italy", gold: 40, silver: 35, bronze: 37 },
        { country: "Sweden", gold: 30, silver: 25, bronze: 27 }
    ],
    primaryXAxis: {
        valueType: 'Category',
        title: 'Countries'
    },
    primaryYAxis:
    {
        minimum: 0, maximum: 80,
        interval: 20, title: 'Medals',
        labelFormat: '${value}K'
    },
    palettes: ["#E94649", "#F6B53F", "#6FAAB0", "#C4C24A"],
    title: "Olympic Medals"
};
},
provide: {
    chart: [ColumnSeries, Category]
}
};
</script>
<style>
    #container {
        height: 350px;
    }
</style>

```

{% previewsample "page.domainurl/code-snippet/chart/axis/category-cs24" %}

### Data point customization

The color of individual data point or data points within a range can be customized using the options below.

#### Point color mapping

You can bind the color for the points from [dataSource](#) for the series using [pointColorMapping](#) property.

### APP.VUE

```

<template>
    <div id="app">
        <ejs-chart id='chartcontainer' :title='title'
        :primaryXAxis='primaryXAxis' :primaryYAxis='primaryYAxis'
        :chartArea='chartArea'>
            <e-series-collection>
                <e-series :pointColorMapping='pointColorMapping'
                :dataSource='seriesData' type='Column' xName='x' yName='y'
                :cornerRadius='cornerRadius'> </e-series>
            </e-series-collection>
        </ejs-chart>
    </div>
</template>
<script>

```

```

import Vue from "vue";
import { ChartPlugin, ColumnSeries, Category } from "@syncfusion/ej2-vue-charts";
Vue.use(ChartPlugin);
export default {
  data() {
    return {
      seriesData: [
        { x: 'Jan', y: 6.96, color: "red" },
        { x: 'Feb', y: 8.9, color: "blue" },
        { x: 'Mar', y: 12, color: "orange" },
        { x: 'Apr', y: 17.5, color: "aqua" },
        { x: 'May', y: 22.1, color: "grey" }
      ],
      primaryXAxis: {
        valueType: 'Category', majorGridLines: { width: 0 }
      },
      pointColorMapping: "color",
      primaryYAxis: {
        lineStyle: { width: 0 },
        majorTickLines: { width: 0 },
        minorTickLines: { width: 0 },
        labelFormat: '{value}°C',
      },
      chartArea: {
        border: {
          width: 0
        }
      },
      title: "USA CLIMATE - WEATHER BY MONTH",
      cornerRadius: {
        topLeft: 10, topRight: 10
      }
    };
  },
  provide: {
    chart: [ColumnSeries, Category]
  }
};
</script>
<style>
  #container {
    height: 350px;
  }
</style>

```

{% previewsample "page.domainurl/code-snippet/chart/axis/category-cs25" %}

#### Range color mapping

You can differentiate data points based on their y values using [rangeColorSettings](#) in the chart.

#### APP.VUE

```

<template>
  <div id="app">

```

```

    <ejs-chart id='chartcontainer' :title='title'
:primaryXAxis='primaryXAxis' :primaryYAxis='primaryYAxis'
      :chartArea='chartArea' :legendSettings='legendSettings'
      :selectionMode="selectionMode">
      <e-series-collection>
        <e-series :dataSource='seriesData' type='Column' xName='x'
yName='y' name='USA' :animation='animation' :cornerRadius='cornerRadius'>
      </e-series>
      </e-series-collection>
      <e-rangecolorsettings>
        <e-rangecolorsetting label="1°C to 10°C" start="1" end="10"
:colors="colors1"></e-rangecolorsetting>
        <e-rangecolorsetting label="11°C to 20°C" start="11"
end="20" :colors="colors2"></e-rangecolorsetting>
        <e-rangecolorsetting label="21°C to 30°C" start="21"
end="30" :colors="colors3"></e-rangecolorsetting>
      </e-rangecolorsettings>
    </ejs-chart>
  </div>
</template>
<script>
import Vue from "vue";
import { ChartPlugin, ColumnSeries, Category, Legend, Selection } from
"@syncfusion/ej2-vue-charts";
Vue.use(ChartPlugin);
export default {
  data() {
    return {
      seriesData: [
        { x: "Jan", y: 6.96 },
        { x: "Feb", y: 8.9 },
        { x: "Mar", y: 12 },
        { x: "Apr", y: 17.5 },
        { x: "May", y: 22.1 },
        { x: "June", y: 25 },
        { x: "July", y: 29.4 },
        { x: "Aug", y: 29.6 },
        { x: "Sep", y: 25.8 },
        { x: "Oct", y: 21.1 },
        { x: "Nov", y: 15.5 },
        { x: "Dec", y: 9.9 }
      ],
      primaryXAxis: {
        valueType: 'Category', majorGridLines: { width: 0 }
      },
      primaryYAxis: {
        lineStyle: { width: 0 },
        majorTickLines: { width: 0 },
        minorTickLines: { width: 0 },
        labelFormat: '{value}°C',
      },
      chartArea: {
        border: {
          width: 0
        }
      },
      tooltip: {

```



```

        enable: false
      },
      title: "USA CLIMATE - WEATHER BY MONTH",
      legendSettings: {
        mode: 'Range',
        visible: true,
        toggleVisibility: false,
      },
      marker: {
        dataLabel: {
          visible: true,
          position: 'Outer',
        }
      },
      selectionMode: 'Point',
      animation: {
        enable: false
      },
      cornerRadius: {
        topLeft: 10, topRight: 10
      },
      colors1: ["#F9D422"],
      colors2: ["#F28F3F"],
      colors3: ["#E94F53"]
    };
  },
  provide: {
    chart: [ColumnSeries, Category, Legend, Selection]
  }
};
</script>
<style>
  #container {
    height: 350px;
  }
</style>

```

{% previewsample "page.domainurl/code-snippet/chart/axis/category-cs26" %}

### Point level customization

Marker, datalabel and fill color of each data point can be customized with

[pointRender](#) and

[textRender](#) event.

### APP.VUE

```

<template>
  <div id="app">
    <ejs-chart id="container" :title='title'
:primaryXAxis='primaryXAxis' :primaryYAxis='primaryYAxis'
:pointRender='pointRender'>
      <e-series-collection>
        <e-series :dataSource='seriesData' type='Column'
xName='country' yName='gold' name='Gold'> </e-series>
      </e-series-collection>
    </div>
  </template>

```

```

        </ejs-chart>
    </div>
</template>
<script>
import Vue from "vue";
import { ChartPlugin, ColumnSeries, Category } from "@syncfusion/ej2-vue-charts";
Vue.use(ChartPlugin);
export default {
    data() {
        return {
            seriesData: [
                { country: "USA", gold: 50 },
                { country: "China", gold: 40 },
                { country: "Japan", gold: 70 },
                { country: "Australia", gold: 60 },
                { country: "France", gold: 50 },
                { country: "Germany", gold: 40 },
                { country: "Italy", gold: 40 },
                { country: "Sweden", gold: 30 }
            ],
            primaryXAxis: {
                valueType: 'Category',
                title: 'Countries'
            },
            primaryYAxis: {
                minimum: 0, maximum: 80,
                interval: 20, title: 'Medals'
            },
            title: "Olympic Medals"
        };
    },
    provide: {
        chart: [ColumnSeries, Category]
    },
    methods: {
        pointRender: function(args) {
            var seriesColor = ['#00bdae', '#404041', '#357cd2', '#e56590', '#f8b883',
                '#70ad47', '#dd8abd', '#7f84e8', '#7bb4eb', '#ea7a57'];
            args.fill = seriesColor[args.point.index];
        },
    },
};
</script>
<style>
    #container {
        height: 350px;
    }
</style>

```

{% previewsample "page.domainurl/code-snippet/chart/series/column-cs6" %}

<!-- markdownlint-disable MD036 -->

### Chart area customization

<!-- markdownlint-disable MD036 -->

#### Customize the chart background

<!-- markdownlint-disable MD013 -->

Using [background](#) and [border](#) properties, you can change the background color and border of the chart.

#### APP.VUE

```
<template>
  <div id="app">
    <ejs-chart id="container" :title='title'
:primaryXAxis='primaryXAxis' :primaryYAxis='primaryYAxis'
background='skyblue' :border='border'>
      <e-series-collection>
        <e-series :dataSource='seriesData' type='Column'
xName='country' yName='gold' name='Gold'> </e-series>
      </e-series-collection>
    </ejs-chart>
  </div>
</template>
<script>
import Vue from "vue";
import { ChartPlugin, ColumnSeries, Category } from "@syncfusion/ej2-vue-
charts";
Vue.use(ChartPlugin);
export default {
  data() {
    return {
      seriesData: [
        { country: "USA", gold: 50 },
        { country: "China", gold: 40 },
        { country: "Japan", gold: 70 },
        { country: "Australia", gold: 60 },
        { country: "France", gold: 50 },
        { country: "Germany", gold: 40 },
        { country: "Italy", gold: 40 },
        { country: "Sweden", gold: 30 }
      ],
      primaryXAxis: {
        valueType: 'Category',
        title: 'Countries'
      },
      primaryYAxis: {
        minimum: 0, maximum: 80,
        interval: 20, title: 'Medals'
      },
      border: {color: "#FF0000", width: 2},
      title: "Olympic Medals"
    };
  },
  provide: {
    chart: [ColumnSeries, Category]
  }
};
</script>
```

```
<style>
  #container {
    height: 350px;
  }
</style>
```

{% previewsample "page.domainurl/code-snippet/chart/series/column-cs7" %}

### Chart margin

You can set margin for chart from its container through [margin](#) property.

### APP.VUE

```
<template>
  <div id="app">
    <ejs-chart id="container" :title='title'
    :primaryXAxis='primaryXAxis' :primaryYAxis='primaryYAxis'
    background='skyblue' :border='border'
    :margin='margin'>
      <e-series-collection>
        <e-series :dataSource='seriesData' type='Column'
        xName='country' yName='gold' name='Gold'> </e-series>
      </e-series-collection>
    </ejs-chart>
  </div>
</template>
<script>
import Vue from "vue";
import { ChartPlugin, ColumnSeries, Category } from "@syncfusion/ej2-vue-
charts";
Vue.use(ChartPlugin);
export default {
  data() {
    return {
      seriesData: [
        { country: "USA", gold: 50 },
        { country: "China", gold: 40 },
        { country: "Japan", gold: 70 },
        { country: "Australia", gold: 60 },
        { country: "France", gold: 50 },
        { country: "Germany", gold: 40 },
        { country: "Italy", gold: 40 },
        { country: "Sweden", gold: 30 }
      ],
      primaryXAxis: {
        valueType: 'Category',
        title: 'Countries'
      },
      primaryYAxis: {
        minimum: 0, maximum: 80,
        interval: 20, title: 'Medals'
      },
      border: {color: "#FF0000", width: 2},
      margin: { left: 40, right: 40, top: 40, bottom: 40 },
      title: "Olympic Medals"
    }
  }
}
```

```

    };
  },
  provide: {
    chart: [ColumnSeries, Category]
  }
};
</script>
<style>
  #container {
    height: 350px;
  }
</style>

```

{% previewsample "page.domainurl/code-snippet/chart/series/column-cs8" %}

### Chart area customization

Using [background](#) and [border](#) properties, you can change the background color and border of the chart area. Width for the chart area can be customized using [width](#) property.

### APP.VUE

```

<template>
  <div id="app">
    <ejs-chart id="container" :title='title'
      :primaryXAxis='primaryXAxis' :primaryYAxis='primaryYAxis'
      background='skyblue' :chartArea='chartArea'>
      <e-series-collection>
        <e-series :dataSource='seriesData' type='Column'
          xName='country' yName='gold' name='Gold'> </e-series>
      </e-series-collection>
    </ejs-chart>
  </div>
</template>
<script>
import Vue from "vue";
import { ChartPlugin, ColumnSeries, Category } from "@syncfusion/ej2-vue-charts";
Vue.use(ChartPlugin);
export default {
  data() {
    return {
      seriesData: [
        { country: "USA", gold: 50 },
        { country: "China", gold: 40 },
        { country: "Japan", gold: 70 },
        { country: "Australia", gold: 60 },
        { country: "France", gold: 50 },
        { country: "Germany", gold: 40 },
        { country: "Italy", gold: 40 },
        { country: "Sweden", gold: 30 }
      ],
      primaryXAxis: {
        valueType: 'Category',
        title: 'Countries'
      },
      primaryYAxis: {

```

```

        minimum: 0, maximum: 80,
        interval: 20, title: 'Medals'
    },
    chartArea: {
        //background for Chart area
        background: "skyblue",
        width: '90%'
    },
    title: "Olympic Medals"
};
},
provide: {
    chart: [ColumnSeries, Category]
}
};
</script>
<style>
    #container {
        height: 350px;
    }
</style>

```

{% previewsample "page.domainurl/code-snippet/chart/series/column-cs9" %}

### Animation

You can customize animation for a particular series using [animation](#) property. You can enable or disable animation of the series using `enable` property, `duration` specifies the duration of an animation and `delay` allows us to start the animation at desire time.

### APP.VUE

```

<template>
    <div id="app">
        <ejs-chart id="container" :title='title'
        :primaryXAxis='primaryXAxis' :primaryYAxis='primaryYAxis'
        background='skyblue'>
            <e-series-collection>
                <e-series :dataSource='seriesData' type='Column'
                xName='country' yName='gold' name='Gold'
                :border='border' :animation='animation'> </e-series>
            </e-series-collection>
        </ejs-chart>
    </div>
</template>
<script>
import Vue from "vue";
import { ChartPlugin, ColumnSeries, Category } from "@syncfusion/ej2-vue-
charts";
Vue.use(ChartPlugin);
export default {
    data() {
        return {
            seriesData: [
                { country: "USA", gold: 50 },
                { country: "China", gold: 40 },
                { country: "Japan", gold: 70 },
            ]
        }
    }
}

```

```

        { country: "Australia", gold: 60 },
        { country: "France", gold: 50 },
        { country: "Germany", gold: 40 },
        { country: "Italy", gold: 40 },
        { country: "Sweden", gold: 30 }
      ],
      primaryXAxis: {
        valueType: 'Category',
        title: 'Countries'
      },
      primaryYAxis: {
        minimum: 0, maximum: 80,
        interval: 20, title: 'Medals'
      },
      border:{ width: 2, color: 'grey'},
      //Animation for chart series
      animation:{
        enable: true,
        duration: 2000,
        delay: 200
      },
      title: "Olympic Medals"
    };
  },
  provide: {
    chart: [ColumnSeries, Category]
  }
};
</script>
<style>
  #container {
    height: 350px;
  }
</style>

```

{% previewsample "page.domainurl/code-snippet/chart/series/column-cs10" %}

#### Fluid animation

Fluid animation used to animate series with updated dataSource continues animation rather than animation whole series. You can customize animation for a particular series using `animate` method.

#### APP.VUE

```

<template>
  <div id="app">
    <ejs-chart id="container" ref="chart" :title='title'
    :primaryXAxis='primaryXAxis' :primaryYAxis='primaryYAxis' :tooltip='tooltip'
    :chartArea='chartArea'
    :legendSettings='legendSettings' :loaded='loaded'>
      <e-series-collection>
        <e-series :dataSource='seriesData' :marker='marker'
        type='Column' xName='x' yName='y' name='Tiger' width='1'
        :cornerRadius="cornerRadius"> </e-series>
      </e-series-collection>
    </ejs-chart>
  </div>

```

```

</template>
<script>
import Vue from "vue";
import { ChartPlugin,
  ColumnSeries,
  Category,
  DataLabel,
  Tooltip,
  Legend } from "@syncfusion/ej2-vue-charts";
Vue.use(ChartPlugin);
let count = 0;
let datasources1 = [
  { x: 'Egg', y: 206 },
  { x: 'Fish', y: 123 },
  { x: 'Misc', y: 48 },
  { x: 'Tea', y: 240 },
  { x: 'Fruit', y: 170 }
];
let datasource2 = [
  { x: 'Egg', y: 86 },
  { x: 'Fish', y: 173 },
  { x: 'Misc', y: 188 },
  { x: 'Tea', y: 109 },
  { x: 'Fruit', y: 100 }
];
let datasource3 = [
  { x: 'Egg', y: 156 },
  { x: 'Fish', y: 33 },
  { x: 'Misc', y: 260 },
  { x: 'Tea', y: 200 },
  { x: 'Fruit', y: 30 }
];
export default {
  data() {
    return {
      seriesData: [
        { x: 'Egg', y: 106 },
        { x: "Fish", y: 103 },
        { x: "Misc", y: 198 },
        { x: "Tea", y: 189 },
        { x: "Fruit", y: 250 }
      ],
      primaryXAxis: {
        valueType: "Category",
        interval: 1,
        majorGridLines: { width: 0 },
        tickPosition: "Inside",
        labelPosition: "Inside",
        labelStyle: { color: "#ffffff" }
      },
      primaryYAxis: {
        minimum: 0,
        maximum: 300,
        interval: 50,
        majorGridLines: { width: 0 },
        majorTickLines: { width: 0 },
        lineStyle: { width: 0 },

```



```

        labelStyle: { color: "transparent" }
    },
    legendSettings: { visible: false },
    tooltip: {
        enable: false
    },
    cornerRadius: {
        bottomLeft: 10,
        bottomRight: 10,
        topLeft: 10,
        topRight: 10
    },
    marker: {
        dataLabel: {
            visible: true,
            position: "Top",
        }
    },
    chartArea: { border: { width: 0 } },
    title: "Trade in Food Groups"
};
},
provide: {
    chart: [ColumnSeries, Legend, DataLabel, Category, Tooltip]
},
methods: {
    loaded: function(args) {
        this.$refs.chart.ej2Instances.loaded = null;
        let columninterval = setInterval(() => {
            if (document.getElementById('container')) {
                if (count === 0) {
                    this.$refs.chart.ej2Instances.series[0].dataSource = datasources1;
                    this.$refs.chart.ej2Instances.animate();
                    count++;
                } else if (count === 1) {
                    this.$refs.chart.ej2Instances.series[0].dataSource = datasources2;
                    this.$refs.chart.ej2Instances.animate();
                    count++;
                } else if (count === 2) {
                    this.$refs.chart.ej2Instances.series[0].dataSource = datasources3;
                    this.$refs.chart.ej2Instances.animate();
                    count = 0;
                } else {
                    clearInterval(columninterval)
                }
            }
        }, 2000);
    }
};
</script>
<style>
    #container {
        height: 350px;
    }
</style>

```

{% previewsample "page.domainurl/code-snippet/chart/series/column-cs11" %}

### Chart title

Chart can be given a title using [title](#) property, to show the information about the data plotted.

### APP.VUE

```
<template>
  <div id="app">
    <ejs-chart id="container" :title='title'
:primaryXAxis='primaryXAxis' :primaryYAxis='primaryYAxis'
:titleStyle='titleStyle'>
      <e-series-collection>
        <e-series :dataSource='seriesData' type='StepLine' xName='x'
yName='y' width=2 name='China' :marker='marker'> </e-series>
        <e-series :dataSource='seriesData' type='StepLine'
xName='x' yName='y1' width=2 name='Australia' :marker='marker'> </e-series>
        <e-series :dataSource='seriesData' type='StepLine'
xName='x' yName='y2' width=2 name='Japan' :marker='marker'> </e-series>
      </e-series-collection>
    </ejs-chart>
  </div>
</template>
<script>
import Vue from "vue";
import { ChartPlugin, StepLineSeries, DateTime } from "@syncfusion/ej2-vue-
charts";
Vue.use(ChartPlugin);
export default {
  data() {
    return {
      seriesData: [
        { x: new Date(1975, 0, 1), y: 16, y1: 10, y2: 4.5 },
        { x: new Date(1980, 0, 1), y: 12.5, y1: 7.5, y2: 5 },
        { x: new Date(1985, 0, 1), y: 19, y1: 11, y2: 6.5 },
        { x: new Date(1990, 0, 1), y: 14.4, y1: 7, y2: 4.4 },
        { x: new Date(1995, 0, 1), y: 11.5, y1: 8, y2: 5 },
        { x: new Date(2000, 0, 1), y: 14, y1: 6, y2: 1.5 },
        { x: new Date(2005, 0, 1), y: 10, y1: 3.5, y2: 2.5 },
        { x: new Date(2010, 0, 1), y: 16, y1: 7, y2: 3.7 }
      ],
      primaryXAxis: {
        title: 'Years',
        lineStyle: { width: 0 },
        labelFormat: 'y',
        intervalType: 'Years',
        valueType: 'DateTime',
        edgeLabelPlacement: 'Shift'
      },
      primaryYAxis: {
        title: 'Percentage (%)',
        minimum: 0, maximum: 20, interval: 2,
        labelFormat: '{value}%'
      },
      marker: {
```

```

        visible: true, width: 10, height: 10
      },
      title: "Unemployment Rates 1975-2010",
      titleStyle: {
        fontFamily: "Arial",
        fontStyle: 'italic',
        fontWeight: 'regular',
        color: "#E27F2D",
        size: '23px'
      }
    };
  },
  provide: {
    chart: [StepLineSeries, DateTime]
  }
};
</script>
<style>
  #container {
    height: 350px;
  }
</style>

```

{% previewsample "page.domainurl/code-snippet/chart/chart-title-cs5" %}

#### Title position

By using the [position](#) property in [titleStyle](#), you can position the [title](#) at left, right, top or bottom of the chart. The title is positioned at the top of the chart, by default.

#### APP.VUE

```

<template>
  <div id="app">
    <ejs-chart id="container" :title='title'
      :primaryXAxis='primaryXAxis' :primaryYAxis='primaryYAxis'
      :titleStyle='titleStyle'>
      <e-series-collection>
        <e-series :dataSource='seriesData' type='StepLine' xName='x'
          yName='y' width=2 name='China' :marker='marker'> </e-series>
        <e-series :dataSource='seriesData' type='StepLine'
          xName='x' yName='y1' width=2 name='Australia' :marker='marker'> </e-series>
        <e-series :dataSource='seriesData' type='StepLine'
          xName='x' yName='y2' width=2 name='Japan' :marker='marker'> </e-series>
      </e-series-collection>
    </ejs-chart>
  </div>
</template>
<script>
import Vue from "vue";
import { ChartPlugin, StepLineSeries, DateTime } from "@syncfusion/ej2-vue-charts";
Vue.use(ChartPlugin);
export default {
  data() {
    return {
      seriesData: [

```

```

        { x: new Date(1975, 0, 1), y: 16, y1: 10, y2: 4.5 },
        { x: new Date(1980, 0, 1), y: 12.5, y1: 7.5, y2: 5 },
        { x: new Date(1985, 0, 1), y: 19, y1: 11, y2: 6.5 },
        { x: new Date(1990, 0, 1), y: 14.4, y1: 7, y2: 4.4 },
        { x: new Date(1995, 0, 1), y: 11.5, y1: 8, y2: 5 },
        { x: new Date(2000, 0, 1), y: 14, y1: 6, y2: 1.5 },
        { x: new Date(2005, 0, 1), y: 10, y1: 3.5, y2: 2.5 },
        { x: new Date(2010, 0, 1), y: 16, y1: 7, y2: 3.7 }
    ],
    primaryXAxis: {
        lineStyle: { width: 0 },
        labelFormat: 'Y',
        intervalType: 'Years',
        valueType: 'DateTime',
        edgeLabelPlacement: 'Shift'
    },
    primaryYAxis: {
        title: 'Percentage (%)',
        minimum: 0, maximum: 20, interval: 2,
        labelFormat: '{value}%'
    },
    marker: {
        visible: true, width: 10, height: 10
    },
    title: "Unemployment Rates 1975-2010",
    titleStyle: {
        position: 'Bottom'
    }
};
},
provide: {
    chart: [StepLineSeries, DateTime]
}
};
</script>
<style>
    #container {
        height: 350px;
    }
</style>

```

{% previewsample "page.domainurl/code-snippet/chart/chart-title-cs8" %}

- The custom option helps you to position the title anywhere in the chart using [x](#) and [y](#) coordinates.

### APP.VUE

```

<template>
    <div id="app">
        <ejs-chart id="container" :title='title'
        :primaryXAxis='primaryXAxis' :primaryYAxis='primaryYAxis'
        :titleStyle='titleStyle'>
            <e-series-collection>

```

```

        <e-series :dataSource='seriesData' type='StepLine' xName='x'
yName='y' width=2 name='China' :marker='marker'> </e-series>
        <e-series :dataSource='seriesData' type='StepLine'
xName='x' yName='y1' width=2 name='Australia' :marker='marker'> </e-series>
        <e-series :dataSource='seriesData' type='StepLine'
xName='x' yName='y2' width=2 name='Japan' :marker='marker'> </e-series>
    </e-series-collection>
</ejs-chart>
</div>
</template>
<script>
import Vue from "vue";
import { ChartPlugin, StepLineSeries, DateTime } from "@syncfusion/ej2-vue-
charts";
Vue.use(ChartPlugin);
export default {
  data() {
    return {
      seriesData: [
        { x: new Date(1975, 0, 1), y: 16, y1: 10, y2: 4.5 },
        { x: new Date(1980, 0, 1), y: 12.5, y1: 7.5, y2: 5 },
        { x: new Date(1985, 0, 1), y: 19, y1: 11, y2: 6.5 },
        { x: new Date(1990, 0, 1), y: 14.4, y1: 7, y2: 4.4 },
        { x: new Date(1995, 0, 1), y: 11.5, y1: 8, y2: 5 },
        { x: new Date(2000, 0, 1), y: 14, y1: 6, y2: 1.5 },
        { x: new Date(2005, 0, 1), y: 10, y1: 3.5, y2: 2.5 },
        { x: new Date(2010, 0, 1), y: 16, y1: 7, y2: 3.7 }
      ],
      primaryXAxis: {
        title: 'Years',
        lineStyle: { width: 0 },
        labelFormat: 'y',
        intervalType: 'Years',
        valueType: 'DateTime',
        edgeLabelPlacement: 'Shift'
      },
      primaryYAxis: {
        title: 'Percentage (%)',
        minimum: 0, maximum: 20, interval: 2,
        labelFormat: '{value}%'
      },
      marker: {
        visible: true, width: 10, height: 10
      },
      title: "Unemployment Rates 1975-2010",
      titleStyle: {
        position: 'Custom',
        x: 300,
        y: 60
      }
    };
  },
  provide: {
    chart: [StepLineSeries, DateTime]
  }
};
</script>

```

```
<style>
  #container {
    height: 350px;
  }
</style>
```

```
{% previewsample "page.domainurl/code-snippet/chart/chart-title-cs9" %}
```

### Title alignment

You can align the title to the near, far, or center of the chart using the [textAlignment](#) property.

### APP.VUE

```
<template>
  <div id="app">
    <ejs-chart id="container" :title='title'
:primaryXAxis='primaryXAxis' :primaryYAxis='primaryYAxis'
:titleStyle='titleStyle'>
      <e-series-collection>
        <e-series :dataSource='seriesData' type='StepLine' xName='x'
yName='y' width=2 name='China' :marker='marker'> </e-series>
        <e-series :dataSource='seriesData' type='StepLine'
xName='x' yName='y1' width=2 name='Australia' :marker='marker'> </e-series>
        <e-series :dataSource='seriesData' type='StepLine'
xName='x' yName='y2' width=2 name='Japan' :marker='marker'> </e-series>
      </e-series-collection>
    </ejs-chart>
  </div>
</template>
<script>
import Vue from "vue";
import { ChartPlugin, StepLineSeries, DateTime } from "@syncfusion/ej2-vue-
charts";
Vue.use(ChartPlugin);
export default {
  data() {
    return {
      seriesData: [
        { x: new Date(1975, 0, 1), y: 16, y1: 10, y2: 4.5 },
        { x: new Date(1980, 0, 1), y: 12.5, y1: 7.5, y2: 5 },
        { x: new Date(1985, 0, 1), y: 19, y1: 11, y2: 6.5 },
        { x: new Date(1990, 0, 1), y: 14.4, y1: 7, y2: 4.4 },
        { x: new Date(1995, 0, 1), y: 11.5, y1: 8, y2: 5 },
        { x: new Date(2000, 0, 1), y: 14, y1: 6, y2: 1.5 },
        { x: new Date(2005, 0, 1), y: 10, y1: 3.5, y2: 2.5 },
        { x: new Date(2010, 0, 1), y: 16, y1: 7, y2: 3.7 }
      ],
      primaryXAxis: {
        lineStyle: { width: 0 },
        labelFormat: 'y',
        intervalType: 'Years',
        valueType: 'DateTime',
        edgeLabelPlacement: 'Shift'
      },
      primaryYAxis: {
        title: 'Percentage (%)',

```

```

        minimum: 0, maximum: 20, interval: 2,
        labelFormat: '{value}%'
    },
    marker: {
        visible: true, width: 10, height: 10
    },
    title: "Unemployment Rates 1975-2010",
    titleStyle: {
        position: 'Bottom',
        textAlignment: 'Far'
    }
};
},
provide: {
    chart: [StepLineSeries, DateTime]
}
};
</script>
<style>
    #container {
        height: 350px;
    }
</style>

```

{% previewsample "page.domainurl/code-snippet/chart/chart-title-cs10" %}

### Chart subTitle

Chart can be given a subtitle using [subTitle](#) property, to show the information about the data plotted.

### APP.VUE

```

<template>
    <div id="app">
        <ejs-chart id="container" :title='title' :subTitle='subTitle'
        :subTitleStyle='subTitleStyle' :primaryXAxis='primaryXAxis'
        :axisLabelRender='axisLabelRender'>
            <e-series-collection>
                <e-series :dataSource='seriesData' type='Column'
                xName='country' yName='gold' name='Gold'> </e-series>
            </e-series-collection>
        </ejs-chart>
    </div>
</template>
<script>
import Vue from "vue";
import { ChartPlugin, ColumnSeries, Category } from "@syncfusion/ej2-vue-
charts";
Vue.use(ChartPlugin);
export default {
    data() {
        return {
            seriesData: [
                { country: "USA", gold: 50 },
                { country: "China", gold: 40 },
                { country: "Japan", gold: 70 },
                { country: "Australia", gold: 60 },
            ]
        }
    }
}

```

```

        { country: "France", gold: 50 },
        { country: "Germany", gold: 40 },
        { country: "Italy", gold: 40 },
        { country: "Sweden", gold: 30 }
      ],
      primaryXAxis: {
        valueType: 'Category',
        title: 'Countries'
      },
      title: "Olympic Medals",
      subTitle: 'In the year 2014',
      subTitleStyle: {
        fontFamily: "Arial",
        fontStyle: 'italic',
        fontWeight: 'regular',
        color: "#E27F2D"
      }
    }
  };
},
provide: {
  chart: [ColumnSeries, Category]
},
methods: {
  axisLabelRender: function(args) {
    if(args.text === 'France') {
      args.labelStyle.color = 'Red';
    }
  }
}
};
</script>
<style>
  #container {
    height: 350px;
  }
</style>

```

{% previewsample "page.domainurl/code-snippet/chart/chart-title-cs6" %}

#### Title wrap

Chart can be given a title using [title](#) property, to show the information about the data plotted.

#### APP.VUE

```

<template>
  <div id="app">
    <ejs-chart id="container" :title='title'
    :primaryXAxis='primaryXAxis' :titleStyle='titleStyle'>
      <e-series-collection>
        <e-series :dataSource='seriesData' type='Line' xName='month'
        yName='sales' name='Gold'
        :marker='marker'> </e-series>
      </e-series-collection>
    </ejs-chart>
  </div>
</template>

```



```

<script>
import Vue from "vue";
import { ChartPlugin, LineSeries, Category } from "@syncfusion/ej2-vue-charts";
Vue.use(ChartPlugin);
export default {
  data() {
    return {
      seriesData: [
        { month: 'Jan', sales: 35 }, { month: 'Feb', sales: 28 },
        { month: 'Mar', sales: 34 }, { month: 'Apr', sales: 32 },
        { month: 'May', sales: 40 }, { month: 'Jun', sales: 32 },
        { month: 'Jul', sales: 35 }, { month: 'Aug', sales: 55 },
        { month: 'Sep', sales: 38 }, { month: 'Oct', sales: 30 },
        { month: 'Nov', sales: 25 }, { month: 'Dec', sales: 32 }
      ],
      primaryXAxis: {
        valueType: 'Category',
      },
      title: "Unemployment Rates 1975-2010",
      marker: {
        visible: true, width: 10, height: 10
      },
      titleStyle: {
        size: '18px', color: 'Red', textAlignment: 'Far', textOverflow:
'Wrap'
      }
    };
  },
  provide: {
    chart: [LineSeries, Category]
  }
};
</script>
<style>
#container {
  height: 350px;
}
</style>

```

{% previewsample "page.domainurl/code-snippet/chart/chart-title-cs7" %}

<!-- markdownlint-disable MD036 -->

## Render methods in Vue Chart component

Chart uses following two rendering methods.

- SVG
- Canvas

### SVG

SVG is used to render Chart by default for all browsers except IE8 and old versions.

## Canvas

You can switch between SVG and Canvas rendering by using the `enableCanvas` option. The canvas mode rendering is used in the following scenarios,

- Plotting large number of data points.
- Performing high frequency live updates.

## Limitations

- Animation is not supported.

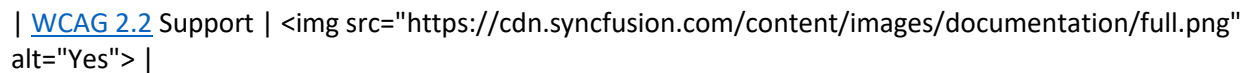
## Accessibility in Vue Chart component

The Chart component followed the accessibility guidelines and standards, including [ADA, Section 508](#), [WCAG 2.2](#) standards, and [WCAG roles](#) that are commonly used to evaluate accessibility.

The accessibility compliance for the Chart component is outlined below.

| Accessibility Criteria | Compatibility |

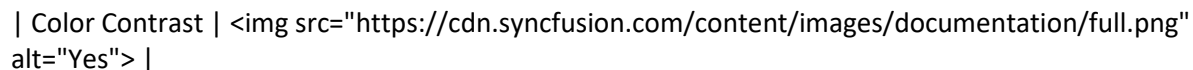
| -- | -- |

| [WCAG 2.2](#) Support |  |

| [Section 508](#) Support |  |

| Screen Reader Support |  |

| Right-To-Left Support |  |

| Color Contrast |  |

| Mobile Device Support |  |

| Keyboard Navigation Support |  |

| [Accessibility Checker](#) Validation |  |

| [Axe-core](#) Accessibility Validation |  |

<style>

.post .post-content img {

display: inline-block;

margin: 0.5em 0;

}

```
</style>
```

```
<div> - All features of the component meet the requirement.</div>
```

```
<div> - Some features of the component do not meet the requirement.</div>
```

```
<div> - The component does not meet the requirement.</div>
```

#### WAI-ARIA attributes

The Chart component followed the [WAI-ARIA](#) patterns to meet the accessibility. The following ARIA attributes are used in the Chart component:

- `img` (role)
- `button` (role)
- `region` (role)
- `aria-label` (attribute)
- `aria-hidden` (attribute)
- `aria-pressed` (attribute)

#### Keyboard interaction

The Chart component followed the [keyboard interaction](#) guideline, making it easy for people who use assistive technologies (AT) and those who completely rely on keyboard navigation. The following keyboard shortcuts are supported by the Chart component.

| **Press** | **To do this** |

| --- | --- |

| **Alt + J** | Moves the focus to the chart element. |

| **Tab** | Moves the focus to the next element in the chart. |

| **Shift + Tab** | Moves the focus to the previous element in the chart. |

| **Down Arrow** | Moves the focus to the data point left side from the selected point. |

| **Up Arrow** | Moves the focus to the data point right side from the selected point. |

| **Left Arrow** | Moves the focus to the next series in the chart. |

| **Right Arrow** | Moves the focus to the previous series in the chart. |

| **ESC** | Cancel the tooltip for the data point. |

| **Enter/Space** | Selects the data point in the series. |

| **Down/Left Arrow** | Moves the focus to the legend left side from the selected legend. |

| **Up/Right Arrow** | Moves the focus to the legend right side from the selected legend. |

| **Enter/Space** | Toggles the visibility of the corresponding series. |

| **Ctrl + +** | Zoom in the chart. |

| Ctrl + - | Zoom out the chart. |

| Down/Up Arrow | Pan the chart vertically. |

| Left/Right Arrow | Pan the chart horizontally. |

| R | Reset the zoomed chart. |

| Ctrl + P | Prints the Chart. |

### Ensuring accessibility

The Chart component's accessibility levels are ensured through an [accessibility-checker](#) and [axe-core](#) software tools during automated testing.

The accessibility compliance of the Chart component is shown in the following sample. Open the [sample](#) in a new window to evaluate the accessibility of the Chart component with accessibility tools.

```
{% previewsample "https://ej2.syncfusion.com/accessibility/chart.html" %}
```

See also

- [Accessibility in Syncfusion Vue components](#)

### Internationalization in Vue Chart component

Chart provide supports for internationalization for below chart elements.

- Datalabel.
- Axis label.
- Tooltip.

For more information about number and date formatter you can refer [internationalization](#).

<!-- markdownlint-disable MD036 -->

### Globalization

Globalization is the process of designing and developing an component that works in different cultures/locales. Internationalization library is used to globalize number, date, time values in Chart component using `labelFormat` property in axis.

### Numeric Format

In the below example axis, point and tooltip labels are globalized to EUR.

### APP.VUE

```
<template>
  <div id="app">
    <ejs-chart id="container" :title='title'
:primaryXAxis='primaryXAxis' :primaryYAxis='primaryYAxis'
:zoomSettings='zoom' :tooltip='tooltip'>
      <e-series-collection>
        <e-series :dataSource='seriesData' type='Column' xName='x'
yName='y' name='Product X' :marker='marker'> </e-series>
        <e-series :dataSource='seriesData' type='Column' xName='x'
yName='y1' name='Product Y' :marker='marker'> </e-series>
      </e-series-collection>
    </div>
  </template>
```

```

    </ejs-chart>
  </div>
</template>
<script>
import Vue from "vue";
import { ChartPlugin, ColumnSeries, Category, Tooltip, Zoom, DataLabel }
from "@syncfusion/ej2-vue-charts";
import { loadCldr, L10n, setCulture, setCurrencyCode }
from '@syncfusion/ej2-base';
setCurrencyCode('EUR');
Vue.use(ChartPlugin);
export default {
  data() {
    return {
      seriesData: [
        { x: 1900, y: 4, y1: 2.6, y2: 2.8 }, { x: 1920, y: 3.0, y1:
2.8, y2: 2.5 },
        { x: 1940, y: 3.8, y1: 2.6, y2: 2.8 }, { x: 1960, y: 3.4,
y1: 3, y2: 3.2 },
        { x: 1980, y: 3.2, y1: 3.6, y2: 2.9 }, { x: 2000, y: 3.9,
y1: 3, y2: 2 }
      ],
      primaryXAxis: {
        title: 'Year',
        edgeLabelPlacement: 'Shift'
      },
      primaryYAxis: {
        title: 'Sales Amount in Millions',
        labelFormat: 'c'
      },
      zoom: {
        enableMouseWheelZooming: true,
        enableDeferredZooming: true,
        enablePinchZooming: true,
        enableSelectionZooming: true
      },
      marker: {
        dataLabel: {
          visible: true
        }
      },
      tooltip: {
        enable: true, format: '${point.x} : ${point.y}'
      },
      title: "Average Sales Comparison"
    };
  },
  provide: {
    chart: [ColumnSeries, Category, Tooltip, Zoom, DataLabel]
  }
};
</script>
<style>
  #container {
    height: 350px;
  }
</style>

```

```
{% previewsample "page.domainurl/code-snippet/chart/number-format-cs1" %}
```

### Localization in Vue Chart component

Localization library allows to localize the default text content of Chart. In Chart component,

it has the static text on some features(like zooming toolbars)

and this can be changed to any other culture(Arabic, Deutsch, French, etc) by defining the locale value and translation object.

<!-- markdownlint-disable MD033 -->

Locale key words	Text to display
Zoom	Zoom
ZoomIn	ZoomIn
ZoomOut	ZoomOut
Reset	Reset
Pan	Pan
ResetZoom	Reset Zoom

To load translation object in an application use load function of L10n class.

For more information about localization, refer this

[localization](#)

### APP.VUE

```
<template>
  <div id="app">
    <ejs-chart id="container" :locale='locale' :title='title'
:primaryXAxis='primaryXAxis' :zoomSettings='zoom'>
      <e-series-collection>
        <e-series :dataSource='seriesData' type='Area' xName='x'
yName='y' name='Product X'> </e-series>
        <e-series :dataSource='seriesData' type='Area' xName='x'
yName='y1' name='Product Y'> </e-series>
      </e-series-collection>
    </ejs-chart>
  </div>
</template>
<script>
import Vue from "vue";
import { ChartPlugin, AreaSeries, Category, Zoom } from "@syncfusion/ej2-
vue-charts";
import { L10n } from '@syncfusion/ej2-base';
Vue.use(ChartPlugin);
L10n.load({
  'ar-AR': {
    'chart': {
```

```

        ZoomIn: 'تكبير',
        ZoomOut: 'تصغير',
        Zoom: 'زوم',
        Pan: 'مقللة',
        Reset: 'إعادة تعيين',
    },
  },
});
export default {
  data() {
    return {
      seriesData: [
        { x: 1900, y: 4, y1: 2.6, y2: 2.8 }, { x: 1920, y: 3.0, y1: 2.8, y2: 2.5 },
        { x: 1940, y: 3.8, y1: 2.6, y2: 2.8 }, { x: 1960, y: 3.4, y1: 3, y2: 3.2 },
        { x: 1980, y: 3.2, y1: 3.6, y2: 2.9 }, { x: 2000, y: 3.9, y1: 3, y2: 2 }
      ],
      primaryXAxis: {
        title: 'Year',
        edgeLabelPlacement: 'Shift'
      },
      zoom: {
        enableMouseWheelZooming: true,
        enableDeferredZooming: true,
        enablePinchZooming: true,
        enableSelectionZooming: true
      },
      title: "Average Sales Comparison",
      locale: "ar-AR"
    };
  },
  provide: {
    chart: [AreaSeries, Category, Zoom]
  },
};
</script>
<style>
  #container {
    height: 350px;
  }
</style>

```

```
{% previewsample "page.domainurl/code-snippet/chart/user-interaction/zoom-cs1" %}
```

## How To

```
<!-- markdownlint-disable MD036 -->
```

### Hide tool tip in Vue Chart component

By using the [tooltipRender](#) event, you can cancel the tooltip for unselected series in the chart.

To hide the tooltip value in unselected series, follow the given steps:

#### Step 1:

By using the [tooltipRender](#) event, you can get the series elements in the arguments. By using this argument we can compare whether `seriesElementclasslist` is deselected container or not. If it is true then we cancel the tooltip by setting the value for `args.cancel` as `true`.

### APP.VUE

```
<template>
  <div id="app">
    <ejs-chart id="container" :title='title'
:primaryXAxis='primaryXAxis' :primaryYAxis='primaryYAxis'
selectionMode='Series'
:tooltipRender='tooltipRender' :tooltip='tooltip'>
      <e-series-collection>
        <e-series :dataSource='seriesData' type='Column'
xName='country' yName='gold' name='Gold'> </e-series>
        <e-series :dataSource='seriesData' type='Column'
xName='country' yName='silver' name='Silver'> </e-series>
      </e-series-collection>
    </ejs-chart>
  </div>
</template>
<script>
import Vue from "vue";
import { ChartPlugin, ColumnSeries, Category, Selection, Series, Tooltip }
from "@syncfusion/ej2-vue-charts";
Vue.use(ChartPlugin);
export default {
  data() {
    return {
      seriesData: [
        { country: "USA", gold: 50, silver: 60 },
        { country: "China", gold: 40, silver: 50 },
        { country: "Japan", gold: 70, silver: 80 },
        { country: "Australia", gold: 60, silver: 70 },
        { country: "France", gold: 50, silver: 60 },
        { country: "Germany", gold: 40, silver: 50 },
        { country: "Italy", gold: 40, silver: 50 },
        { country: "Sweden", gold: 30, silver: 70 }
      ],
      primaryXAxis: {
        valueType: 'Category',
        title: 'Countries'
      },
      primaryYAxis: {
        minimum: 0, maximum: 80,
        interval: 20, title: 'Medals'
      },
      title: "Olympic Medals",
      tooltip: { enable: true }
    };
  },
  provide: {
    chart: [ColumnSeries, Category, Selection, Tooltip]
  },
  methods: {
    tooltipRender: function(args) {
```



```

    var series = (args.series);
    if (series.seriesElement.classList[0] === 'container_ej2_deselected')
    {
        args.cancel = true;
    }
}
};
</script>
<style>
#container {
    height: 350px;
}
</style>

```

{% previewsample "page.domainurl/code-snippet/chart/axis/category-cs30" %}

<!-- markdownlint-disable MD036 -->

### Dotted line in Vue Chart component

By using **annotation**, you can add dotted lines in the chart.

To add dotted lines in the chart, follow the given steps:

#### Step 1:

Initialize the custom elements by using the **annotation** property.

By setting **coordinateUnits** value as **point** in annotation object you can placed dotted lines in the chart based on point x and y values.

### APP.VUE

```

<template>
  <div id="app">
    <ejs-chart id="container" :title='title'
:primaryXAxis='primaryXAxis' :primaryYAxis='primaryYAxis'
:annotations='annotations'>
      <e-series-collection>
        <e-series :dataSource='seriesData' type='Line'
xName='country' yName='gold' name='Gold'> </e-series>
      </e-series-collection>
    </ejs-chart>
  </div>
</template>
<script>
import Vue from "vue";
import { ChartPlugin, LineSeries, Category, Selection, ChartAnnotation }
from "@syncfusion/ej2-vue-charts";
Vue.use(ChartPlugin);
export default {
  data() {
    return {
      seriesData: [
        { country: "USA", gold: 50, silver: 60 },
        { country: "China", gold: 40, silver: 50 },
        { country: "Japan", gold: 70, silver: 80 },

```

```

        { country: "Australia", gold: 60, silver: 70 },
        { country: "France", gold: 50, silver: 60 },
        { country: "Germany", gold: 40, silver: 50 },
        { country: "Italy", gold: 40, silver: 50 },
        { country: "Sweden", gold: 30, silver: 70 }
      ],
      primaryXAxis: {
        valueType: 'Category',
        title: 'Countries'
      },
      primaryYAxis:
      {
        minimum: 0, maximum: 80,
        interval: 20, title: 'Medals'
      },
      annotations:[{
        content: '<div style="border-top:3px dashed grey;border-top-width:
2px; width: 1000px"></div>',
        coordinateUnits: 'Point',
        x: 'France',
        y: 50
      }],
      title: "Olympic Medals"
    };
  },
  provide: {
    chart: [LineSeries, Category, Selection, ChartAnnotation]
  },
  methods: {
  }
};
</script>
<style>
  #container {
    height: 350px;
  }
</style>

```

{% previewsample "page.domainurl/code-snippet/chart/axis/category-cs28" %}

<!-- markdownlint-disable MD036 -->

### Dynamic chart in Vue Chart component

By using html button, you can add the chart dynamically when click the button.

To add the chart dynamically through button click, follow the given steps:

#### Step 1:

Initially create the html button.

Then create chart inside of button `onClick` function. Now click the button charts will render based on click count.

The following code sample demonstrates the output.

#### APP.VUE

```

<template>
  <div id="app">
    <ejs-button @click.native="addChart">Add Chart</ejs-button>
  </div>
</template>
<script>
import Vue from 'vue';
import { ButtonPlugin } from '@syncfusion/ej2-vue-buttons';
import { Chart } from '@syncfusion/ej2-charts';
import { ChartPlugin, LineSeries, Category, Selection, Series, Tooltip }
from '@syncfusion/ej2-vue-charts';
Chart.Inject(LineSeries);
Vue.use(ChartPlugin);
Vue.use(ButtonPlugin);
var count = 1;
export default {
  data () {
    return {
      seriesData: [
        { x: 1, y: 21 }, { x: 2, y: 24 },
        { x: 3, y: 36 }, { x: 4, y: 38 },
        { x: 5, y: 54 }, { x: 6, y: 57 },
        { x: 7, y: 70 }
      ],
      title: "Inflation - Consumer Price"
    }
  },
  provide: {
    chart: [LineSeries]
  },
  methods: {
    addChart: function(e) {
      var chartEle = document.createElement('div');
      chartEle.id = 'container' + count;
      document.getElementsByTagName('body')[0].appendChild(chartEle);
      let chart = new Chart({
        series: [{
          type: 'Line', xName: 'x', width: 2, marker: { visible: true },
          yName: 'y', name: 'Germany',
          dataSource: [{ x: 1, y: 21 }, { x: 2, y: 24 }, { x: 3, y: 36 },
            { x: 4, y: 38 }, { x: 5, y: 54 }, { x: 6, y: 57 }, { x:
7, y: 70 } ],
          title: 'Inflation - Consumer Price', tooltip: { enable: true },
height: '400', width: '800'
        }
      ]
    }
  }
}
</script>
<style>
  @import '../node_modules/@syncfusion/ej2-base/styles/material.css';
  @import '../node_modules/@syncfusion/ej2-buttons/styles/material.css';
</style>

```

```
{% previewsample "page.domainurl/code-snippet/chart/axis/category-cs29" %}
```

## CheckBox

### Getting Started with the Vue Checkbox Component in Vue 2

This article provides a step-by-step guide for setting up a Vue 2 project using [Vue-CLI](#) and integrating the Syncfusion Vue Checkbox component using the [Composition API](#) / [Options API](#).

#### Prerequisites

[System requirements for Syncfusion Vue UI components](#)

#### Dependencies

The list of dependencies required to use the Checkbox component in your application is given below:

```
`js
|-- @syncfusion/ej2-vue-buttons
|-- @syncfusion/ej2-base
|-- @syncfusion/ej2-buttons
|-- @syncfusion/ej2-vue-base
`,`
```

#### Setting up the Vue 2 project

To generate a Vue 2 project using Vue-CLI, use the [vue create](#) command. Follow these steps to install Vue CLI and create a new project:

```
`bash
npm install -g @vue/cli
vue create quickstart
cd quickstart
npm run serve
`,`

or

`bash
yarn global add @vue/cli
vue create quickstart
cd quickstart
yarn run serve
`,`
```

When creating a new project, choose the option **Default ([Vue 2] babel, eslint)** from the menu.

```
E:\samples>vue create quickstart

Vue CLI v5.0.8
? Please pick a preset:
  Default ([Vue 3] babel, eslint)
> Default ([Vue 2] babel, eslint)
  Manually select features
```

Once the `quickstart` project is set up with default settings, proceed to add Syncfusion components to the project.

### Add Syncfusion Vue packages

Syncfusion packages are available at [npmjs.com](https://www.npmjs.com). To use Vue components, install the required npm package.

This article uses the [Vue Checkbox component](#) as an example. Install the `@syncfusion/ej2-vue-buttons` package by running the following command:

```
`bash
```

```
npm install @syncfusion/ej2-vue-buttons --save
```

```
,
```

or

```
`bash
```

```
yarn add @syncfusion/ej2-vue-buttons
```

```
,
```

### Import Syncfusion CSS styles

You can import themes for the Syncfusion Vue component in various ways, such as using CSS or SASS styles from npm packages, CDN, [CRG](#) and [Theme Studio](#). Refer to [themes topic](#) to know more about built-in themes and different ways to refer to themes in a Vue project.

In this article, the `Material` theme is applied using CSS styles, which are available in installed packages. The necessary `Material` CSS styles for the Checkbox component and its dependents were imported into the `<style>` section of `src/App.vue` file.

### ~/SRC/APP.VUE

```
<style>
@import '../node_modules/@syncfusion/ej2-base/styles/material.css';
@import '../node_modules/@syncfusion/ej2-buttons/styles/material.css';
</style>
```

### Add Syncfusion Vue component

Follow the below steps to add the Vue Checkbox component using `Composition API` or `Options API`:

1\ First, import and register the Checkbox component in the `script` section of the `src/App.vue` file. If you are using the `Composition API`, you should add the `setup` attribute to the `script` tag to indicate that Vue will be using the `Composition API`.

#### **COMPOSITION API (~SRC/APP.VUE)**

```
<script setup>
import { CheckBoxComponent as EjsCheckbox } from "@syncfusion/ej2-vue-
buttons";
</script>
```

#### **OPTIONS API (~SRC/APP.VUE)**

```
<script>
import { CheckBoxComponent } from "@syncfusion/ej2-vue-buttons";
export default {
  components: {
    'ejs-checkbox': CheckBoxComponent
  }
}
</script>
```

2\ In the `template` section define the Checkbox component with the [label](#)

#### **~/SRC/APP.VUE**

```
<template>
<ejs-checkbox label='Default'></ejs-checkbox>
</template>
```

Here is the summarized code for the above steps in the `src/App.vue` file:

#### **COMPOSITION API (~SRC/APP.VUE)**

```
<template>
<ejs-checkbox label='Default'></ejs-checkbox>
</template>
<script setup>
import { CheckBoxComponent as EjsCheckbox } from "@syncfusion/ej2-vue-
buttons";
import { enableRipple } from '@syncfusion/ej2-base';
enableRipple(true);
</script>
<style>
  @import '../node_modules/@syncfusion/ej2-base/styles/material.css';
  @import '../node_modules/@syncfusion/ej2-buttons/styles/material.css';
  .e-checkbox-wrapper {
    margin-top: 18px;
  }
</style>
```

#### **OPTIONS API (~SRC/APP.VUE)**

```
<template>
<ejs-checkbox label='Default'></ejs-checkbox>
```

```

</template>
<script>
import { CheckBoxComponent } from "@syncfusion/ej2-vue-buttons";
import { enableRipple } from '@syncfusion/ej2-base';
enableRipple(true);
export default {
  components: {
    'ejs-checkbox': CheckBoxComponent
  }
}
</script>
<style>
  @import '../node_modules/@syncfusion/ej2-base/styles/material.css';
  @import '../node_modules/@syncfusion/ej2-buttons/styles/material.css';
  .e-checkbox-wrapper {
    margin-top: 18px;
  }
</style>

```

### Run the project

To run the project, use the following command:

```
`bash
```

```
npm run serve
```

```
,
```

or

```
`bash
```

```
yarn run serve
```

```
,
```

```
{% previewsample "page.domainurl/code-snippet/check-box/default-cs2" %}
```

### Change the Checkbox state

The Essential JS 2 Checkbox contains 3 different states visually, they are:

- Checked
- Unchecked
- Indeterminate

The Checkbox [checked](#) property is used to handle the checked and unchecked state. In checked state a tick mark will be added to the visualization of Checkbox.

#### Indeterminate

The Checkbox indeterminate state can be set through [indeterminate](#) property. Checkbox indeterminate state masks the real value of Checkbox visually. The Checkbox cannot be changed to indeterminate state through the user interface, this state can be achieved only through the property.

### COMPOSITION API (~SRC/APP.VUE)

```

<template>
<ul>

```

```
<li><ejs-checkbox label='Checked State' checked=true></ejs-checkbox></li>
<li><ejs-checkbox label='Unchecked State'></ejs-checkbox></li>
<li><ejs-checkbox label='Indeterminate State' indeterminate=true></ejs-
checkbox></li>
</ul>
</template>
<script setup>
import { CheckBoxComponent as EjsCheckbox } from "@syncfusion/ej2-vue-
buttons";
import { enableRipple } from '@syncfusion/ej2-base';
enableRipple(true);
</script>
<style>
  @import '../node_modules/@syncfusion/ej2-base/styles/material.css';
  @import '../node_modules/@syncfusion/ej2-buttons/styles/material.css';
  .e-checkbox-wrapper {
    margin-top: 18px;
  }
  li {
    list-style: none;
  }
</style>
```

#### **OPTIONS API (~SRC/APP.VUE)**

```
<template>
<ul>
<li><ejs-checkbox label='Checked State' checked=true></ejs-checkbox></li>
<li><ejs-checkbox label='Unchecked State'></ejs-checkbox></li>
<li><ejs-checkbox label='Indeterminate State' indeterminate=true></ejs-
checkbox></li>
</ul>
</template>
<script>
import { CheckBoxComponent } from "@syncfusion/ej2-vue-buttons";
import { enableRipple } from '@syncfusion/ej2-base';
enableRipple(true);
export default {
  components: {
    'ejs-checkbox': CheckBoxComponent
  }
}
</script>
<style>
  @import '../node_modules/@syncfusion/ej2-base/styles/material.css';
  @import '../node_modules/@syncfusion/ej2-buttons/styles/material.css';
  .e-checkbox-wrapper {
    margin-top: 18px;
  }
  li {
    list-style: none;
  }
</style>
```

{% previewsample "page.domainurl/code-snippet/check-box/default-cs3" %}



## Getting Started with the Vue CheckBox Component in Vue 3

This article provides a step-by-step guide for setting up a [Vite](#) project with a JavaScript environment and integrating the Syncfusion Vue CheckBox component using the [Composition API](#) / [Options API](#).

The **Composition API** is a new feature introduced in Vue.js 3 that provides an alternative way to organize and reuse component logic. It allows developers to write components as functions that use smaller, reusable functions called composition functions to manage their properties and behavior.

The **Options API** is the traditional way of writing Vue.js components, where the component logic is organized into a series of options that define the component's properties and behavior. These options include data, methods, computed properties, watchers, lifecycle hooks, and more.

### Prerequisites

#### [System requirements for Syncfusion Vue UI components](#)

#### Set up the Vite project

A recommended approach for beginning with Vue is to scaffold a project using [Vite](#). To create a new Vite project, use one of the commands that are specific to either NPM or Yarn.

```
`bash
```

```
npm create vite@latest
```

```
`
```

or

```
`bash
```

```
yarn create vite
```

```
`
```

Using one of the above commands will lead you to set up additional configurations for the project as below:

1. Define the project name: We can specify the name of the project directly. Let's specify the name of the project as **my-project** for this article.

```
`bash
```

```
? Project name: » my-project
```

```
`
```

2. Select **Vue** as the framework. It will create a Vue 3 project.

```
`bash
```

```
? Select a framework: » - Use arrow-keys. Return to submit.
```

```
Vanilla
```

```
Vue
```

```
React
```

```
Preact
```

```
Lit
```

Svelte

Others

,

3. Choose **JavaScript** as the framework variant to build this Vite project using JavaScript and Vue.

```
`bash
```

? Select a variant: » - Use arrow-keys. Return to submit.

JavaScript

TypeScript

Customize with create-vue ↗

Nuxt ↗

,

4. Upon completing the aforementioned steps to create the **my-project**, run the following command to install its dependencies:

```
`bash
```

```
cd my-project
```

```
npm install
```

,

or

```
`bash
```

```
cd my-project
```

```
yarn install
```

,

Now that **my-project** is ready to run with default settings, let's add Syncfusion components to the project.

#### Add Syncfusion Vue packages

Syncfusion Vue component packages are available at [npmjs.com](https://www.npmjs.com). To use Syncfusion Vue components in the project, install the corresponding npm package.

This article uses the [Vue CheckBox component](#) as an example. To use the Vue Button component in the project, the **@syncfusion/ej2-vue-buttons** package needs to be installed using the following command:

```
`bash
```

```
npm install @syncfusion/ej2-vue-buttons --save
```

,

or

```
`bash
```

```
yarn add @syncfusion/ej2-vue-buttons
```

### Import Syncfusion CSS styles

You can import themes for the Syncfusion Vue component in various ways, such as using CSS or SASS styles from npm packages, CDN, [CRG](#) and [Theme Studio](#). Refer to [themes topic](#) to know more about built-in themes and different ways to refer to themes in a Vue project.

In this article, **Material** theme is applied using CSS styles, which are available in installed packages. The necessary **Material** CSS styles for the CheckBox component and its dependents were imported into the `<style>` section of **src/App.vue** file.

#### ~/SRC/APP.VUE

```
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-buttons/styles/material.css";
</style>
```

The order of importing CSS styles should be in line with its dependency graph.

### Add Syncfusion Vue component

Follow the below steps to add the Vue CheckBox component using **Composition API** or **Options API**:

1.First, import and register the CheckBox component and its child directives in the **script** section of the **src/App.vue** file. If you are using the **Composition API**, you should add the **setup** attribute to the **script** tag to indicate that Vue will be using the **Composition API**.

#### COMPOSITION API (~/SRC/APP.VUE)

```
<script setup>
import { CheckBoxComponent as EjsCheckBox } from "@syncfusion/ej2-vue-
buttons";
</script>
```

#### OPTIONS API (~/SRC/APP.VUE)

```
<script>
import { CheckBoxComponent } from "@syncfusion/ej2-vue-buttons";
//Component registration
export default {
  name: "App",
  components: {
    "ejs-checkbox": CheckBoxComponent
  }
}
</script>
```

2.In the **template** section, define the CheckBox component with the **label** property.

#### ~/SRC/APP.VUE

```
<template>
<ejs-checkbox label='Default'></ejs-checkbox>
```

```
</template>
```

3. Declare the label property in the script section.

### COMPOSITION API (~ /SRC/APP.VUE)

```
<template>
<ejs-checkbox label='Default'></ejs-checkbox>
</template>
<script setup>
import { CheckBoxComponent as EjsCheckBox } from "@syncfusion/ej2-vue-buttons";
</script>
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-buttons/styles/material.css";
</style>
```

### OPTIONS API (~ /SRC/APP.VUE)

```
<template>
<ejs-checkbox label='Default'></ejs-checkbox>
</template>
<script>
import { CheckBoxComponent } from "@syncfusion/ej2-vue-buttons";
//Component registration
export default {
name: "App",
components: {
"ejs-checkbox": CheckBoxComponent
}
}
</script>
```

### Run the project

To run the project, use the following command:

```
`bash
```

```
npm run dev
```

```
,
```

or

```
`bash
```

```
yarn run dev
```

```
,
```

The output will appear as follows:

☒ Default

**Sample:** [vue-3-checkbox-getting-started](#).

For migrating from Vue 2 to Vue 3, refer to the [migration](#) documentation.

See also

- [Getting Started with Vue UI Components using Composition API and TypeScript](#)
- [Getting Started with Vue UI Components using Options API and TypeScript](#)

## Label and size in Vue Check box component

This section explains the different sizes and labels.

### Label

The CheckBox caption can be defined by using the [label](#) property. This reduces the manual addition of label for CheckBox. You can customize the label position before or after the CheckBox through the [labelPosition](#) property.

#### APP.VUE

```
<template>
<ul>
<li><ejs-checkbox label='Left Side Label' labelPosition='Before'></ejs-
checkbox></li>
<li><ejs-checkbox label='Right Side Label' checked=true></ejs-checkbox></li>
</ul>
</template>
<script>
import Vue from 'vue';
import { CheckBoxPlugin } from '@syncfusion/ej2-vue-buttons';
import { enableRipple } from '@syncfusion/ej2-base';
enableRipple(true);
Vue.use(CheckBoxPlugin);
export default {}
</script>
<style>
  @import '../node_modules/@syncfusion/ej2-base/styles/material.css';
  @import '../node_modules/@syncfusion/ej2-buttons/styles/material.css';
  .e-checkbox-wrapper {
    margin-top: 18px;
  }
  li {
    list-style: none;
  }
</style>
```

{% previewsample "page.domainurl/code-snippet/check-box/default-cs9" %}

### Size

The different CheckBox sizes available are default and small. To reduce the size of default CheckBox to small, set the [cssClass](#) property to `e-small`.

#### APP.VUE

```
<template>
<ul>
<li><ejs-checkbox label='Small' cssClass='e-small'></ejs-checkbox></li>
<li><ejs-checkbox label='Default'></ejs-checkbox></li>
```

```

</ul>
</template>
<script>
import Vue from 'vue';
import { CheckBoxPlugin } from '@syncfusion/ej2-vue-buttons';
import { enableRipple } from '@syncfusion/ej2-base';
enableRipple(true);
Vue.use(CheckBoxPlugin);
export default {}
</script>
<style>
  @import '../node_modules/@syncfusion/ej2-base/styles/material.css';
  @import '../node_modules/@syncfusion/ej2-buttons/styles/material.css';
  .e-checkbox-wrapper {
    margin-top: 18px;
  }
  li {
    list-style: none;
  }
</style>

```

{% previewsample "page.domainurl/code-snippet/check-box/default-cs10" %}

See Also

- [CheckBox customization](#)

## Accessibility in Vue Check box component

The Check box component followed the accessibility guidelines and standards, including [ADA](#), [Section 508](#), [WCAG 2.2](#) standards, and [WCAG roles](#) that are commonly used to evaluate accessibility.

The accessibility compliance for the Check box component is outlined below.

| Accessibility Criteria | Compatibility |

| -- | -- |

| [WCAG 2.2](#) Support |  |

| [Section 508](#) Support |  |

| Screen Reader Support |  |

| Right-To-Left Support |  |

| Color Contrast |  |

| Mobile Device Support |  |

| Keyboard Navigation Support |  |

| [Accessibility Checker](#) Validation |  |

| [Axe-core](#) Accessibility Validation |  |

<style>

.post .post-content img {

display: inline-block;

margin: 0.5em 0;

}

</style>

<div> - All features of the component meet the requirement.</div>

<div> - Some features of the component do not meet the requirement.</div>

<div> - The component does not meet the requirement.</div>

### WAI-ARIA attributes

The Check box component followed the [WAI-ARIA](https://www.w3.org/WAI/ARIA/apg/patterns/Check box/) patterns to meet the accessibility. The following ARIA attributes are used in the Check box component:

| Attributes | Purpose |

| --- | --- |

| **aria-disabled** | Indicates that the element is perceivable but disabled, so it is not editable or otherwise operable. |

### Keyboard interaction

The Check box component followed the [keyboard interaction](https://www.w3.org/WAI/ARIA/apg/patterns/Check box/#keyboardinteraction) guideline, making it easy for people who use assistive technologies (AT) and those who completely rely on keyboard navigation. The following keyboard shortcuts are supported by the Check box component.

| **Press** | **To do this** |

| --- | --- |

| **Space** | When the Check box has focus, pressing the Space key changes the state of the Check box. |

### Ensuring accessibility

The Check box component's accessibility levels are ensured through an [accessibility-checker](#) and [axe-core](#) software tools during automated testing.

The accessibility compliance of the Check box component is shown in the following sample. Open the [sample](https://ej2.syncfusion.com/accessibility/Check box.html) in a new window to evaluate the accessibility of the Check box component with accessibility tools.

{% previewsample "page.domainurl/code-snippet/check-box/default-cs1" %}

See also

- [Accessibility in Syncfusion Vue components](#)

### Two way binding in Vue Check box component

It can be achieved by using the `v-model` directive in Vue. In the following sample, when you check or uncheck a value in one CheckBox will automatically check or unchecks the value in the other CheckBox. It updates the other CheckBox using `value` property.

#### APP.VUE

```
<template>
<div>
<div id="wrapper1">
<ejs-checkbox label='Default' v-model="value" :checked="value"></ejs-
checkbox>
</div>
<div id="wrapper2">
<ejs-checkbox label='Default' v-model="value" :checked="value"></ejs-
checkbox>
</div>
</div>
</template>
<script>
import Vue from 'vue';
import { CheckBoxPlugin } from "@syncfusion/ej2-vue-buttons";
import { enableRipple } from '@syncfusion/ej2-base';
enableRipple(true);
Vue.use(CheckBoxPlugin);
export default {
  data() {
    return {
      value: true
    }
  }
}
</script>
<style>
@import '../node_modules/@syncfusion/ej2-base/styles/material.css';
@import '../node_modules/@syncfusion/ej2-buttons/styles/material.css';
.e-checkbox-wrapper {
  margin-top: 18px;
}
#wrapper1 {
  float: left;
  padding: 0 0 0 80px;
}
#wrapper2 {
  float: right;
```



```
padding: 0 460px 0 0;
}
</style>
```

{% previewsample "page.domainurl/code-snippet/radio-button/default-cs1" %}

### Style and appearance in Vue Check box component

To modify the CheckBox appearance, you need to override the default CSS of CheckBox component. Please find the list of CSS classes and its corresponding section in CheckBox. Also, you have an option to create your own custom theme for the controls using our [Theme Studio](#).

#### CSS Class | Purpose of Class

- |.e-checkbox-wrapper .e-frame|To customize the checkbox frame.
- |.e-checkbox-wrapper:hover .e-frame|To customize the checkbox frame on hover.
- |.e-checkbox-wrapper .e-label|To customize the checkbox label.
- |.e-checkbox-wrapper:hover .e-label|To customize the checkbox label on hover.
- |.e-checkbox-wrapper .e-frame.e-check|To customize the checked checkbox.
- |.e-checkbox-wrapper:hover .e-frame.e-check|To customize the checked checkbox when hover

### How To

#### Customized checkbox in Vue Check box component

##### *Customize CheckBox Appearance*

You can customize the appearance of the CheckBox component using the CSS rules. Define own CSS rules according to your requirement and assign the class name to the [cssClass](#) property.

The background and border color of the CheckBox is customized through the custom classes to create primary, success, warning, and danger info type of checkbox.

#### **APP.VUE**

```
<template>
<ul>
<li><ejs-checkbox label='Primary' cssClass='e-primary' checked=true></ejs-
checkbox></li>
<li><ejs-checkbox label='Success' cssClass='e-success' checked=true></ejs-
checkbox></li>
<li><ejs-checkbox label='Info' cssClass='e-info' checked=true></ejs-
checkbox></li>
<li><ejs-checkbox label='Warning' cssClass='e-warning' checked=true></ejs-
checkbox></li>
<li><ejs-checkbox label='Danger' cssClass='e-danger' checked=true></ejs-
checkbox></li>
</ul>
</template>
<script>
import Vue from 'vue';
import { CheckBoxPlugin } from '@syncfusion/ej2-vue-buttons';
Vue.use(CheckBoxPlugin);
export default {}
</script>
<style>
```

```

@import '../node_modules/@syncfusion/ej2-base/styles/material.css';
@import '../node_modules/@syncfusion/ej2-buttons/styles/material.css';
li {
    list-style: none;
}
.e-checkbox-wrapper {
    margin-top: 18px;
}
.e-checkbox-wrapper.e-primary:hover .e-frame.e-check { /* csslint allow:
adjoining-classes */
    background-color: #e03872;
}
.e-checkbox-wrapper.e-success .e-frame.e-check,
.e-checkbox-wrapper.e-success .e-checkbox:focus + .e-frame.e-check { /*
csslint allow: adjoining-classes */
    background-color: #689f38;
}
.e-checkbox-wrapper.e-success:hover .e-frame.e-check { /* csslint allow:
adjoining-classes */
    background-color: #449d44;
}
.e-checkbox-wrapper.e-info .e-frame.e-check,
.e-checkbox-wrapper.e-info .e-checkbox:focus + .e-frame.e-check { /* csslint
allow: adjoining-classes */
    background-color: #2196f3;
}
.e-checkbox-wrapper.e-info:hover .e-frame.e-check { /* csslint allow:
adjoining-classes */
    background-color: #0b7dda;
}
.e-checkbox-wrapper.e-warning .e-frame.e-check,
.e-checkbox-wrapper.e-warning .e-checkbox:focus + .e-frame.e-check { /*
csslint allow: adjoining-classes */
    background-color: #ef6c00;
}
.e-checkbox-wrapper.e-warning:hover .e-frame.e-check { /* csslint allow:
adjoining-classes */
    background-color: #cc5c00;
}
.e-checkbox-wrapper.e-danger .e-frame.e-check,
.e-checkbox-wrapper.e-danger .e-checkbox:focus + .e-frame.e-check { /*
csslint allow: adjoining-classes */
    background-color: #d84315;
}
.e-checkbox-wrapper.e-danger:hover .e-frame.e-check { /* csslint allow:
adjoining-classes */
    background-color: #ba3912;
}
</style>

```

{% previewsample "page.domainurl/code-snippet/check-box/default-cs4" %}

#### Custom Frame

CheckBox frame can be customized as per the requirement by adding CSS rules.

In the following example, to-do list is displayed with round checkbox by changing **border-radius** as **100%** by adding **e-custom** class.

### APP.VUE

```
<template>
<ul>
<li><ejs-checkbox label='Buy Groceries' cssClass='e-custom'
checked=true></ejs-checkbox></li>
<li><ejs-checkbox label='Pay Rent' cssClass='e-custom'></ejs-checkbox></li>
<li><ejs-checkbox label='Make Dinner' cssClass='e-custom'></ejs-
checkbox></li>
<li><ejs-checkbox label='Finish To-do List Article' cssClass='e-
custom'></ejs-checkbox></li>
</ul>
</template>
<script>
import Vue from 'vue';
import { CheckBoxPlugin } from '@syncfusion/ej2-vue-buttons';
import { enableRipple } from '@syncfusion/ej2-base';
enableRipple(true);
Vue.use(CheckBoxPlugin);
export default {}
</script>
<style>
@import '../node_modules/@syncfusion/ej2-base/styles/material.css';
@import '../node_modules/@syncfusion/ej2-buttons/styles/material.css';
li {
list-style: none;
}
.e-checkbox-wrapper {
margin-top: 18px;
}
.e-custom .e-frame {
border-radius: 100%;
}
.e-checkicon.e-checkbox-wrapper .e-frame.e-check::before {
content: '\e77d';
}
.e-checkicon.e-checkbox-wrapper .e-check {
font-size: 8.5px;
}
.e-checkicon.e-checkbox-wrapper .e-frame.e-check {
background-color: white;
border-color: grey;
color: grey;
}
.e-checkicon.e-checkbox-wrapper:hover .e-frame.e-check {
background-color: white;
border-color: grey;
color: grey;
}
.e-checkicon.e-checkbox-wrapper .e-checkbox:focus + .e-frame.e-check {
background-color: white;
border-color: grey;
box-shadow: none;
color: grey;
}
```

```
}
</style>
```

{% previewsample "page.domainurl/code-snippet/check-box/default-cs5" %}

### Custom Check Icon

CheckBox check icon can be customized as per the requirement by adding CSS rules.

In the following example, the check icon can be customized by changing check icon content, background and border color in focus and hovered states by adding `e-checkicon` class.

### APP.VUE

```
<template>
<ul>
<li><ejs-checkbox label='Buy Groceries' cssClass='e-checkicon'
checked=true></ejs-checkbox></li>
<li><ejs-checkbox label='Pay Rent' cssClass='e-checkicon'></ejs-
checkbox></li>
<li><ejs-checkbox label='Make Dinner' cssClass='e-checkicon'></ejs-
checkbox></li>
<li><ejs-checkbox label='Finish To-do List Article' cssClass='e-
checkicon'></ejs-checkbox></li>
</ul>
</template>
<script>
import Vue from 'vue';
import { CheckBoxPlugin } from '@syncfusion/ej2-vue-buttons';
import { enableRipple } from '@syncfusion/ej2-base';
enableRipple(true);
Vue.use(CheckBoxPlugin);
export default {}
</script>
<style>
@import '../node_modules/@syncfusion/ej2-base/styles/material.css';
@import '../node_modules/@syncfusion/ej2-buttons/styles/material.css';
li {
list-style: none;
}
@font-face {
font-family: 'btn-icon';
src:
url(data:application/x-font-ttf;charset=utf-
8;base64,AAEAAAKAIIAAwAgTlMvMjltSfgAAAEoAAAVmNtYXNlH+dzAAABoAAAAEJnbHlm1v4
8pAAAAfgAAQYAGVhZBOPfZcAAADQAAANmhoZWEIUQQJAAArAAAACRobXR4IAAAAAAYAAAAA
gbG9jYQYQ6ApQAAAHkAAAAEm1heHABFQCqAAABCAAAACBuYW1l071lFxAAABhAAAAIxCG9zdK9uovo
AAAhEAAAAGABAAAAEAAAAAFwEAAAAAAD9AABAAAAAACAABAAAAQAAJ1LUzF8
PPPUACwQAAAAAANG+nFMAAAAA2D6cUwAAAAAD9AP0AAAACAACAAAAAEEAAAAIAJ4AAwAAAAA
AAgAAAAoACgAAAP8AAAAAQAQAAZAABQAAOkCzAAAAI8CiQLMAAAB6wAyAQgAAAIABQMAAAA
AAAAAAAAAAAAAAAAAUGZFZABA5wDnBgQAAAAAXAQAAAAAABAAAAAABAAAAAQAQA
EAAAABAAAAQAQAAEAAAABAAAAQAQAAAAAACAQAAAwAAABQAQAAwABAAAAFAAEAC4AAAAEAAQAQA
A5wb//wAA5wD//wAAAAEABAAAAEAAgADAAQABQAGAAcAAAAAAAAAADgAkADIAhAEuAewCDAAAAE
AAAAA2ED9AACAA3CQGeAsT9PAwB9AH0AAACAAAAAPH/AAAwAHAAALIREhASERIQJpAV7+ov3
QAV7+ogwD6PwYA+gAAAEAAAAA4sD9AACAAATARF0AxgCAP4MA+gAAAAABAAAAAP0A/QAQwAAExE
fDyE/DxEvDyEPDgwBAGMFBQcICQkLCwWMDQ4NAtONdG0MDAsLCQkIBwUFAwIBAQIDBQUHCAkJCws
MDA00Df0mDQ4NDawLCwkJCACFBQMCA239Jg4NDQ0LCwsJCQgHBQUDAgEBAGMFBQcICQkLCwsNDQ0
```

```
OAtOoDQ0NCwsLCQkIBwUFAwIBAQIDBUHCAkJCwsLDQ0NAAIAAAAAA/MDxQADAIwAADczESMBDwM
VFw8METM3HwQ3Fz8KPQEvBT8LLwg3NT8INS8FNT8NNS8JByU/BDUvCyMPAQytrQH5AgoEAQEBARG
hERESEyIJCsgQBiEHNQceOZPbDgUICw0LCQUDBAICBAkGAgEBAQMOBAkIBgcDAwEBAQEDAwMJAgE
BAxYLBQQEAWMCAgIEBAoBAQEECgcHBgUFBAMDAQEBAQQFBwkFBQUGef6tDwKEAwIBAQMDCgwVAwc
GDAsNBwdaAYcB3gEFAwN2HwoELDodGxwaLwkIGwz+igEBHwMBAQECAQEDBgoKDAYICAgFCAkICwU
EBAQFAwYDBwgIDAgHCACGBgYFBQkEAgYCBawJBgUGBwkJCgkICAcLBAIFAwIEBAQFBQcGBwgHBgY
GBgoJCAYCAGeBAQFGMRkaGw0NDA0LIh4xBAQCBAEBAgADAAAAAOKA/MAHABCAJ0AAAEzHwIRDwM
hLwIDNzM/CjUTHwcVIwcVIy8HETcXmZ8KNScxBxEfDjsBHQEfDTMhMz8OES8PIz0BLw4hA0EDBQQ
DAQIEBf5eBQQCAW4RDg0LCQgGBQUDBAFEBAMDawIBAQGL7Y0EAWQCAGIBAYYKChEQDQsJCACeBAU
CYt8BAQIDBAUFBQcHBwgICQgKjQECAGMEBAUFBgYHBgcIBwGcCACHBwYGBgUFBQDAGIBAQEBAgI
DBAQFBQYGBgcHBwgMAQMDAwUFBgYHBwgICQkJ/tQCiwMEBf3XAwYEAgIEBgFoAQEDBQYGBwgIBw0
KhQEiAQECAGMDAwTV+94BAQECAwMDBAGYAQECBAYHCAgJCgkQCaQC6/47CQkICQcIBwYGBQQEAWI
CUAgHBwCGBgYFBQQEAWMBAgIBAwMEBAUFBQcGBwCHCAImCACHBwYGBgUFBQDAGIBAdUJCQgICAg
GBwYFBQDAGEBAAAAAIAAAAAA6cD9AADAawAADchNSElAQcJAScBESNZA078sgGB/uMuAXkBgDb
+1EWMTZcBCD3+ngFiPf7pAxMAAAAAABIA3gABAAAAAEEEEAAAAABAAAAAABAAgAAQABAAAAAA
CAACACQABAAAAAADAAGAEAAABAAAAAEEAGAGAABAAAAAFAAsAIAABAAAAAAGAAgAKwABAAA
AAAAKACwAMwABAAAAAALABIAxwADAAEECQAAAAIACQADAAEECQABABAAcWADAAEECQACAA4AgwA
DAAEECQADABAAKQADAAEECQAEABAAoQADAAEECQAFABYAsQADAAEECQAGABAAxwADAAEECQAKAFg
A1wADAAEECQALACQBLyBidG4taWNvblJlZ3VsYXJidG4taWNvbmJ0bilpY29uVmVyc2lvbiAxLjB
idG4taWNvbkbZvbnQgZ2VuZXJhdGVkIHVzaW5nIFN5bmNmdXNpb24gTWV0cm8gU3R1ZGlvd3d3LnN
5bmNmdXNpb24uY29tACAAYgB0AG4ALQBpAGMABwBuAFIAZQBnAHUAbABhAHIAyAgB0AG4ALQBpAGM
ABwBuAGIAAdABuAC0AaQBJAG8AbgBWAGUAcgBzAGkAbwBuACAAMQAuADAAYgB0AG4ALQBpAGMABwB
uAEYABwBuAHQAIAbnAGUAbgBlAHIAyQB0AGUAZAAGAHUAcwBpAG4AZwAgAFMAeQBuaGMAZgB1AHM
AaQBvAG4AIAbnAGUAdABYAG8AIAbTAHQAdQBkAGkAbwB3AHcAdwAuAHMAeQBuaGMAZgB1AHMAaQB
vAG4ALgBjAG8AbQAAAAACAAAAAoooooAAAAAAAAAAAAAAAAAAAAAGBAGEDAQQBBQE
GAQcBCAEJAaptZWRpYS1wbGF5C2l1ZGlhLXBhdXNlDmFycm93aGVhZC1sZWZ0BHN0b3AJbGlrZS0
tLTaxBGNvcHkQLWRvd25sb2FkLTAYLXdmLQAA) format('true');
font-weight: normal;
font-style: normal;
}
.e-icons {
font-family: 'btn-icon' !important;
speak: none;
font-size: 55px;
font-style: normal;
font-weight: normal;
font-variant: normal;
text-transform: none;
line-height: 1;
-webkit-font-smoothing: antialiased;
-moz-osx-font-smoothing: grayscale;
}
.e-checkbox-wrapper {
margin-top: 18px;
}
.e-checkicon.e-checkbox-wrapper .e-frame.e-check::before {
content: '\e703';
}
.e-checkicon.e-checkbox-wrapper .e-check {
font-size: 8px;
}
.e-checkicon.e-checkbox-wrapper .e-frame.e-check {
background-color: white;
border-color: grey;
color: grey;
}
.e-checkicon.e-checkbox-wrapper:hover .e-frame.e-check {
```

```

        background-color: white;
        border-color: grey;
        color: grey;
    }
    .e-checkicon.e-checkbox-wrapper .e-checkbox:focus + .e-frame.e-check {
        background-color: white;
        border-color: grey;
        box-shadow: none;
        color: grey;
    }
    .e-checkicon.e-checkbox-wrapper .e-ripple-element {
        background: grey;
    }
}
</style>

```

{% previewsample "page.domainurl/code-snippet/check-box/default-cs6" %}

### Name and value in form submit in Vue Check box component

The [name](#) attribute of the CheckBox is used to group Checkboxes. When the Checkboxes are grouped in form, the checked items [value](#) attribute will post to the server on form submit that can be retrieved through the name. The disabled and unchecked CheckBox value will not be sent to the server on form submit.

In the following code snippet, Cricket and Hockey are in the [checked](#) state, Tennis is in [disabled](#) state and Basketball is in unchecked state. Now, the value that is in checked state only be sent on form submit.

### APP.VUE

```

<template>
<form>
<ul>
<li><ejs-checkbox name='Sport' value='Cricket' label='Cricket'
checked=true></ejs-checkbox></li>
<li><ejs-checkbox name='Sport' value='Hockey' label='Hockey'
checked=true></ejs-checkbox></li>
<li><ejs-checkbox name='Sport' value='Tennis' label='Tennis'
disabled=true></ejs-checkbox></li>
<li><ejs-checkbox name='Sport' value='Basketball' label='Basketball'></ejs-
checkbox></li>
<li><ejs-button isPrimary=true>Submit</ejs-button></li>
</ul>
</form>
</template>
<script>
import Vue from 'vue';
import { CheckBoxPlugin, ButtonPlugin } from '@syncfusion/ej2-vue-buttons';
import { enableRipple } from '@syncfusion/ej2-base';
enableRipple(true);
Vue.use(CheckBoxPlugin);
Vue.use(ButtonPlugin);
export default {}
</script>
<style>
    @import '../node_modules/@syncfusion/ej2-base/styles/material.css';
    @import '../node_modules/@syncfusion/ej2-buttons/styles/material.css';

```

```
.e-checkbox-wrapper {
  margin-top: 18px;
}
button {
  margin: 20px 0 0 5px;
}
li {
  list-style: none;
}
</style>
```

{% previewsample "page.domainurl/code-snippet/check-box/default-cs7" %}

### Right to left in Vue Check box component

CheckBox component has RTL support. This can be achieved by setting [enableRtl](#) as `true`.

The following example illustrates how to enable right-to-left support in CheckBox component.

#### APP.VUE

```
<template>
<ejs-checkbox label='Default' enableRtl=true></ejs-checkbox>
</template>
<script>
import Vue from 'vue';
import { CheckBoxPlugin } from '@syncfusion/ej2-vue-buttons';
import { enableRipple } from '@syncfusion/ej2-base';
enableRipple(true);
Vue.use(CheckBoxPlugin);
export default {}
</script>
<style>
  @import '../node_modules/@syncfusion/ej2-base/styles/material.css';
  @import '../node_modules/@syncfusion/ej2-buttons/styles/material.css';
  .e-checkbox-wrapper {
    margin-top: 18px;
  }
</style>
```

{% previewsample "page.domainurl/code-snippet/check-box/default-cs8" %}

## Chips

### Getting Started with the Vue Chips Component in Vue 2

This article provides a step-by-step guide for setting up a Vue 2 project using [Vue-CLI](#) and integrating the Syncfusion Vue Chips component using the [Composition API](#) / [Options API](#).

#### Prerequisites

##### [System requirements for Syncfusion Vue UI components](#)

#### Dependencies

The list of dependencies required to use the Chip component in your application is given below :

```
`js
```

```
|-- @syncfusion/ej2-vue-buttons  
|-- @syncfusion/ej2-base  
|-- @syncfusion/ej2-buttons  
|-- @syncfusion/ej2-vue-base  
,
```

### Setting up the Vue 2 project

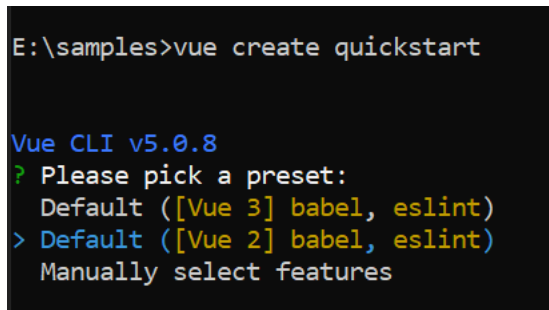
To generate a Vue 2 project using Vue-CLI, use the [vue create](#) command. Follow these steps to install Vue CLI and create a new project:

```
`bash  
npm install -g @vue/cli  
vue create quickstart  
cd quickstart  
npm run serve  
,
```

or

```
`bash  
yarn global add @vue/cli  
vue create quickstart  
cd quickstart  
yarn run serve  
,
```

When creating a new project, choose the option **Default ([Vue 2] babel, eslint)** from the menu.



```
E:\samples>vue create quickstart  
  
Vue CLI v5.0.8  
? Please pick a preset:  
  Default ([Vue 3] babel, eslint)  
> Default ([Vue 2] babel, eslint)  
  Manually select features
```

Once the **quickstart** project is set up with default settings, proceed to add Syncfusion components to the project.

### Add Syncfusion Vue packages

Syncfusion packages are available at [npmjs.com](https://www.syncfusion.com/). To use Vue components, install the required npm package.



This article uses the [Vue Chips component](#) as an example. Install the `@syncfusion/ej2-vue-buttons` package by running the following command:

```
`bash
```

```
npm install @syncfusion/ej2-vue-buttons --save
```

```
,
```

or

```
`bash
```

```
yarn add @syncfusion/ej2-vue-buttons
```

```
,
```

### Import Syncfusion CSS styles

You can import themes for the Syncfusion Vue component in various ways, such as using CSS or SASS styles from npm packages, CDN, [CRG](#) and [Theme Studio](#). Refer to [themes topic](#) to know more about built-in themes and different ways to refer to themes in a Vue project.

In this article, the `Material` theme is applied using CSS styles, which are available in installed packages. The necessary `Material` CSS styles for the Chips component and its dependents were imported into the `<style>` section of `src/App.vue` file.

#### ~/SRC/APP.VUE

```
<style>
@import '../node_modules/@syncfusion/ej2-vue-buttons/styles/material.css';
</style>
```

### Add Syncfusion Vue component

Follow the below steps to add the Vue Chips component using `Composition API` or `Options API`:

1\ First, import and register the Chips component in the `script` section of the `src/App.vue` file. If you are using the `Composition API`, you should add the `setup` attribute to the `script` tag to indicate that Vue will be using the `Composition API`.

#### COMPOSITION API (~/SRC/APP.VUE)

```
<script setup>
import { ChipListComponent as EjsChiplist } from '@syncfusion/ej2-vue-buttons';
</script>
```

#### OPTIONS API (~/SRC/APP.VUE)

```
<script>
import { ChipListComponent } from '@syncfusion/ej2-vue-buttons';
export default {
  components: {
    'ejs-chiplist': ChipListComponent
  }
}
</script>
```

2\ In the `template` section, define the Chips component with the `text` property.

#### ~/SRC/APP.VUE

```
<template>
<ejs-chiplist id="chip" text="Janet Leverling"></ejs-chiplist>
</template>
```

Here is the summarized code for the above steps in the `src/App.vue` file:

#### COMPOSITION API (~/SRC/APP.VUE)

```
<template>
  <ejs-chiplist id="chip" text="Janet Leverling"></ejs-chiplist>
</template>
<script setup>
import { ChipListComponent as EjsChiplist } from '@syncfusion/ej2-vue-
buttons';
</script>
<style>
@import '../node_modules/@syncfusion/ej2-vue-buttons/styles/material.css';
</style>
```

#### OPTIONS API (~/SRC/APP.VUE)

```
<template>
  <ejs-chiplist id="chip" text="Janet Leverling"></ejs-chiplist>
</template>
<script>
import { ChipListComponent } from '@syncfusion/ej2-vue-buttons';
export default {
  components: {
    'ejs-chiplist': ChipListComponent
  }
}
</script>
<style>
@import '../node_modules/@syncfusion/ej2-vue-buttons/styles/material.css';
</style>
```

#### Run the project

To run the project, use the following command:

```
`bash
```

```
npm run serve
```

```
`
```

or

```
`bash
```

```
yarn run serve
```

```
{% previewsample "page.domainurl/code-snippet/chips/default-cs8" %}
```

## Getting Started with the Vue Chips Component in Vue 3

This article provides a step-by-step guide for setting up a [Vite](#) project with a JavaScript environment and integrating the Syncfusion Vue Chips component using the [Composition API](#) / [Options API](#).

The **Composition API** is a new feature introduced in Vue.js 3 that provides an alternative way to organize and reuse component logic. It allows developers to write components as functions that use smaller, reusable functions called composition functions to manage their properties and behavior.

The **Options API** is the traditional way of writing Vue.js components, where the component logic is organized into a series of options that define the component's properties and behavior. These options include data, methods, computed properties, watchers, lifecycle hooks, and more.

### Prerequisites

#### [System requirements for Syncfusion Vue UI components](#)

#### Set up the Vite project

A recommended approach for beginning with Vue is to scaffold a project using [Vite](#). To create a new Vite project, use one of the commands that are specific to either NPM or Yarn.

```
`bash
```

```
npm create vite@latest
```

```
`
```

or

```
`bash
```

```
yarn create vite
```

```
`
```

Using one of the above commands will lead you to set up additional configurations for the project as below:

1. Define the project name: We can specify the name of the project directly. Let's specify the name of the project as **my-project** for this article.

```
`bash
```

```
? Project name: » my-project
```

```
`
```

2. Select **Vue** as the framework. It will create a Vue 3 project.

```
`bash
```

```
? Select a framework: » - Use arrow-keys. Return to submit.
```

```
Vanilla
```

```
Vue
```

```
React
```

Preact

Lit

Svelte

Others

,

3. Choose **JavaScript** as the framework variant to build this Vite project using JavaScript and Vue.

```
`bash
```

? Select a variant: » - Use arrow-keys. Return to submit.

JavaScript

TypeScript

Customize with create-vue ↗

Nuxt ↗

,

4. Upon completing the aforementioned steps to create the **my-project**, run the following command to install its dependencies:

```
`bash
```

```
cd my-project
```

```
npm install
```

,

or

```
`bash
```

```
cd my-project
```

```
yarn install
```

,

Now that **my-project** is ready to run with default settings, let's add Syncfusion components to the project.

#### Add Syncfusion Vue packages

Syncfusion Vue component packages are available at [npmjs.com](https://www.npmjs.com). To use Syncfusion Vue components in the project, install the corresponding npm package.

This article uses the [Vue Chips component](#) as an example. To use the Vue Chips component in the project, the **@syncfusion/ej2-vue-buttons** package needs to be installed using the following command:

```
`bash
```

```
npm install @syncfusion/ej2-vue-buttons --save
```

,

or

```
`bash
```

```
yarn add @syncfusion/ej2-vue-buttons
```

```
,
```

### Import Syncfusion CSS styles

You can import themes for the Syncfusion Vue component in various ways, such as using CSS or SASS styles from npm packages, CDN, [CRG](#) and [Theme Studio](#). Refer to [themes topic](#) to know more about built-in themes and different ways to refer to themes in a Vue project.

In this article, **Material** theme is applied using CSS styles, which are available in installed packages. The necessary **Material** CSS styles for the Chips component and its dependents were imported into the `<style>` section of **src/App.vue** file.

#### ~/SRC/APP.VUE

```
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-buttons/styles/material.css";
</style>
```

### Add Syncfusion Vue component

Follow the below steps to add the Vue Chips component using **Composition API** or **Options API**:

1.First, import and register the Grid component and its child directives in the **script** section of the **src/App.vue** file. If you are using the **Composition API**, you should add the **setup** attribute to the **script** tag to indicate that Vue will be using the **Composition API**. Find the list of child directives and the tag names that can be used in the Chips component in the following table.

Directive Name	Tag Name
-----	-----
ChipsDirective	e-chips
ChipDirective	e-chip

#### COMPOSITION API (~/SRC/APP.VUE)

```
<script setup>
import { ChipListComponent as EjsChiplist, ChipsDirective as EChips,
ChipDirective as EChip } from '@syncfusion/ej2-vue-buttons';
</script>
```

#### OPTIONS API (~/SRC/APP.VUE)

```
<script>
import { ChipListComponent, ChipsDirective, ChipDirective } from
'@syncfusion/ej2-vue-buttons';
//Component registration
export default {
name: "App",
components: {
```

```
'ejs-chiplist': ChipListComponent,
"e-chips": ChipsDirective,
"e-chip": ChipDirective
}
}
</script>
```

2. Add the component definition in template section.

### ~/SRC/APP.VUE

```
<template>
<ejs-chiplist id="chip">
<e-chips>
<e-chip text="Andrew"></e-chip>
<e-chip text="Janet"></e-chip>
<e-chip text="Laura"></e-chip>
<e-chip text="Margaret"></e-chip>
</e-chips>
</ejs-chiplist>
</template>
```

Here is the summarized code for the above steps in the **src/App.vue** file:

### COMPOSITION API (~/SRC/APP.VUE)

```
<template>
<ejs-chiplist id="chip">
<e-chips>
<e-chip text="Andrew"></e-chip>
<e-chip text="Janet"></e-chip>
<e-chip text="Laura"></e-chip>
<e-chip text="Margaret"></e-chip>
</e-chips>
</ejs-chiplist>
</template>
<script setup>
import { ChipListComponent as EjsChiplist, ChipsDirective as EChips,
ChipDirective as EChip } from '@syncfusion/ej2-vue-buttons';
</script>
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-buttons/styles/material.css";
</style>
```

### OPTIONS API (~/SRC/APP.VUE)

```
<template>
<ejs-chiplist id="chip">
<e-chips>
<e-chip text="Andrew"></e-chip>
<e-chip text="Janet"></e-chip>
<e-chip text="Laura"></e-chip>
<e-chip text="Margaret"></e-chip>
</e-chips>
</ejs-chiplist>
```

```
</template>
<script>
import { ChipListComponent, ChipsDirective, ChipDirective } from
'@syncfusion/ej2-vue-buttons';
//Component registration
export default {
  name: "App",
  components: {
    'ejs-chiplist': ChipListComponent,
    "e-chips": ChipsDirective,
    "e-chip": ChipDirective
  }
}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-buttons/styles/material.css";
</style>
```

### Run the project

To run the project, use the following command:

```
`bash
```

```
npm run dev
```

```
,
```

or

```
`bash
```

```
yarn run dev
```

```
,
```

The output will appear as follows:



Andrew Janet Laura Margaret

For migrating from Vue 2 to Vue 3, refer to the [migration](#) documentation.

See also

- [Getting Started with Vue UI Components using Composition API and TypeScript](#)
- [Getting Started with Vue UI Components using Options API and TypeScript](#)

### Types in Vue Chips component

The ChipList control has the following types.

- Input Chip
- Choice Chip

- Filter Chip
- Action Chip

### Input Chip

Input Chip holds information in compact form. It converts user input into chips.

#### APP.VUE

```
<template>
  <ejs-chiplist id="chip">
    <e-chips>
      <e-chip text="Andrew"></e-chip>
      <e-chip text="Janet"></e-chip>
      <e-chip text="Laura"></e-chip>
      <e-chip text="Margaret"></e-chip>
    </e-chips>
  </ejs-chiplist>
</template>
<script>
import Vue from 'vue';
import { ChipListPlugin } from '@syncfusion/ej2-vue-buttons';
Vue.use(ChipListPlugin);
export default {}
</script>
```

{% previewsample "page.domainurl/code-snippet/chips/default-cs9" %}

### Choice Chip

Choice Chip allows you to select a single chip from the set of ChipList/ChipCollection. It can be enabled by setting the **selection** property to **Single**.

#### APP.VUE

```
<template>
  <ejs-chiplist id="chip" selection="Single">
    <e-chips>
      <e-chip text="Small"></e-chip>
      <e-chip text="Medium"></e-chip>
      <e-chip text="Large"></e-chip>
      <e-chip text="Extra Large"></e-chip>
    </e-chips>
  </ejs-chiplist>
</template>
<script>
import Vue from 'vue';
import { ChipListPlugin } from '@syncfusion/ej2-vue-buttons';
Vue.use(ChipListPlugin);
export default {}
</script>
```

{% previewsample "page.domainurl/code-snippet/chips/default-cs10" %}

### Filter Chip

Filter Chip allows you to select a multiple chip from the set of ChipList/ChipCollection. It can be enabled by setting the **selection** property to **Multiple**.



**APP.VUE**

```
<template>
  <ejs-chiplist id="chip" selection="Multiple">
    <e-chips>
      <e-chip text="Chai"></e-chip>
      <e-chip text="Chang"></e-chip>
      <e-chip text="Aniseed Syrup"></e-chip>
      <e-chip text="Ikura"></e-chip>
    </e-chips>
  </ejs-chiplist>
</template>
<script>
import Vue from 'vue';
import { ChipListPlugin } from '@syncfusion/ej2-vue-buttons';
Vue.use(ChipListPlugin);
export default {}
</script>
```

{% previewsample "page.domainurl/code-snippet/chips/default-cs11" %}

**Action Chip**

The Action Chip triggers the event like click or delete, which helps doing action based on the event.

**APP.VUE**

```
<template>
  <ejs-chiplist id="chip" v-on:click.native="chipclick">
    <e-chips>
      <e-chip text="Send a text"></e-chip>
      <e-chip text="Set a remainder"></e-chip>
      <e-chip text="Read my emails"></e-chip>
      <e-chip text="Set alarm"></e-chip>
    </e-chips>
  </ejs-chiplist>
</template>
<script>
import Vue from 'vue';
import { ChipListPlugin } from '@syncfusion/ej2-vue-buttons';
Vue.use(ChipListPlugin);
export default {
  methods: {
    chipclick: function(e) {
      alert('you have clicked ' + e.target.textContent);
    }
  }
}
</script>
```

{% previewsample "page.domainurl/code-snippet/chips/default-cs12" %}

**Deletable Chip**

Deletable Chip allows you to delete a chip from ChipList/ChipCollection. It can be enabled by setting the enableDelete property to true.

**APP.VUE**

```

<template>
  <ejs-chiplist id="chip" enableDelete="true">
    <e-chips>
      <e-chip text="Send a text"></e-chip>
      <e-chip text="Set a remainder"></e-chip>
      <e-chip text="Read my emails"></e-chip>
      <e-chip text="Set alarm"></e-chip>
    </e-chips>
  </ejs-chiplist>
</template>
<script>
import Vue from 'vue';
import { ChipListPlugin } from '@syncfusion/ej2-vue-buttons';
Vue.use(ChipListPlugin);
export default {
  methods: {
    chipclick: function(e) {
      alert('you have clicked ' + e.target.textContent);
    }
  }
}
</script>

```

{% previewsample "page.domainurl/code-snippet/chips/default-cs13" %}

### Customization in Vue Chips component

This section explains the customization of styles, leading icons, avatar, and trailing icons in Chip control.

#### Styles

The Chip control has the following predefined styles that can be defined using the `cssClass` property.

Class	Description
-----	-----
e-primary	Represents a primary chip.
e-success	Represents a positive chip.
e-info	Represents an informative chip.
e-warning	Represents a chip with caution.
e-danger	Represents a negative chip.

#### APP.VUE

```

<template>
  <ejs-chiplist id="chip" enableDelete="true">
    <e-chips>
      <e-chip text="Primary" cssClass="e-primary"></e-chip>
      <e-chip text="Success" cssClass="e-success"></e-chip>
      <e-chip text="Info" cssClass="e-info"></e-chip>
      <e-chip text="Warning" cssClass="e-warning"></e-chip>
      <e-chip text="Danger" cssClass="e-danger"></e-chip>
    </e-chips>
  </ejs-chiplist>
</template>

```

```
<script>
import Vue from 'vue';
import { ChipListPlugin } from '@syncfusion/ej2-vue-buttons';
Vue.use(ChipListPlugin);
export default {
}
</script>
```

{% previewsample "page.domainurl/code-snippet/chips/default-cs2" %}

### Leading Icon

You can add and customize the leading icon of chip using the `leadingIconCss` property.

#### APP.VUE

```
<template>
  <ejs-chiplist id="chip">
    <e-chips>
      <e-chip text="Andrew" leadingIconCss='andrew'></e-chip>
      <e-chip text="Janet" leadingIconCss='janet'></e-chip>
      <e-chip text="Laura" leadingIconCss='laura'></e-chip>
      <e-chip text="Margaret" leadingIconCss='margaret'></e-chip>
    </e-chips>
  </ejs-chiplist>
</template>
<script>
import Vue from 'vue';
import { ChipListPlugin } from '@syncfusion/ej2-vue-buttons';
Vue.use(ChipListPlugin);
export default {
}
</script>
<style>
#chip .andrew {
  background-image: url('./andrew.png')
}
#chip .margaret {
  background-image: url('./margaret.png')
}
#chip .laura {
  background-image: url('./laura.png')
}
#chip .janet {
  background-image: url('./janet.png')
}
</style>
```

{% previewsample "page.domainurl/code-snippet/chips/default-cs3" %}

### Avatar

You can add and customize the avatar of chip using the `avatarIconCss` property.

#### APP.VUE

```
<template>
  <ejs-chiplist id="chip">
```

```

        <e-chips>
          <e-chip text="Andrew" avatarIconCss='andrew'></e-chip>
          <e-chip text="Janet" avatarIconCss='janet'></e-chip>
          <e-chip text="Laura" avatarIconCss='laura'></e-chip>
          <e-chip text="Margaret" avatarIconCss='margaret'></e-chip>
        </e-chips>
      </ejs-chiplist>
    </template>
    <script>
    import Vue from 'vue';
    import { ChipListPlugin } from '@syncfusion/ej2-vue-buttons';
    Vue.use(ChipListPlugin);
    export default {
    }
    </script>
    <style>
    #chip .andrew {
      background-image:
url('https://ej2.syncfusion.com/demos/src/chips/images/andrew.png')
    }
    #chip .margaret {
      background-image:
url('https://ej2.syncfusion.com/demos/src/chips/images/margaret.png')
    }
    #chip .laura {
      background-image:
url('https://ej2.syncfusion.com/demos/src/chips/images/laura.png')
    }
    #chip .janet {
      background-image:
url('https://ej2.syncfusion.com/demos/src/chips/images/janet.png')
    }
    </style>

```

{% previewsample "page.domainurl/code-snippet/chips/default-cs4" %}

### Avatar Content

You can add and customize the avatar content of chip using the `avatarText` property.

### APP.VUE

```

<template>
  <ejs-chiplist id="chip">
    <e-chips>
      <e-chip text="Andrew" avatarText= 'A'></e-chip>
      <e-chip text="Janet" avatarText= 'J'></e-chip>
      <e-chip text="Laura" avatarText= 'L'></e-chip>
      <e-chip text="Margaret" avatarText= 'M'></e-chip>
    </e-chips>
  </ejs-chiplist>
</template>
<script>
import Vue from 'vue';
import { ChipListPlugin } from '@syncfusion/ej2-vue-buttons';
Vue.use(ChipListPlugin);
export default {

```

```
}
</script>
```

```
{% previewsample "page.domainurl/code-snippet/chips/default-cs5" %}
```

### Trailing Icon

You can add and customize the trailing icon of chip using the `trailingIconCss` property.

#### APP.VUE

```
<template>
  <ejs-chiplist id="chip">
    <e-chips>
      <e-chip text="Andrew" trailingIconCss= 'e-dlt-btn'></e-chip>
      <e-chip text="Janet" trailingIconCss= 'e-dlt-btn'></e-chip>
      <e-chip text="Laura" trailingIconCss= 'e-dlt-btn'></e-chip>
      <e-chip text="Margaret" trailingIconCss= 'e-dlt-btn'></e-chip>
    </e-chips>
  </ejs-chiplist>
</template>
<script>
import Vue from 'vue';
import { ChipListPlugin } from '@syncfusion/ej2-vue-buttons';
Vue.use(ChipListPlugin);
export default {
}
</script>
```

```
{% previewsample "page.domainurl/code-snippet/chips/default-cs6" %}
```

### Outline Chip

Outline chip has the border with the background transparent. It can be set using the `cssClass` property.

#### APP.VUE

```
<template>
<div>
  <ejs-chiplist id="chip" cssClass="e-outline">
    <e-chips>
      <e-chip text="Chai"></e-chip>
      <e-chip text="Chang"></e-chip>
      <e-chip text="Aniseed Syrup"></e-chip>
      <e-chip text="Ikura"></e-chip>
    </e-chips>
  </ejs-chiplist>
  <ejs-chiplist id="chip" cssClass="e-outline" enableDelete="true">
    <e-chips>
      <e-chip text="Andrew"></e-chip>
      <e-chip text="Janet"></e-chip>
      <e-chip text="Laura"></e-chip>
      <e-chip text="Margaret"></e-chip>
    </e-chips>
  </ejs-chiplist>
</div>
</template>
<script>
```

```
import Vue from 'vue';
import { ChipListPlugin } from '@syncfusion/ej2-vue-buttons';
Vue.use(ChipListPlugin);
export default {
}
</script>
```

{% previewsample "page.domainurl/code-snippet/chips/default-cs7" %}

### Style in Vue Chips component

The following content provides the exact CSS structure that can be used to modify the control's appearance based on the user preference.

#### Customizing the chip text

Use the following CSS to customize the chip text properties.

```
,
.e-chip .e-chip-text {
font-size: 20px;
color: black;
font-weight: normal;
}
,
```

#### Customizing the chip icon

Use the following CSS to customize the chip icon properties.

```
,
.e-chip .e-icon {
background-image: url('https://ej2.syncfusion.com/demos/src/chips/images/laura.png');
opacity: 0.8;
}
,
```

#### Customizing the chip delete button

Use the following CSS to customize the chip delete button.

```
,
.e-chip-list .e-chip .e-chip-delete.e-dlt-btn {
color: #e3165b;
font-size: 12px;
}
,
```

### Customizing the chip outline

Use the following CSS to customize the chip outline.

```
,  
  
.e-chip-list .e-chip.e-outline {  
  border-color: #e3165b;  
  border-width: 3px;  
}  
,
```

### Customizing the chip on selection

Use the following CSS to customize the chip on selection.

```
,  
  
/ To customize single chip on selection /  
.e-chip-list.e-selection .e-chip.e-active {  
  background-color: #ffca1c;  
  color: #e3165b;  
}  
  
/ To customize multiple chip on selection /  
.e-chip-list .e-chip.e-active {  
  background-color: #e3165b;  
  color: white;  
}  
,
```

### Customizing the chip avatar text

Use the following CSS to customize the chip avatar text properties.

```
,  
  
.e-chip-list .e-chip .e-chip-avatar {  
  background-color: #d51a1a;  
  color: #fafafa;  
}  
,
```

### Accessibility in Vue Chips component

The Chips component followed the accessibility guidelines and standards, including [ADA, Section 508](#), [WCAG 2.2](#) standards, and [WCAG roles](#) that are commonly used to evaluate accessibility.

The accessibility compliance for the Chips component is outlined below.

| Accessibility Criteria | Compatibility |

| -- | -- |

| [WCAG 2.2](#) Support |  |

| [Section 508](#) Support |  |

| Screen Reader Support |  |

| Right-To-Left Support |  |

| Color Contrast |  |

| Mobile Device Support |  |

| Keyboard Navigation Support |  |

| [Accessibility Checker](#) Validation |  |

| [Axe-core](#) Accessibility Validation |  |

<style>

.post .post-content img {

display: inline-block;

margin: 0.5em 0;

}

</style>

<div> - All features of the component meet the requirement.</div>

<div> - Some features of the component do not meet the requirement.</div>

<div> - The component does not meet the requirement.</div>

#### [WAI-ARIA attributes](#)

The Chips component followed the [WAI-ARIA](#) patterns to meet the accessibility. The following ARIA attributes are used in the Chips component:

| Attributes | Purpose |

| --- | --- |



- | `role=checkbox` | Indicates the ChipList component wrapper element as `checkbox`. |
- | `role=option` | Used to convey a significant and contextual message to the user(ChipList). |
- | `role=button` | Used to convey a significant and contextual message to the user(Single Chip). |
- | `aria-label` | Provides an accessible name for the Chip. |
- | `aria-selected` | Indicates the element is selected. |
- | `aria-disabled` | Indicates element is perceivable but disabled. |
- | `aria-multiselectable` | Indicates multiple items to be selected. |

### Keyboard interaction

The Chips component followed the [keyboard interaction](#) guideline, making it easy for people who use assistive technologies (AT) and those who completely rely on keyboard navigation. The following keyboard shortcuts are supported by the Chips component.

| Keyboard shortcuts | Actions |

|-----|-----|

| Enter / Space | Selects the targeted chip from the ChipList/ChipCollection. |

| Delete / Backspace | Deletes the targeted chip from the ChipList/ChipCollection. |

### Ensuring accessibility

The Chips component's accessibility levels are ensured through an [accessibility-checker](#) and [axe-core](#) software tools during automated testing.

The accessibility compliance of the Chips component is shown in the following sample. Open the [sample](#) in a new window to evaluate the accessibility of the Chips component with accessibility tools.

```
{% previewsample "https://ej2.syncfusion.com/accessibility/chips.html" %}
```

See also

- [Accessibility in Syncfusion Vue components](#)

## Circular gauge

### Getting Started with the Vue Circular Gauge Component in Vue 2

This article provides a step-by-step guide for setting up a Vue 2 project using [Vue-CLI](#) and integrating the Syncfusion Vue Circular Gauge component.

#### Prerequisites

[System requirements for Syncfusion Vue UI components](#)

#### Dependencies

Below is the list of minimum dependencies required to use the Circular Gauge.

```
`javascript
```

```
|-- @syncfusion/ej2-vue-circulargauge
```

```
|-- @syncfusion/ej2-base
```

```
|-- @syncfusion/ej2-buttons  
|-- @syncfusion/ej2-popups  
|-- @syncfusion/ej2-splitbuttons  
|-- @syncfusion/ej2-vue-base  
|-- @syncfusion/ej2-svg-base  
|-- @syncfusion/ej2-circulargauge  
`
```

### Setting up the Vue 2 project

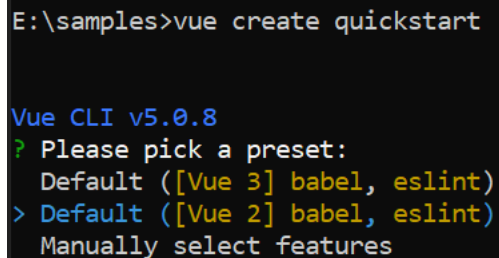
To generate a Vue 2 project using Vue-CLI, use the [vue create](#) command. Follow these steps to install Vue CLI and create a new project:

```
`bash  
npm install -g @vue/cli  
vue create quickstart  
cd quickstart  
npm run serve  
`
```

or

```
`bash  
yarn global add @vue/cli  
vue create quickstart  
cd quickstart  
yarn run serve  
`
```

When creating a new project, choose the option **Default ([Vue 2] babel, eslint)** from the menu.



```
E:\samples>vue create quickstart  
  
Vue CLI v5.0.8  
? Please pick a preset:  
  Default ([Vue 3] babel, eslint)  
> Default ([Vue 2] babel, eslint)  
  Manually select features
```

Once the **quickstart** project is set up with default settings, proceed to add Syncfusion components to the project.

### Add Syncfusion Vue packages

Syncfusion packages are available at [npmjs.com](https://www.npmjs.com). To use Vue components, install the required npm package.

This article uses the [Vue Circular gauge component](#) as an example. Install the `@syncfusion/ej2-vue-circulargauge` package by running the following command:

```
`bash
```

```
npm install @syncfusion/ej2-vue-circulargauge --save
```

```
,
```

or

```
`bash
```

```
yarn add @syncfusion/ej2-vue-circulargauge
```

```
,
```

### Adding Syncfusion Vue Circular Gauge component

Follow the below steps to add the Vue Circular Gauge component:

1\ First, import and register the Circular Gauge component in the `script` section of the `src/App.vue` file.

#### ~/SRC/APP.VUE

```
<script>
import { CircularGaugeComponent } from '@syncfusion/ej2-vue-circulargauge';
export default {
  components: {
    'ejs-circulargauge': CircularGaugeComponent
  }
}
</script>
```

2\ In the `template` section, define the Circular Gauge component.

#### ~/SRC/APP.VUE

```
<template>
<div id="app">
<div class='wrapper'>
<ejs-circulargauge ></ejs-circulargauge>
</div>
</div>
</template>
```

Here is the summarized code for the above steps in the `src/App.vue` file:

#### ~/SRC/APP.VUE

```
<template>
  <div id="app">
    <div class='wrapper'>
      <ejs-circulargauge ></ejs-circulargauge>
    </div>
  </div>
```

```
    </div>
  </template>
  <script>
  import { CircularGaugeComponent } from '@syncfusion/ej2-vue-circulargauge';
  export default {
    components: {
      'ejs-circulargauge': CircularGaugeComponent
    },
  }
  </script>
  <style>
    .wrapper {
      max-width: 300px;
      margin: 0 auto;
    }
  </style>
```

### Run the project

To run the project, use the following command:

```
`bash
```

```
npm run serve
```

```
,
```

or

```
`bash
```

```
yarn run serve
```

```
,
```

```
{% previewsample "page.domainurl/code-snippet/circular-gauge/getting-started-cs64" %}
```

### Set Pointer Value

You can change the pointer value in the above sample using [value](#) property in [pointers](#).

#### ~/SRC/APP.VUE

```
<template>
  <div id="app">
    <div class='wrapper'>
      <ejs-circulargauge >
        <e-axes>
          <e-axis>
            <e-pointers>
              <e-pointer :value='val' />
            </e-pointers>
          </e-axis>
        </e-axes>
      </ejs-circulargauge>
    </div>
  </div>
</template>
<script>
```

```
import { CircularGaugeComponent, AxisDirective, AxesDirective,
PointerDirective, PointersDirective } from '@syncfusion/ej2-vue-
circulargauge';
export default {
  components: {
    'ejs-circulargauge': CircularGaugeComponent,
    'e-axis': AxisDirective,
    'e-axes': AxesDirective,
    'e-pointer': PointerDirective,
    'e-pointers': PointersDirective
  },
  data () {
    return {
      val : 35
    }
  }
}
</script>
<style>
  .wrapper {
    max-width: 300px;
    margin: 0 auto;
  }
</style>
```

{% previewsample "page.domainurl/code-snippet/circular-gauge/getting-started-cs65" %}

### Getting Started with the Vue Circular Gauge Component in Vue 3

This article provides a step-by-step guide for setting up a [Vite](#) project with a JavaScript environment and integrating the Syncfusion Vue Circular Gauge component using the [Composition API](#) / [Options API](#).

The **Composition API** is a new feature introduced in Vue.js 3 that provides an alternative way to organize and reuse component logic. It allows developers to write components as functions that use smaller, reusable functions called composition functions to manage their properties and behavior.

The **Options API** is the traditional way of writing Vue.js components, where the component logic is organized into a series of options that define the component's properties and behavior. These options include data, methods, computed properties, watchers, lifecycle hooks, and more.

#### Prerequisites

##### [System requirements for Syncfusion Vue UI components](#)

##### Set up the Vite project

A recommended approach for beginning with Vue is to scaffold a project using [Vite](#). To create a new Vite project, use one of the commands that are specific to either NPM or Yarn.

```
`bash
```

```
npm create vite@latest
```

```
`
```

or

```
`bash
```

```
yarn create vite
```

`

Using one of the above commands will lead you to set up additional configurations for the project as below:

1. Define the project name: We can specify the name of the project directly. Let's specify the name of the project as `my-project` for this article.

`bash

? Project name: » my-project

`

2. Select `Vue` as the framework. It will create a Vue 3 project.

`bash

? Select a framework: » - Use arrow-keys. Return to submit.

Vanilla

Vue

React

Preact

Lit

Svelte

Others

`

3. Choose `JavaScript` as the framework variant to build this Vite project using JavaScript and Vue.

`bash

? Select a variant: » - Use arrow-keys. Return to submit.

JavaScript

TypeScript

Customize with create-vue ↗

Nuxt ↗

`

4. Upon completing the aforementioned steps to create the `my-project`, run the following command to install its dependencies:

`bash

cd my-project

npm install

`

or

```
`bash
cd my-project
yarn install
`
```

Now that `my-project` is ready to run with default settings, let's add Syncfusion components to the project.

#### Add Syncfusion Vue packages

Syncfusion Vue component packages are available at [npmjs.com](https://www.syncfusion.com). To use Syncfusion Vue components in the project, install the corresponding npm package.

This article uses the [Vue Circular Gauge component](#) as an example. To use the Vue Circular Gauge component in the project, the `@syncfusion/ej2-vue-circulargauge` package needs to be installed using the following command:

```
`bash
npm install @syncfusion/ej2-vue-circulargauge --save
`
```

or

```
`bash
yarn add @syncfusion/ej2-vue-circulargauge
`
```

#### Add Syncfusion Vue component

Follow the below steps to add the Vue Circular Gauge component using `Composition API` or `Options API`:

1. First, import and register the Circular Gauge component and its child directives in the `script` section of the `src/App.vue` file. If you are using the `Composition API`, you should add the `setup` attribute to the `script` tag to indicate that Vue will be using the `Composition API`.

##### COMPOSITION API (~SRC/APP.VUE)

```
<script setup>
import { CircularGaugeComponent as EjsCirculargauge, AxesDirective as EAxes,
AxisDirective as EAxis, PointersDirective as EPointers, PointerDirective as
EPointer, RangesDirective as ERanges, RangeDirective as ERRange } from
'@syncfusion/ej2-vue-circulargauge';
</script>
```

##### OPTIONS API (~SRC/APP.VUE)

```
<script>
import { CircularGaugeComponent, AxesDirective, AxisDirective,
PointersDirective, PointerDirective, RangesDirective, RangeDirective } from
'@syncfusion/ej2-vue-circulargauge';
//Component registration
export default {
```

```
name: "App",
components: {
  'ejs-circulargauge' : CircularGaugeComponent,
  'e-axes' : AxesDirective,
  'e-axis' : AxisDirective,
  'e-pointers' : PointersDirective,
  'e-pointer' : PointerDirective,
  'e-ranges' : RangesDirective,
  'e-range' : RangeDirective
}
}
</script>
```

2. In the **template** section, define the Circular Gauge component with the axis, pointer and range property definitions.

#### ~/SRC/APP.VUE

```
<template>
<ejs-circulargauge :title ='title' orientation='Horizontal'>
<e-axes>
<e-axis minimum=0 maximum=200>
<e-pointers>
<e-pointer value=140></e-pointer>
</e-pointers>
<e-ranges>
<e-range start=0 end=80 startWidth=15 endWidth=15></e-range>
<e-range start=80 end=120 startWidth=15 endWidth=15></e-range>
<e-range start=120 end=140 startWidth=15 endWidth=15></e-range>
<e-range start=140 end=200 startWidth=15 endWidth=15></e-range>
</e-ranges>
</e-axis>
</e-axes>
</ejs-circulargauge>
</template>
```

3. In the **script** section, declare the values for the properties defined in the **template** section.

#### COMPOSITION API (~/SRC/APP.VUE)

```
<script setup>
const title = "Circular Gauge";
</script>
```

#### OPTIONS API (~/SRC/APP.VUE)

```
<script>
data() {
  return {
    title: 'Circular Gauge'
  };
}
</script>
```



Here is the summarized code for the above steps in the **src/App.vue** file:

#### **COMPOSITION API (~SRC/APP.VUE)**

```
<template>
<ejs-circulargauge :title ='title' orientation='Horizontal'>
<e-axes>
<e-axis minimum=0 maximum=200>
<e-pointers>
<e-pointer value=140></e-pointer>
</e-pointers>
<e-ranges>
<e-range start=0 end=80 startWidth=15 endWidth=15></e-range>
<e-range start=80 end=120 startWidth=15 endWidth=15></e-range>
<e-range start=120 end=140 startWidth=15 endWidth=15></e-range>
<e-range start=140 end=200 startWidth=15 endWidth=15></e-range>
</e-ranges>
</e-axis>
</e-axes>
</ejs-circulargauge>
</template>
<script setup>
import { CircularGaugeComponent as EjsCirculargauge, AxesDirective as EAxes,
AxisDirective as EAxis, PointersDirective as EPointers, PointerDirective as
EPointer, RangesDirective as ERanges, RangeDirective as ERange } from
'@syncfusion/ej2-vue-circulargauge';
const title = "Circular Gauge";
</script>
```

#### **OPTIONS API (~SRC/APP.VUE)**

```
<template>
<ejs-circulargauge :title ='title' orientation='Horizontal'>
<e-axes>
<e-axis minimum=0 maximum=200>
<e-pointers>
<e-pointer value=140></e-pointer>
</e-pointers>
<e-ranges>
<e-range start=0 end=80 startWidth=15 endWidth=15></e-range>
<e-range start=80 end=120 startWidth=15 endWidth=15></e-range>
<e-range start=120 end=140 startWidth=15 endWidth=15></e-range>
<e-range start=140 end=200 startWidth=15 endWidth=15></e-range>
</e-ranges>
</e-axis>
</e-axes>
</ejs-circulargauge>
</template>
<script>
import { CircularGaugeComponent, AxesDirective, AxisDirective,
PointersDirective, PointerDirective, RangesDirective, RangeDirective } from
'@syncfusion/ej2-vue-circulargauge';
// Component registration
export default {
name: "App",
// Declaring component and its directives
components: {
```

```
'ejs-circulargauge' : CircularGaugeComponent,  
'e-axes' : AxesDirective,  
'e-axis' : AxisDirective,  
'e-pointers': PointersDirective,  
'e-pointer' : PointerDirective,  
'e-ranges' : RangesDirective,  
'e-range' : RangeDirective  
},  
// Bound properties declarations  
data() {  
  return {  
    title: 'Circular Gauge'  
  };  
}  
};  
</script>
```

### Run the project

To run the project, use the following command:

```
`bash
```

```
npm run dev
```

```
`
```

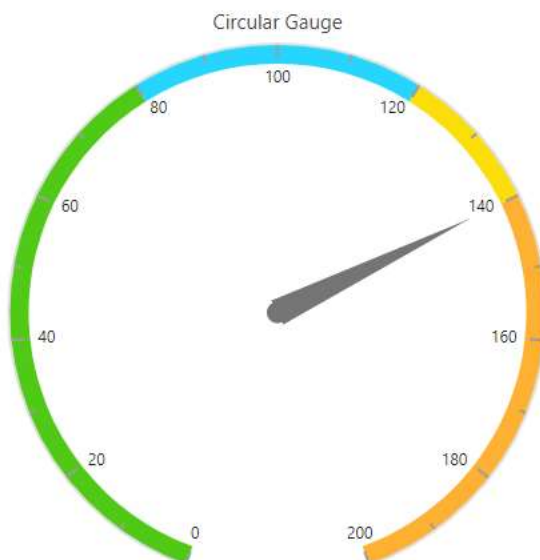
or

```
`bash
```

```
yarn run dev
```

```
`
```

The output will appear as follows:



**Sample:** [vue3-circulargauge-getting-started](#).

For migrating from Vue 2 to Vue 3, refer to the [migration](#) documentation.

See also

- [Getting Started with Vue UI Components using Composition API and TypeScript](#)
- [Getting Started with Vue UI Components using Options API and TypeScript](#)

## Gauge dimensions in Vue Circular gauge component

### Size for Container

Circular gauge can render to its container size. You can set the size via inline or CSS as demonstrated below.

#### APP.VUE

```
<template>
  <div id="app">
    <div class='wrapper'>
      <ejs-circulargauge ></ejs-circulargauge>
    </div>
  </div>
</template>
<script>
import Vue from 'vue';
import { CircularGaugePlugin } from '@syncfusion/ej2-vue-circulargauge';
Vue.use(CircularGaugePlugin);
export default {}
</script>
<style>
.wrapper {
  max-width: 300px;
  margin: 0 auto;
}
</style>
```

{% previewsample "page.domainurl/code-snippet/circular-gauge/getting-started-cs28" %}

### Size for Circular Gauge

<!-- markdownlint-disable MD036 -->

You can also set size for the gauge directly through [width](#) and [height](#) properties.

#### In Pixel

You can set the size of the gauge in pixel as demonstrated below.

#### APP.VUE

```
<template>
  <div id="app">
    <div class='wrapper'>
      <ejs-circulargauge width='650' height='350' ></ejs-
circulargauge>
    </div>
  </div>
</template>
```

```

<script>
import Vue from 'vue';
import { CircularGaugePlugin } from '@syncfusion/ej2-vue-circulargauge';
Vue.use(CircularGaugePlugin);
export default {}
</script>
<style>
  .wrapper {
    max-width: 300px;
    margin: 0 auto;
  }
</style>

```

{% previewsample "page.domainurl/code-snippet/circular-gauge/getting-started-cs29" %}

### In Percentage

By setting value in percentage, gauge gets its dimension with respect to its container. For example, when

the height is '50%', gauge renders to half of the container height.

### APP.VUE

```

<template>
  <div id="app">
    <div class='wrapper'>
      <ejs-circulargauge width='80%' height='50%'></ejs-circulargauge>
    </div>
  </div>
</template>
<script>
import Vue from 'vue';
import { CircularGaugePlugin } from '@syncfusion/ej2-vue-circulargauge';
Vue.use(CircularGaugePlugin);
export default {}
</script>
<style>
  .wrapper {
    max-width: 300px;
    margin: 0 auto;
  }
</style>

```

{% previewsample "page.domainurl/code-snippet/circular-gauge/getting-started-cs30" %}

Note: When you do not specify the size, it takes 450px as the height and window size as its width.

### Gauge axes in Vue Circular gauge component

By default, gauge will be displayed with an axis. Each axis contains its own ranges, pointers and annotation.

### Axis Customization

You can customize the width and color of an axis line by using [lineStyle](#) property.

Background for an axis can be customized by using [background](#)

property.

### APP.VUE

```
<template>
  <div id="app">
    <div class='wrapper'>
      <ejs-circulargauge>
        <e-axes>
          <e-axis :lineStyle= 'lineStyle' background= 'rgba(0, 128,
128, 0.3)'>
        </e-axis>
      </e-axes>
    </ejs-circulargauge>
  </div>
</div>
</template>
<script>
import Vue from 'vue';
import { CircularGaugePlugin } from "@syncfusion/ej2-vue-circulargauge";
Vue.use(CircularGaugePlugin);
export default {
  data: function () {
    return {
      lineStyle: {
        width: 2,
        color: 'red'
      }
    }
  }
};
</script>
<style>
  .wrapper {
    max-width: 300px;
    margin: 0 auto;
  }
</style>
```

{% previewsample "page.domainurl/code-snippet/circular-gauge/getting-started-cs10" %}

### Angles and Direction

Circular gauge axis can sweep from 0 to 360 degrees. By default start angle of an axis is 200 degree and end angle is 160 degree and you can customize this option by using

[startAngle](#) and [endAngle](#) property.

### APP.VUE

```
<template>
  <div id="app">
    <div class='wrapper'>
      <ejs-circulargauge :axes = 'axes'>
    </ejs-circulargauge>
    </div>
  </div>
</template>
```

```

<script>
import Vue from 'vue';
import { CircularGaugePlugin } from "@syncfusion/ej2-vue-circulargauge";
Vue.use(CircularGaugePlugin);
export default {
  data: function () {
    return {
      axes: [{
        startAngle: 270,
        endAngle: 90
      }]
    }
  };
};
</script>
<style>
.wrapper {
  max-width: 300px;
  margin: 0 auto;
}
</style>

```

{% previewsample "page.domainurl/code-snippet/circular-gauge/getting-started-cs11" %}

#### *AntiClockWise Direction*

The [direction](#) property enables you to render the gauge axis either in

ClockWise or in AntiClockWise direction.

#### **APP.VUE**

```

<template>
<div id="app">
  <div class='wrapper'>
    <ejs-circulargauge >
      <e-axes>
        <e-axis direction= 'AntiClockWise'>
        </e-axis>
      </e-axes>
    </ejs-circulargauge>
  </div>
</div>
</template>
<script>
import Vue from 'vue';
import { CircularGaugePlugin } from "@syncfusion/ej2-vue-circulargauge";
Vue.use(CircularGaugePlugin);
export default { };
</script>
<style>
.wrapper {
  max-width: 300px;
  margin: 0 auto;
}
</style>

```

{% previewsample "page.domainurl/code-snippet/circular-gauge/getting-started-cs12" %}

### Axis Radius

By default, radius of an axis is calculated based on the available size.

You can customize this, by using [radius](#) property.

It takes value either in **percentage** or in **pixel**.

#### In Pixel

You can set the radius of the gauge in pixel as demonstrated below,

#### APP.VUE

```
<template>
<div id="app">
  <div class='wrapper'>
    <ejs-circulargauge >
      <e-axes>
        <e-axis radius= '150'>
        </e-axis>
      </e-axes>
    </ejs-circulargauge>
  </div>
</div>
</template>
<script>
import Vue from 'vue';
import { CircularGaugePlugin } from "@syncfusion/ej2-vue-circulargauge";
Vue.use(CircularGaugePlugin);
export default { };
</script>
<style>
.wrapper {
  max-width: 300px;
  margin: 0 auto;
}
</style>
```

{% previewsample "page.domainurl/code-snippet/circular-gauge/getting-started-cs13" %}

### In Percentage

By setting value in percentage, gauge gets its dimension with respect to its available size.

For example, when the radius is '50%', gauge renders to half of the available size.

#### APP.VUE

```
<template>
<div id="app">
  <div class='wrapper'>
    <ejs-circulargauge>
      <e-axes>
        <e-axis radius= '50%'>
        </e-axis>
      </e-axes>
    </ejs-circulargauge>
  </div>
```

```

</div>
</template>
<script>
import Vue from 'vue';
import { CircularGaugePlugin } from '@syncfusion/ej2-vue-circulargauge';
Vue.use(CircularGaugePlugin);
export default { };
</script>
<style>
.wrapper {
  max-width: 300px;
  margin: 0 auto;
}
</style>

```

{% previewsample "page.domainurl/code-snippet/circular-gauge/getting-started-cs14" %}

### Ticks

You can customize the [height](#),

[color](#) and

[width](#) of major and minor ticks by

using [majorTicks](#)

and [minorTicks](#) property.

By default, [interval](#) for

**majorTicks** will be calculated automatically

and also you can customize the interval for major and minor ticks using

**interval** property.

### APP.VUE

```

<template>
  <div id="app">
    <div class='wrapper'>
      <ejs-circulargauge >
        <e-axes>
          <e-axis :majorTicks= 'majorTicks' :minorTicks=
'minorTicks' >
            </e-axis>
          </e-axes>
        </ejs-circulargauge>
      </div>
    </div>
  </template>
<script>
import Vue from 'vue';
import { CircularGaugePlugin } from '@syncfusion/ej2-vue-circulargauge';
Vue.use(CircularGaugePlugin);
export default {
  data: function () {
    return {

```



```

        majorTicks: {
            interval: 10,
            color: 'red',
            height: 10,
            width: 3
        },
        minorTicks: {
            interval: 5,
            color: 'green',
            height: 5,
            width: 2
        }
    }
}
};
</script>
<style>
    .wrapper {
        max-width: 300px;
        margin: 0 auto;
    }
</style>

```

{% previewsample "page.domainurl/code-snippet/circular-gauge/getting-started-cs15" %}

#### Tick Position

Both minor and major ticks can be moved by using [offset](#) and [position](#) property. The [offset](#) defines the distance between the axis and ticks.

By default, offset value is 0. The [position](#) will place the ticks either inside or outside of the axis.

By default, ticks will be placed **inside** the axis.

#### APP.VUE

```

<template>
    <div id="app">
        <div class='wrapper'>
            <ejs-circulargauge>
                <e-axes>
                    <e-axis :majorTicks= 'majorTicks' :minorTicks=
'minorTicks'>
                        </e-axis>
                    </e-axes>
                </ejs-circulargauge>
            </div>
        </div>
    </template>
<script>
import Vue from 'vue';
import { CircularGaugePlugin } from "@syncfusion/ej2-vue-circulargauge";
Vue.use(CircularGaugePlugin);
export default {
    data: function () {
        return {
            majorTicks: {
                interval: 10,

```

```

        color: 'red',
        height: 10,
        width: 3,
        position: 'Inside',
        offset: 5
    },
    minorTicks: {
        interval: 5,
        color: 'green',
        height: 5,
        width: 2,
        position: 'Inside',
        offset: 5
    }
}
};
</script>
<style>
    .wrapper {
        max-width: 300px;
        margin: 0 auto;
    }
</style>

```

{% previewsample "page.domainurl/code-snippet/circular-gauge/getting-started-cs16" %}

### Labels

Labels of an axis can be customized by using [font](#) property in [labelStyle](#) options.

### APP.VUE

```

<template>
    <div id="app">
        <div class='wrapper'>
            <ejs-circulargauge>
                <e-axes>
                    <e-axis :labelStyle ='labelStyle' >
                    </e-axis>
                </e-axes>
            </ejs-circulargauge>
        </div>
    </div>
</template>
<script>
import Vue from 'vue';
import { CircularGaugePlugin } from "@syncfusion/ej2-vue-circulargauge";
Vue.use(CircularGaugePlugin);
export default {
    data: function () {
        return {
            labelStyle: {
                font: {
                    color: 'red',
                    size: '20px',
                    fontWeight: 'Bold'
                }
            }
        }
    }
}

```

```

    }
  }
}
};
</script>
<style>
  .wrapper {
    max-width: 300px;
    margin: 0 auto;
  }
</style>

```

{% previewsample "page.domainurl/code-snippet/circular-gauge/getting-started-cs17" %}

### Label Position

Labels can be moved by using [offset](#) or [position](#) property.

The [offset](#) defines the distance between the labels and ticks.

By default, offset value is 0.

The [position](#) will place the labels either inside or outside of the axis.

By default, labels will be placed **inside** the axis.

### APP.VUE

```

<template>
  <div id="app">
    <div class='wrapper'>
      <ejs-circulargauge>
        <e-axes>
          <e-axis :labelStyle ='labelStyle'>
        </e-axis>
        </e-axes>
      </ejs-circulargauge>
    </div>
  </div>
</template>
<script>
import Vue from 'vue';
import { CircularGaugePlugin } from '@syncfusion/ej2-vue-circulargauge';
Vue.use(CircularGaugePlugin);
export default {
  data: function () {
    return {
      labelStyle: {
        position: 'Outside',
        offset: 5
      }
    }
  }
};
</script>
<style>

```

```
.wrapper {
  max-width: 300px;
  margin: 0 auto;
}
</style>
```

{% previewsample "page.domainurl/code-snippet/circular-gauge/getting-started-cs18" %}

*Display the last label, even if it isn't in the visible range\*\**

If the last label is not in the visible range, it will be hidden by default. If you want to show the last label, set the `showLastLabel` property to `true` in the `axes` API of circular gauge.

#### APP.VUE

```
<template>
  <div id="app">
    <div class='wrapper'>
      <ejs-circulargauge>
        <e-axes>
          <e-axis showLastLabel="true" minimum= 0 maximum= 170
startAngle= 210 endAngle= 150>
        </e-axis>
      </e-axes>
    </ejs-circulargauge>
  </div>
</div>
</template>
<script>
import Vue from 'vue';
import { CircularGaugePlugin } from "@syncfusion/ej2-vue-circulargauge";
Vue.use(CircularGaugePlugin);
export default {
  data: function () {
    return {}
  }
};
</script>
<style>
  .wrapper {
    max-width: 300px;
    margin: 0 auto;
  }
</style>
```

{% previewsample "page.domainurl/code-snippet/circular-gauge/getting-started-cs19" %}

*Display the last label, even if it isn't in the visible range\*\**

If the last label is not in the visible range, it will be hidden by default. If you want to show the last label, set the `showLastLabel` property to `true` in the `axes` API of circular gauge.

#### APP.VUE

```
<template>
  <div id="app">
    <div class='wrapper'>
      <ejs-circulargauge >
```

```

    <e-axes>
      <e-axis showLastLabel="true" minimum= 0 maximum= 170 startAngle= 210
endAngle= 150 >
    </e-axis>
  </e-axes>
</ejs-circulargauge>
</div>
</div>
</template>
<script>
import Vue from 'vue';
import { CircularGaugePlugin } from "@syncfusion/ej2-vue-circulargauge";
Vue.use(CircularGaugePlugin);
export default {
  data: function () {
    return {}
  }
};
</script>
<style>
@import "../node_modules/@syncfusion/ej2-vue-circulargauge/styles/material.css";
.wrapper {
  max-width: 300px;
  margin: 0 auto;
}
</style>

```

{% previewsample "page.domainurl/code-snippet/circular-gauge/getting-started-cs20" %}

#### Auto Angle

Labels can be swept along the axis angle by enabling [autoAngle](#) property.

#### APP.VUE

```

<template>
  <div id="app">
    <div class='wrapper'>
      <ejs-circulargauge>
        <e-axes>
          <e-axis :labelStyle ='labelStyle'>
          </e-axis>
        </e-axes>
      </ejs-circulargauge>
    </div>
  </div>
</template>
<script>
import Vue from 'vue';
import { CircularGaugePlugin } from "@syncfusion/ej2-vue-circulargauge";
Vue.use(CircularGaugePlugin);
export default {
  data: function () {
    return {
      labelStyle: {
        autoAngle: true
      }
    }
  }
};

```

```

    }
  }
};
</script>
<style>
  .wrapper {
    max-width: 300px;
    margin: 0 auto;
  }
</style>

```

{% previewsample "page.domainurl/code-snippet/circular-gauge/getting-started-cs21" %}

### Smart Labels

When an axis makes a complete circle, then the first and last label of the axis will get overlap with each other.

In this scenario, you can either hide 1st or last label using [hiddenLabel](#) property.

When **hiddenLabel** value is **First**,

then the 1st label will be hidden and when the

**hiddenLabel** value is 'Last',

then the last label will be hidden.

### APP.VUE

```

<template>
  <div id="app">
    <div class='wrapper'>
      <ejs-circulargauge>
        <e-axes>
          <e-axis minimum= 0 maximum= 12 startAngle= 0 endAngle=
360 :labelStyle = 'labelStyle' :minorTicks= 'minorTicks' :majorTicks=
'majorTicks' >
            </e-axis>
          </e-axes>
        </ejs-circulargauge>
      </div>
    </div>
  </template>
<script>
import Vue from 'vue';
import { CircularGaugePlugin } from "@syncfusion/ej2-vue-circulargauge";
Vue.use(CircularGaugePlugin);
export default {
  data: function () {
    return {
      majorTicks: {
        interval: 1,
        position: 'Inside',
        height: 10
      },
      minorTicks: {
        interval: 0.2,

```

```

        position: 'Inside',
        height: 5
      },
      labelStyle: {
        position: 'Inside',
        hiddenLabel: 'First'
      }
    }
  }
};
</script>
<style>
  .wrapper {
    max-width: 300px;
    margin: 0 auto;
  }
</style>

```

{% previewsample "page.domainurl/code-snippet/circular-gauge/getting-started-cs22" %}

### Label Format

Axis labels can be formatted by using [format](#) property in [labelStyle](#) and its supports all globalize format.

### APP.VUE

```

<template>
  <div id="app">
    <div class='wrapper'>
      <ejs-circulargauge>
        <e-axes>
          <e-axis :labelStyle ='labelStyle'>
          </e-axis>
        </e-axes>
      </ejs-circulargauge>
    </div>
  </div>
</template>
<script>
import Vue from 'vue';
import { CircularGaugePlugin } from "@syncfusion/ej2-vue-circulargauge";
Vue.use(CircularGaugePlugin);
export default {
  data: function () {
    return {
      labelStyle: {
        format: 'p1'
      }
    }
  }
};
</script>
<style>
  .wrapper {
    max-width: 300px;
    margin: 0 auto;
  }

```

```
</style>
```

```
{% previewsample "page.domainurl/code-snippet/circular-gauge/getting-started-cs23" %}
```

The following table describes the result of applying some commonly used label formats on numeric values.

```
<!-- markdownlint-disable MD033 -->
```

Label Value	Label Format property value	Result	Description
1000	n1	1000.0	The Number is rounded to 1 decimal place
1000	n2	1000.00	The Number is rounded to 2 decimal place
1000	n3	1000.000	The Number is rounded to 3 decimal place
0.01	p1	1.0%	The Number is converted to percentage with 1 decimal place
0.01	p2	1.00%	The Number is converted to percentage with 2 decimal place
0.01	p3	1.000%	The Number is converted to percentage with 3 decimal place
1000	c1	\$1,000.0	The Currency symbol is appended to number and number is rounded to 1 decimal place
1000	c2	\$1,000.00	The Currency symbol is appended to number and number is rounded to 2 decimal place

### Custom Label Format

Axis labels support custom label format using placeholder like {value}°C, in which the value represent the axis label e.g 20°C.

### APP.VUE

```
<template>
  <div id="app">
    <div class='wrapper'>
      <ejs-circulargauge>
        <e-axes>
          <e-axis :labelStyle ='labelStyle'>
            </e-axis>
          </e-axes>
        </ejs-circulargauge>
      </div>
    </div>
  </template>
<script>
import Vue from 'vue';
import { CircularGaugePlugin } from "@syncfusion/ej2-vue-circulargauge";
Vue.use(CircularGaugePlugin);
export default {
```



```

    data: function () {
      return {
        labelStyle: {
          format: '{value}°C'
        }
      }
    }
  };
</script>
<style>
  .wrapper {
    max-width: 300px;
    margin: 0 auto;
  }
</style>

```

{% previewsample "page.domainurl/code-snippet/circular-gauge/getting-started-cs24" %}

#### *Hide intersecting axis labels*

When the axis labels overlap with each other, you can hide the intersected labels by setting the `hideIntersectingLabel` property to true in the axis.

#### **APP.VUE**

```

<template>
  <div id="app">
    <div class='wrapper'>
      <ejs-circulargauge>
        <e-axes>
          <e-axis :startAngle='startAngle'
:hideIntersectingLabel='hideIntersectingLabel' :endAngle='endAngle'
:majorTicks='majorTicks' :minorTicks='minorTicks' minimum=0 maximum=200>
        </e-axis>
      </e-axes>
    </ejs-circulargauge>
  </div>
</template>
<script>
import Vue from 'vue';
import { CircularGaugePlugin } from "@syncfusion/ej2-vue-circulargauge";
Vue.use(CircularGaugePlugin);
export default {
  data: function () {
    return {
      startAngle: 270,
      endAngle: 90,
      hideIntersectingLabel: true,
      majorTicks: {
        interval: 4
      },
      minorTicks: {
        interval: 2
      }
    }
  }
}

```

```
};
</script>
<style>
  .wrapper {
    max-width: 300px;
    margin: 0 auto;
  }
</style>
```

{% previewsample "page.domainurl/code-snippet/circular-gauge/getting-started-cs25" %}

### Minimum and Maximum

The [minimum](#) and [maximum](#) properties

enables you to customize the start and end values of an axis.

### APP.VUE

```
<template>
  <div id="app">
    <div class='wrapper'>
      <ejs-circulargauge>
        <e-axes>
          <e-axis minimum= 50 maximum= 250>
        </e-axis>
        </e-axes>
      </ejs-circulargauge>
    </div>
  </div>
</template>
<script>
import Vue from 'vue';
import { CircularGaugePlugin } from "@syncfusion/ej2-vue-circulargauge";
Vue.use(CircularGaugePlugin);
export default { };
</script>
<style>
  .wrapper {
    max-width: 300px;
    margin: 0 auto;
  }
</style>
```

{% previewsample "page.domainurl/code-snippet/circular-gauge/getting-started-cs26" %}

### Multiple Axes

In addition to the default axis, you can add n number of axis to a gauge.

Each axis will have its own ranges, pointers, annotations and customization options.

### APP.VUE

```
<template>
  <div id="app">
    <div class='wrapper'>
      <ejs-circulargauge :axes= 'axes'>
    </ejs-circulargauge>
```

```

        </div>
    </div>
</template>
<script>
import Vue from 'vue';
import { CircularGaugePlugin } from '@syncfusion/ej2-vue-circulargauge';
Vue.use(CircularGaugePlugin);
export default {
    data: function () {
        return {
            axes: [{
                majorTicks: {
                    interval: 10,
                    position: 'Inside',
                    height: 10,
                },
                pointers: [],
                minorTicks: {
                    interval: 5,
                    position: 'Inside',
                    height: 5,
                }
            },
            {
                pointers: [],
                majorTicks: {
                    interval: 10,
                    position: 'Inside',
                    height: 10,
                    color: '#27d5ff'
                },
                minorTicks: {
                    interval: 5,
                    position: 'Inside',
                    height: 5,
                    color: '#27d5ff'
                }
            }
        ]
    }
};
</script>
<style>
    .wrapper {
        max-width: 300px;
        margin: 0 auto;
    }
</style>

```

{% previewsample "page.domainurl/code-snippet/circular-gauge/getting-started-cs27" %}

### Gauge ranges in Vue Circular gauge component

You can categories certain interval on gauge axis using [ranges](#) property.

#### Start and End

Start and end value of a range in an axis can be customized by using [start](#) and [end](#) properties.

**APP.VUE**

```

<template>
<div id="app">
  <div class='wrapper'>
    <ejs-circulargauge>
      <e-axes>
        <e-axis :ranges='ranges'></e-axis>
      </e-axes>
    </ejs-circulargauge>
  </div>
</div>
</template>
<script>
import Vue from 'vue';
import { CircularGaugePlugin } from "@syncfusion/ej2-vue-circulargauge";
Vue.use(CircularGaugePlugin);
export default {
  data: function () {
    return {
      ranges: [{
        start: 40,
        end: 80
      }]
    }
  }
};
</script>
<style>
.wrapper {
  max-width: 300px;
  margin: 0 auto;
}
</style>

```

{% previewsample "page.domainurl/code-snippet/circular-gauge/getting-started-cs51" %}

**Customization**

Color and thickness of the range can be customized by using [color](#), [startWidth](#) and [endWidth](#) property.

**APP.VUE**

```

<template>
<div id="app">
  <div class='wrapper'>
    <ejs-circulargauge>
      <e-axes>
        <e-axis :ranges='ranges'></e-axis>
      </e-axes>
    </ejs-circulargauge>
  </div>
</div>
</template>
<script>
import Vue from 'vue';
import { CircularGaugePlugin } from "@syncfusion/ej2-vue-circulargauge";
Vue.use(CircularGaugePlugin);

```

```

export default {
  data: function () {
    return {
      ranges: [{
        start: 40,
        end: 80, endWidth: 15,
        startWidth: 15,
        color: '#ff5985'
      }]
    }
  }
};
</script>
<style>
.wrapper {
  max-width: 300px;
  margin: 0 auto;
}
</style>

```

{% previewsample "page.domainurl/code-snippet/circular-gauge/getting-started-cs52" %}

### Rounded Corner Ranges

The corners of the ranges can be rounded by specifying desired values to the `roundedCornerRadius` property.

### APP.VUE

```

<template>
<div id="app">
  <div class='wrapper'>
    <ejs-circulargauge >
      <e-axes>
        <e-axis minimum= 0 maximum= 120 :majorTicks= 'majorTicks' :ranges
        ='ranges'>
          <e-pointers>
            <e-pointer value=70 radius= '60%' :animation='animation' ></e-
            pointer>
          </e-pointers>
        </e-axis>
      </e-axes>
    </ejs-circulargauge>
  </div>
</div>
</template>
<script>
import Vue from 'vue';
import { CircularGaugePlugin } from "@syncfusion/ej2-vue-circulargauge";
Vue.use(CircularGaugePlugin);
export default {
  data: function () {
    return {
      animation: { enable: false },
      majorTicks: {
        height: 5
      },

```

```

    ranges: [{
      start: 0,
      end: 40,
      startWidth: 15,
      endWidth: 15,
      roundedCornerRadius: 10,
      radius: '110%'
    }, {
      start: 40,
      end: 80,
      startWidth: 15,
      endWidth: 15,
      roundedCornerRadius: 10,
      radius: '110%'
    }, {
      start: 80,
      end: 120,
      startWidth: 15,
      endWidth: 15,
      roundedCornerRadius: 10,
      radius: '110%'
    }
  ]
}
}
};
</script>
<style>
.wrapper {
  max-width: 300px;
  margin: 0 auto;
}
</style>

```

{% previewsample "page.domainurl/code-snippet/circular-gauge/getting-started-cs53" %}

### Radius

You can place the range inside or outside of the axis by using [radius](#) property. The radius of the range can takes value either in percentage or in pixels. By default, ranges take 100% of the axis radius.

### In Pixel

You can set the radius of the range in pixel as demonstrated below,

### APP.VUE

```

<template>
  <div id="app">
    <div class='wrapper'>
      <ejs-circulargauge>
        <e-axes>
          <e-axis :ranges='ranges'></e-axis>
        </e-axes>
      </ejs-circulargauge>
    </div>
  </div>

```

```

</template>
<script>
import Vue from 'vue';
import { CircularGaugePlugin } from '@syncfusion/ej2-vue-circulargauge';
Vue.use(CircularGaugePlugin);
export default {
  data: function () {
    return {
      ranges: [{
        start: 40,
        end: 80,
        radius: '100'
      }]
    }
  }
};
</script>
<style>
.wrapper {
  max-width: 300px;
  margin: 0 auto;
}
</style>

```

{% previewsample "page.domainurl/code-snippet/circular-gauge/getting-started-cs54" %}

#### *In Percentage*

By setting value in percentage, range gets its dimension with respect to its axis radius.

For example, when the radius is '50%', range renders to half of the axis radius.

#### **APP.VUE**

```

<template>
  <div id="app">
    <div class='wrapper'>
      <ejs-circulargauge>
        <e-axes>
          <e-axis minimum= 0 maximum= 100 :ranges='ranges'></e-axis>
        </e-axes>
      </ejs-circulargauge>
    </div>
  </div>
</template>
<script>
import Vue from 'vue';
import { CircularGaugePlugin } from '@syncfusion/ej2-vue-circulargauge';
Vue.use(CircularGaugePlugin);
export default {
  data: function () {
    return {
      ranges: [{
        start: 40,
        end: 80,
        radius: '50%'
      }]
    }
  }
}

```

```

    }
  };
</script>
<style>
  .wrapper {
    max-width: 300px;
    margin: 0 auto;
  }
</style>

```

{% previewsample "page.domainurl/code-snippet/circular-gauge/getting-started-cs55" %}

### Dragging Range

The ranges can be dragged on the axis line by clicking and dragging the same. To enable or disable the range drag, use the [enableRangeDrag](#) property.

### APP.VUE

```

<template>
  <div id="app">
    <div class='wrapper'>
      <ejs-circulargauge enableRangeDrag='true' height='250px' width='250px'>
        <e-axes>
          <e-axis :ranges='ranges'>
            <e-pointers>
              <e-pointer value=50></e-pointer>
            </e-pointers>
          </e-axis>
        </e-axes>
      </ejs-circulargauge>
    </div>
  </div>
</template>
<script>
import Vue from 'vue';
import { CircularGaugePlugin } from '@syncfusion/ej2-vue-circulargauge';
Vue.use(CircularGaugePlugin);
export default {
  data: function () {
    return {
      ranges: [{
        start: 0,
        end: 100,
        startWidth: 8, endWidth: 8,
        radius: '108%',
        color: '#30B32D'
      }]
    }
  }
};
</script>
<style>
  .wrapper {
    max-width: 300px;
    margin: 0 auto;
  }

```



```
</style>
```

```
{% previewsample "page.domainurl/code-snippet/circular-gauge/getting-started-cs56" %}
```

### Multiple Ranges

You can add multiple ranges to an axis with the above customization as demonstrated below.

Note: You can set the range color to axis ticks and labels by enabling `useRangeColor` property in [majorTicks](#),

[minorTicks](#) and [labelStyle](#) object.

### APP.VUE

```
<template>
  <div id="app">
    <div class='wrapper'>
      <ejs-circulargauge >
        <e-axes>
          <e-axis :majorTicks= 'majorTicks' :minorTicks= 'minorTicks'
:labelStyle= 'labelStyle' :ranges='ranges'></e-axis>
        </e-axes>
      </ejs-circulargauge>
    </div>
  </div>
</template>
<script>
import Vue from 'vue';
import { CircularGaugePlugin } from "@syncfusion/ej2-vue-circulargauge";
Vue.use(CircularGaugePlugin);
export default {
  data: function () {
    return {
      majorTicks: {
        useRangeColor: true
      },
      minorTicks: {
        useRangeColor: true
      },
      labelStyle: {
        useRangeColor: true
      },
      ranges: [{
        start: 0,
        end: 25,
        radius: '108%'
      }, {
        start: 25,
        end: 50,
        radius: '70%'
      }, {
        start: 50,
        end: 75,
        radius: '70%'
      }, {
        start: 75,
        end: 100,
```

```

        radius: '108%'
      }
    }
  }
};
</script>
<style>
  .wrapper {
    max-width: 300px;
    margin: 0 auto;
  }
</style>

```

{% previewsample "page.domainurl/code-snippet/circular-gauge/getting-started-cs57" %}

### Gradient Color

Gradient support allows to add multiple colors in the ranges and pointers of the circular gauge. The following gradient types are supported in the circular gauge.

- Linear Gradient
- Radial Gradient

#### Linear Gradient

Using linear gradient, colors will be applied in a linear progression. The start value of the linear gradient can be set using the [startValue](#) property. The end value of the linear gradient will be set using the [endValue](#) property. The color stop values such as color, opacity and offset are set using [colorStop](#) property.

To apply linear gradient to the range, follow the below code sample.

#### APP.VUE

```

<template>
  <div id="app">
    <div class="wrapper">
      <ejs-circulargauge style='display:block' align='center' :title='title'
      :titleStyle='titleStyle' :centerY='centerY'>
        <e-axes>
          <e-axis :radius='gaugeRadius' :startAngle='startAngle' minimum=0
          maximum=14 :endAngle='endAngle' :majorTicks='majorTicks'
          :lineStyle='lineStyle' :minorTicks='minorTicks' :labelStyle='labelStyle'
          :annotations='annotations' :ranges='ranges'>
            <e-pointers>
              <e-pointer :type='type1' :value='value1' :radius='pointerRadius1'
              :markerShape='markerShape1' :imageUrl='imageUrl1'
              :markerWidth='markerWidth1' :markerHeight='markerHeight1'
              :animation='animation1'>
            </e-pointer>
              <e-pointer :type='type2' :value='value2' :radius='pointerRadius2'
              :markerShape='markerShape2' :imageUrl='imageUrl2'
              :markerWidth='markerWidth2' :markerHeight='markerHeight2'
              :animation='animation2'>
            </e-pointer>
              <e-pointer :type='type3' :value='value3' :radius='pointerRadius3'
              :markerShape='markerShape3' :imageUrl='imageUrl3'

```

```

:markerWidth='markerWidth3' :markerHeight='markerHeight3'
:animation='animation3'>
  </e-pointer>
  <e-pointer :type='type4' :value='value4' :radius='pointerRadius4'
:markerShape='markerShape4' :imageUrl='imageUrl4'
:markerWidth='markerWidth4' :markerHeight='markerHeight4'
:animation='animation4'>
  </e-pointer>
  <e-pointer :type='type5' :value='value5' :radius='pointerRadius5'
:markerShape='markerShape5' :imageUrl='imageUrl5'
:markerWidth='markerWidth5' :markerHeight='markerHeight5'
:animation='animation5'>
  </e-pointer>
  <e-pointer :type='type6' :value='value6' :radius='pointerRadius6'
:markerShape='markerShape6' :imageUrl='imageUrl6'
:markerWidth='markerWidth6' :markerHeight='markerHeight6'
:animation='animation6'>
  </e-pointer>
  <e-pointer :type='type7' :value='value7' :radius='pointerRadius7'
:markerShape='markerShape7' :imageUrl='imageUrl7'
:markerWidth='markerWidth7' :markerHeight='markerHeight7'
:animation='animation7'>
  </e-pointer>
</e-pointers>
</e-axis>
</e-axes>
</ejs-circulargauge>
</div>
</div>
</template>
<script>
import Vue from 'vue';
import { CircularGaugePlugin, Gradient, Annotations } from "@syncfusion/ej2-
vue-circulargauge";
Vue.use(CircularGaugePlugin);
export default {
  data: function () {
    var rangeLinearGradient = {
      startValue: '0%', endValue: '100%',
      colorStop: [
        { color: '#9E40DC', offset: '0%', opacity: 0.9 },
        { color: '#E63B86', offset: '70%', opacity: 0.9 } ]
    }
    return {
      title: 'Short Put Distance',
      titleStyle: {
        size: '18px'
      },
    },
    centerY: '57%',
    annotations: [{
      content: '12 M', radius: '108%', angle: 98, zIndex: '1'
    }, {
      content: '11 M', radius: '80%', angle: 81, zIndex: '1'
    }, {
      content: '10 M', radius: '50%', angle: 69, zIndex: '1'
    }, {
      content: 'Doe', radius: '108%', angle: 190, zIndex: '1'

```

```

    }, {
      content: 'Almada', radius: '80%', angle: 185, zIndex: '1'
    }, {
      content: 'John', radius: '50%', angle: 180, zIndex: '1'
    }],
    lineStyle: {
      width: 0
    },
    gaugeRadius: '90%',
    labelStyle: {
      font: {
        size: '0px'
      }
    },
    majorTicks: {
      width: 0,
    },
    minorTicks: {
      width: 0,
    },
    startAngle: 200, endAngle: 130,
    ranges: [{
      start: 0, end: 12, radius: '115%',
      startWidth: 25, endWidth: 25,
      linearGradient : rangeLinearGradient
    }, {
      start: 0, end: 11, radius: '85%',
      startWidth: 25, endWidth: 25,
      linearGradient : rangeLinearGradient
    }, {
      start: 0, end: 10, radius: '55%',
      startWidth: 25, endWidth: 25,
      linearGradient : rangeLinearGradient
    }],
    type1: 'Marker', value1: 12, markerShape1: 'Image',
    imageUrl1: 'https://ej2.syncfusion.com/vue/demos/src/circular-
gauge/images/foot-ball.png',
    pointerRadius1: '108%', markerWidth1: 28, markerHeight1: 28,
    animation1: { duration: 1500 },
    type2: 'Marker', value2: 11, markerShape2: 'Image',
    imageUrl2: 'https://ej2.syncfusion.com/vue/demos/src/circular-
gauge/images/basket-ball.png',
    pointerRadius2: '78%', markerWidth2: 28, markerHeight2: 28,
    animation2: { duration: 1200 },
    type3: 'Marker', value3: 10, markerShape3: 'Image',
    imageUrl3: 'https://ej2.syncfusion.com/vue/demos/src/circular-
gauge/images/golf-ball.png',
    pointerRadius3: '48%', markerWidth3: 28, markerHeight3: 28,
    animation3: { duration: 900 },
    type4: 'Marker', value4: 12, markerShape4: 'Image',
    imageUrl4: 'https://ej2.syncfusion.com/vue/demos/src/circular-
gauge/images/athletics.png',
    pointerRadius4: '0%', markerWidth4: 90, markerHeight4: 90,
    animation4: { duration: 0 },
    type5: 'Marker', value5: 0.1, markerShape5: 'Image',
    imageUrl5: 'https://ej2.syncfusion.com/vue/demos/src/circular-
gauge/images/girl.png',

```

```

        pointerRadius5: '108%', markerWidth5: 28, markerHeight5: 28,
        animation5: { duration: 1500 },
        type6: 'Marker', value6: 0.1, markerShape6: 'Image',
        imageUrl6: 'https://ej2.syncfusion.com/vue/demos/src/circular-
        gauge/images/man-one.png',
        pointerRadius6: '78%', markerWidth6: 28, markerHeight6: 28,
        animation6: { duration: 1500 },
        type7: 'Marker', value7: 0.1, markerShape7: 'Image',
        imageUrl7: 'https://ej2.syncfusion.com/vue/demos/src/circular-
        gauge/images/man-two.png',
        pointerRadius7: '48%', markerWidth7: 28, markerHeight7: 28,
        animation7: { duration: 1500 }
    },
    provide: {
        circulargauge: [Gradient, Annotations]
    }
};
</script>
<style>
    .wrapper {
        max-width: 500px;
        margin: 0 auto;
    }
</style>

```

{% previewsample "page.domainurl/code-snippet/circular-gauge/getting-started-cs58" %}

### Radial Gradient

Using radial gradient, colors will be applied in circular progression. The inner circle position of the radial gradient will be set using the [innerPosition](#) property. The outer circle position of the radial gradient can be set using the [outerPosition](#) property. The color stop values such as color, opacity and offset are set using [colorStop](#) property.

To apply radial gradient to the range, follow the below code sample.

### APP.VUE

```

<template>
  <div id="app">
    <div class="wrapper">
      <ejs-circulargauge style='display:block' align='center' :title='title'
      :titleStyle='titleStyle' :centerY='centerY'>
        <e-axes>
          <e-axis :radius='gaugeRadius' :startAngle='startAngle' minimum=0
          maximum=14 :endAngle='endAngle' :majorTicks='majorTicks'
          :lineStyle='lineStyle' :minorTicks='minorTicks' :labelStyle='labelStyle'
          :annotations='annotations' :ranges='ranges'>
            <e-pointers>
              <e-pointer :type='type1' :value='value1' :radius='pointerRadius1'
              :markerShape='markerShape1' :imageUrl='imageUrl1'
              :markerWidth='markerWidth1' :markerHeight='markerHeight1'
              :animation='animation1'></e-pointer>
              <e-pointer :type='type2' :value='value2' :radius='pointerRadius2'
              :markerShape='markerShape2' :imageUrl='imageUrl2'

```

```

:markerWidth='markerWidth2' :markerHeight='markerHeight2'
:animation='animation2'></e-pointer>
  <e-pointer :type='type3' :value='value3' :radius='pointerRadius3'
:markerShape='markerShape3' :imageUrl='imageUrl3'
:markerWidth='markerWidth3' :markerHeight='markerHeight3'
:animation='animation3'></e-pointer>
  <e-pointer :type='type4' :value='value4' :radius='pointerRadius4'
:markerShape='markerShape4' :imageUrl='imageUrl4'
:markerWidth='markerWidth4' :markerHeight='markerHeight4'
:animation='animation4'></e-pointer>
  <e-pointer :type='type5' :value='value5' :radius='pointerRadius5'
:markerShape='markerShape5' :imageUrl='imageUrl5'
:markerWidth='markerWidth5' :markerHeight='markerHeight5'
:animation='animation5'></e-pointer>
  <e-pointer :type='type6' :value='value6' :radius='pointerRadius6'
:markerShape='markerShape6' :imageUrl='imageUrl6'
:markerWidth='markerWidth6' :markerHeight='markerHeight6'
:animation='animation6'></e-pointer>
  <e-pointer :type='type7' :value='value7' :radius='pointerRadius7'
:markerShape='markerShape7' :imageUrl='imageUrl7'
:markerWidth='markerWidth7' :markerHeight='markerHeight7'
:animation='animation7'></e-pointer>
</e-pointers>
</e-axis>
</e-axes>
</ejs-circulargauge>
</div>
</div>
</template>
<script>
import Vue from 'vue';
import { CircularGaugePlugin, Gradient, Annotations } from "@syncfusion/ej2-vue-circulargauge";
Vue.use(CircularGaugePlugin);
export default {
  data: function () {
    var rangeRadialGradient = {
      radius: '50%', innerPosition: { x: '50%', y: '50%' },
      outerPosition: { x: '50%', y: '50%' },
      colorStop: [
        { color: '#9E40DC', offset: '90%', opacity: 0.9 },
        { color: '#E63B86', offset: '160%', opacity: 0.9 }
      ]
    }
    return {
      title: 'Short Put Distance',
      titleStyle: {
        size: '18px'
      },
      centerY: '57%',
      annotations: [{
        content: '12 M', radius: '108%', angle: 98, zIndex: '1'
      }, {
        content: '11 M', radius: '80%', angle: 81, zIndex: '1'
      }, {
        content: '10 M', radius: '50%', angle: 69, zIndex: '1'
      }, {
        content: 'Doe', radius: '108%', angle: 190, zIndex: '1'
      }
    ]
  }
}

```

```

    }, {
      content: 'Almaida', radius: '80%', angle: 185, zIndex: '1'
    }, {
      content: 'John', radius: '50%', angle: 180, zIndex: '1'
    }],
    lineStyle: {
      width: 0
    },
    gaugeRadius: '90%',
    labelStyle: {
      font: {
        size: '0px'
      }
    },
    majorTicks: {
      width: 0
    },
    minorTicks: {
      width: 0
    },
    startAngle: 200, endAngle: 130,
    ranges: [{
      start: 0, end: 12, radius: '115%',
      startWidth: 25, endWidth: 25,
      radialGradient: rangeRadialGradient
    }, {
      start: 0, end: 11, radius: '85%',
      startWidth: 25, endWidth: 25,
      radialGradient: rangeRadialGradient
    }, {
      start: 0, end: 10, radius: '55%',
      startWidth: 25, endWidth: 25,
      radialGradient: rangeRadialGradient
    }],
    type1: 'Marker', value1: 12, markerShape1: 'Image',
    imageUrl1: 'https://ej2.syncfusion.com/vue/demos/src/circular-gauge/images/foot-ball.png',
    pointerRadius1: '108%', markerWidth1: 28, markerHeight1: 28,
    animation1: { duration: 1500 },
    type2: 'Marker', value2: 11, markerShape2: 'Image',
    imageUrl2: 'https://ej2.syncfusion.com/vue/demos/src/circular-gauge/images/basket-ball.png',
    pointerRadius2: '78%', markerWidth2: 28, markerHeight2: 28,
    animation2: { duration: 1200 },
    type3: 'Marker', value3: 10, markerShape3: 'Image',
    imageUrl3: 'https://ej2.syncfusion.com/vue/demos/src/circular-gauge/images/golf-ball.png',
    pointerRadius3: '48%', markerWidth3: 28, markerHeight3: 28,
    animation3: { duration: 900 },
    type4: 'Marker', value4: 12, markerShape4: 'Image',
    imageUrl4: 'https://ej2.syncfusion.com/vue/demos/src/circular-gauge/images/athletics.png',
    pointerRadius4: '0%', markerWidth4: 90, markerHeight4: 90,
    animation4: { duration: 0 },
    type5: 'Marker', value5: 0.1, markerShape5: 'Image',
    imageUrl5: 'https://ej2.syncfusion.com/vue/demos/src/circular-gauge/images/girl.png',

```

```

        pointerRadius5: '108%', markerWidth5: 28, markerHeight5: 28,
        animation5: { duration: 1500 },
        type6: 'Marker', value6: 0.1, markerShape6: 'Image',
        imageUrl6: 'https://ej2.syncfusion.com/vue/demos/src/circular-
gauge/images/man-one.png',
        pointerRadius6: '78%', markerWidth6: 28, markerHeight6: 28,
        animation6: { duration: 1500 },
        type7: 'Marker', value7: 0.1, markerShape7: 'Image',
        imageUrl7: 'https://ej2.syncfusion.com/vue/demos/src/circular-
gauge/images/man-two.png',
        pointerRadius7: '48%', markerWidth7: 28, markerHeight7: 28,
        animation7: { duration: 1500 }
    },
    provide: {
        circulargauge: [Gradient, Annotations]
    }
};
</script>
<style>
    .wrapper {
        max-width: 500px;
        margin: 0 auto;
    }
</style>

```

{% previewsample "page.domainurl/code-snippet/circular-gauge/getting-started-cs59" %}

See also

- [Tooltip for Ranges](#)

## Gauge pointers in Vue Circular gauge component

Pointers are used to indicate values on the axis. Value of the pointer can be modified using the [value](#) property.

### APP.VUE

```

<template>
  <div id="app">
    <div class='wrapper'>
      <ejs-circulargauge>
        <e-axes>
          <e-axis>
            <e-pointers>
              <e-pointer value=90 ></e-pointer>
            </e-pointers>
          </e-axis>
        </e-axes>
      </ejs-circulargauge>
    </div>
  </div>
</template>
<script>
import Vue from 'vue';

```



```
import { CircularGaugePlugin } from "@syncfusion/ej2-vue-circulargauge";
Vue.use(CircularGaugePlugin);
export default { };
</script>
<style>
    .wrapper {
        max-width: 300px;
        margin: 0 auto;
    }
</style>
```

{% previewsample "page.domainurl/code-snippet/circular-gauge/getting-started-cs35" %}

### Needle Pointers

A needle pointer contains three parts, a needle, a cap / knob and a tail. The length of the needle can be customized by using [radius](#) property. The length of the tail can be customized by using [length](#) property. The radius of the cap can be customized by using [radius](#) in cap object. The needle and tail length takes value either in [percentage](#) or [pixel](#).

### APP.VUE

```
<template>
    <div id="app">
        <div class='wrapper'>
            <ejs-circulargauge>
                <e-axes>
                    <e-axis>
                        <e-pointers>
                            <e-pointer value=90 radius= '50%' :needleTail=
'needleTail' :cap = 'cap'></e-pointer>
                        </e-pointers>
                    </e-axis>
                </e-axes>
            </ejs-circulargauge>
        </div>
    </div>
</template>
<script>
import Vue from 'vue';
import { CircularGaugePlugin } from "@syncfusion/ej2-vue-circulargauge";
Vue.use(CircularGaugePlugin);
export default {
    data: function () {
        return {
            cap: {
                radius: 10
            },
            needleTail: {
                length: '25%'
            }
        }
    }
}
```

```
};
</script>
<style>
  .wrapper {
    max-width: 300px;
    margin: 0 auto;
  }
</style>
```

{% previewsample "page.domainurl/code-snippet/circular-gauge/getting-started-cs36" %}

### Customization

Needle color and width can be customized by using [color](#) and [pointerWidth](#) property.

Cap and tails can be customized by using [cap](#) and

[needleTail](#) object.

### APP.VUE

```
<template>
  <div id="app">
    <div class='wrapper'>
      <ejs-circulargauge>
        <e-axes>
          <e-axis>
            <e-pointers>
              <e-pointer value=90 color= '#007DD1'
pointerWidth= 25 radius= '50%' :needleTail= 'needleTail' :cap = 'cap'></e-
pointer>
            </e-pointers>
          </e-axis>
        </e-axes>
      </ejs-circulargauge>
    </div>
  </div>
</template>
<script>
import Vue from 'vue';
import { CircularGaugePlugin } from "@syncfusion/ej2-vue-circulargauge";
Vue.use(CircularGaugePlugin);
export default {
  data: function () {
    return {
      cap: {
        radius: 15,
        color: 'white',
        border: {
          color: '#007DD1',
          width: 5
        }
      },
      needleTail: {
        length: '22%',
        color: '#007DD1'
      }
    }
  }
}
```

```

    }
  };
</script>
<style>
  .wrapper {
    max-width: 300px;
    margin: 0 auto;
  }
</style>

```

{% previewsample "page.domainurl/code-snippet/circular-gauge/getting-started-cs37" %}

The appearance of the needle pointer can be customized by using [needleStartWidth](#) and [needleEndWidth](#).

### APP.VUE

```

<template>
  <div class='wrapper'>
    <div id="app">
      <ejs-circulargauge>
        <e-axes>
          <e-axis :radius='gauge3Radius' :startAngle='startAngle'
:endAngle='endAngle' :lineStyle= 'lineStyle' :labelStyle= 'labelStyle'
          :majorTicks= 'majorTicks' :minorTicks= 'minorTicks'
minimum=0 maximum=100
          :annotations='annotations'>
            <e-pointers>
              <e-pointer value=70 radius= '80%' color='green'
:pointerWidth='pointerWidth' :needleStartWidth='needleStartWidth'
              :needleEndWidth='needleEndWidth' :cap= 'cap'
:needleTail= 'needleTail' :animation= 'animation'></e-pointer>
            </e-pointers>
          </e-axis>
        </e-axes>
      </ejs-circulargauge>
    </div>
  </div>
</template>
<script>
import Vue from 'vue';
import { CircularGaugePlugin, Annotations } from "@syncfusion/ej2-vue-
circulargauge";
let contentVue = Vue.component("contentTemplate", {
  template: '<div style="color:#757575; font-family:Roboto; font-
size:14px;padding-top: 26px">Customized Needle</div>',
  data() {
    return {
      data: {}
    };
  }
});
let contentTemplate = function() {
  return { template: contentVue };
};
Vue.use(CircularGaugePlugin);

```

```
export default {
  data: function () {
    return {
      annotations:[{
        content:contentTemplate,
        zIndex: '1'
      }],
      gauge3Radius: '90%',
      startAngle:270,
      endAngle:90,
      lineStyle: {
        width: 3,
        color: '#1E7145'
      },
      labelStyle: {
        position:'Outside',
        font: {
          color: '#1E7145',
          size: '0px'
        }
      },
      majorTicks: {
        interval: 100,
        height: 0,
        width: 1
      },
      minorTicks: {
        height: 0,
        width: 0
      },
      cap: {
        radius: 8,
        color: 'green'
      },
      needleTail: {
        length:'0%'
      },
      animation:{
        enable:'true',
        duration:1000
      },
      pointerWidth: 2,
      needleStartWidth: 4,
      needleEndWidth: 4
    }
  },
  provide: {
    circulargauge: [Annotations]
  },
};
</script>
<style>
  .wrapper {
    max-width: 300px;
    margin: 0 auto;
  }
</style>
```

{% previewsample "page.domainurl/code-snippet/circular-gauge/getting-started-cs38" %}

### RangeBar Pointer

RangeBar pointer is like ranges in an axis, that can be placed on gauge to mark the pointer value.

RangeBar starts from the beginning of the gauge and ends at the pointer value.

#### APP.VUE

```
<template>
  <div id="app">
    <div class='wrapper'>
      <ejs-circulargauge>
        <e-axes>
          <e-axis>
            <e-pointers>
              <e-pointer value=50 type= 'RangeBar' radius=
'60%'></e-pointer>
            </e-pointers>
          </e-axis>
        </e-axes>
      </ejs-circulargauge>
    </div>
  </div>
</template>
<script>
import Vue from 'vue';
import { CircularGaugePlugin } from "@syncfusion/ej2-vue-circulargauge";
Vue.use(CircularGaugePlugin);
export default { };
</script>
<style>
.wrapper {
  max-width: 300px;
  margin: 0 auto;
}
</style>
```

{% previewsample "page.domainurl/code-snippet/circular-gauge/getting-started-cs39" %}

### RangeBar Pointer with rounded corners

The corners of the range bar pointer can be rounded by specifying desired values to the `roundedCornerRadius` property.

#### APP.VUE

```
<template>
<div id="app">
  <div class='wrapper'>
    <ejs-circulargauge>
      <e-axes>
        <e-axis>
          <e-pointers>
            <e-pointer value=60 type= 'RangeBar' radius= '60%'
roundedCornerRadius= 10 :animation = 'animation'></e-pointer>
          </e-pointers>
        </e-axis>
      </e-axes>
    </ejs-circulargauge>
  </div>
</div>
</template>
```

```

        </e-pointers>
      </e-axis>
    </e-axes>
  </ejs-circulargauge>
</div>
</div>
</template>
<script>
import Vue from 'vue';
import { CircularGaugePlugin } from '@syncfusion/ej2-vue-circulargauge';
Vue.use(CircularGaugePlugin);
export default {
  data: function () {
    return {
      animation: { enable: false }
    }
  }
};
</script>
<style>
.wrapper {
  max-width: 300px;
  margin: 0 auto;
}
</style>

```

{% previewsample "page.domainurl/code-snippet/circular-gauge/getting-started-cs40" %}

#### [RangeBar Customization](#)

RangeBar can be customized in terms of color, border and thickness by using

[color](#), [border](#) and [pointerWidth](#) property.

#### **APP.VUE**

```

<template>
<div id="app">
  <div class='wrapper'>
    <ejs-circulargauge>
      <e-axes>
        <e-axis>
          <e-pointers>
            <e-pointer value=50 type= 'RangeBar' radius= '60%'
color= '#007DD1' pointerWidth: 15 :border= 'border'></e-pointer>
          </e-pointers>
        </e-axis>
      </e-axes>
    </ejs-circulargauge>
  </div>
</div>
</template>
<script>
import Vue from 'vue';
import { CircularGaugePlugin } from '@syncfusion/ej2-vue-circulargauge';
Vue.use(CircularGaugePlugin);
export default {
  data: function () {

```

```

        return {
            border: {
                color: 'grey',
                width: 2
            }
        }
    };
</script>
<style>
.wrapper {
    max-width: 300px;
    margin: 0 auto;
}
</style>

```

{% previewsample "page.domainurl/code-snippet/circular-gauge/getting-started-cs41" %}

### Marker Pointer

Different type of marker shape can be used to mark the pointer value in axis. You can change the marker shape using [markerShape](#) property in pointer. Gauge supports the below marker shape.

- Circle
- Rectangle
- Triangle
- InvertedTriangle
- Diamond

We can use image instead of rendering marker shape to denote the pointer value. It can be achieved by setting [markerShape](#) to Image and assigning image path to [imageUrl](#) in pointer.

### APP.VUE

```

<template>
  <div id="app">
    <div class='wrapper'>
      <ejs-circulargauge>
        <e-axes>
          <e-axis :pointers ='pointers'>
          </e-axis>
        </e-axes>
      </ejs-circulargauge>
    </div>
  </div>
</template>
<script>
import Vue from 'vue';
import { CircularGaugePlugin } from '@syncfusion/ej2-vue-circulargauge';
Vue.use(CircularGaugePlugin);
export default {
  data: function () {
    return {
      pointers: [{
        value: 90,

```

```

        type: 'Marker',
        markerShape: 'InvertedTriangle',
        radius: '100%',
        markerHeight: 15,
        markerWidth: 15
      }
    ]
  }
};
</script>
<style>
.wrapper {
  max-width: 300px;
  margin: 0 auto;
}
</style>

```

{% previewsample "page.domainurl/code-snippet/circular-gauge/getting-started-cs42" %}

### Marker Customization

The marker can be customized in terms of color, border, width and height by using

[color](#),

[border](#),

[markerWidth](#) and

[markerHeight](#) property in

[pointer](#).

### APP.VUE

```

<template>
  <div id="app">
    <div class='wrapper'>
      <ejs-circulargauge>
        <e-axes>
          <e-axis>
            <e-pointers>
              <e-pointer value=90 :border= 'border' type=
'Marker' markerWidth= 15 markerHeight= 15 markerShape= 'Triangle' radius=
'100%' color= 'white' ></e-pointer>
            </e-pointers>
          </e-axis>
        </e-axes>
      </ejs-circulargauge>
    </div>
  </div>
</template>
<script>
import Vue from 'vue';
import { CircularGaugePlugin } from "@syncfusion/ej2-vue-circulargauge";
Vue.use(CircularGaugePlugin);
export default {
  data: function () {
    return {

```



```

        border: {
            color: '#007DD1',
            width: 2
        }
    }
}
};
</script>
<style>
.wrapper {
    max-width: 300px;
    margin: 0 auto;
}
</style>

```

{% previewsample "page.domainurl/code-snippet/circular-gauge/getting-started-cs43" %}

### Dragging pointer

The pointers can be dragged over the axis line by clicking and dragging the same. To enable or disable the pointer drag, use the [enablePointerDrag](#) property.

### APP.VUE

```

<template>
  <div id="app">
    <div class='wrapper'>
      <ejs-circulargauge enablePointerDrag='true' height='250px'
width='250px'>
        <e-axes>
          <e-axis>
            <e-pointers>
              <e-pointer value=50></e-pointer>
            </e-pointers>
          </e-axis>
        </e-axes>
      </ejs-circulargauge>
    </div>
  </div>
</template>
<script>
import Vue from 'vue';
import { CircularGaugePlugin } from "@syncfusion/ej2-vue-circulargauge";
Vue.use(CircularGaugePlugin);
export default {
  data: function () {
    return {
    }
  }
};
</script>
<style>
.wrapper {
    max-width: 300px;
    margin: 0 auto;
}
</style>

```

{% previewsample "page.domainurl/code-snippet/circular-gauge/getting-started-cs44" %}

### Multiple Pointers

In addition to the default pointer, you can add n number of pointer to an axis by using **pointers** property.

#### APP.VUE

```
<template>
  <div id="app">
    <div class='wrapper'>
      <ejs-circulargauge>
        <e-axes>
          <e-axis>
            <e-pointers>
              <e-pointer value=90 type= 'Marker' markerWidth=
15 markerHeight= 15 markerShape= 'InvertedTriangle' radius= '100%' ></e-
pointer>
              <e-pointer value=90 type= 'RangeBar'
markerWidth= 10 radius= '60%' ></e-pointer>
              <e-pointer value=90 pointerWidth= 25 :cap=
'cap' :needleTail= 'needleTail' radius= '60%' ></e-pointer>
            </e-pointers>
          </e-axis>
        </e-axes>
      </ejs-circulargauge>
    </div>
  </div>
</template>
<script>
import Vue from 'vue';
import { CircularGaugePlugin } from "@syncfusion/ej2-vue-circulargauge";
Vue.use(CircularGaugePlugin);
export default {
  data: function () {
    return {
      cap: {
        radius: 15,
        border: {
          width: 5
        }
      },
      needleTail: {
        length: '22%',
      }
    }
  }
};
</script>
<style>
.wrapper {
  max-width: 300px;
  margin: 0 auto;
}
</style>
```

{% previewsample "page.domainurl/code-snippet/circular-gauge/getting-started-cs45" %}

### Animation

Pointer will get animate on loading the gauge, this can be handled by using

[animation](#) property in pointer.

The [enable](#) property in animation allows you to enable or disable the animation.

The [duration](#) property specify the duration of the animation in milliseconds.

### APP.VUE

```
<template>
  <div id="app">
    <div class='wrapper'>
      <ejs-circulargauge>
        <e-axes>
          <e-axis>
            <e-pointers>
              <e-pointer value=90 :animation= 'animation'
            >>/e-pointer>
            </e-pointers>
          </e-axis>
        </e-axes>
      </ejs-circulargauge>
    </div>
  </div>
</template>
<script>
import Vue from 'vue';
import { CircularGaugePlugin } from "@syncfusion/ej2-vue-circulargauge";
Vue.use(CircularGaugePlugin);
export default {
  data: function () {
    return {
      animation: {
        enable: true,
        duration: 1500
      }
    }
  }
};
</script>
<style>
.wrapper {
  max-width: 300px;
  margin: 0 auto;
}
</style>
```

{% previewsample "page.domainurl/code-snippet/circular-gauge/getting-started-cs46" %}

### Gradient Color

Gradient support allows to add multiple colors in the ranges and pointers of the circular gauge. The following gradient types are supported in the circular gauge.

- Linear Gradient
- Radial Gradient

#### Linear Gradient

Using linear gradient, colors will be applied in a linear progression. The start value of the linear gradient can be set using the [startValue](#) property. The end value of the linear gradient will be set using the [endValue](#) property. The color stop values such as color, opacity and offset are set using [colorStop](#) property.

The linear gradient can be applied to all pointer types like marker, range bar and needle. To do so, follow the below code sample.

#### APP.VUE

```
<template>
  <div id="app">
    <div class='wrapper'>
      <ejs-circulargauge>
        <e-axes>
          <e-axis :startAngle = 270 :endAngle= 90 :lineStyle=
'lineStyle' :labelStyle= 'labelStyle' :majorTicks= 'majorTicks' :minorTicks=
'minorTicks' :radius= 'radius' :minimum= 0 :maximum= 100
:pointers='pointers' >
          </e-axis>
        </e-axes>
      </ejs-circulargauge>
    </div>
  </div>
</template>
<script>
import Vue from 'vue';
import { CircularGaugePlugin, Gradient } from "@syncfusion/ej2-vue-
circulargauge";
Vue.use(CircularGaugePlugin);
export default {
  data: function () {
    var pointerLinearGradient = {
      startValue: '0%',
      endValue: '100%',
      colorStop: [
        { color: '#FEF3F9', offset: '0%', opacity: 0.9 },
        { color: '#E63B86', offset: '70%', opacity: 0.9 } ]
    }
    return {
      lineStyle: { width: 3, color: '#E63B86' },
      labelStyle: {
        font: { size: '0px' }
      }, majorTicks: {
        height: 0
      }, minorTicks: {
        height: 0
      }
    }
  }
}
```

```

    },
    radius: '90%',
    pointers: [{
      radius: '80%',
      value: 80,
      animation: { enable: true, duration: 1000 },
      pointerWidth: 10,
      linearGradient: pointerLinearGradient,
      cap: {
        radius: 8,
        color: 'white',
        border: {
          color: '#E63B86',
          width: 1
        }
      }
    },
    needleTail: {
      length: '20%',
      linearGradient: pointerLinearGradient
    }
  ], {
    radius: '60%', value: 40,
    animation: { duration: 1000 },
    pointerWidth: 10,
    linearGradient: pointerLinearGradient,
    cap: {
      radius: 8, color: 'white',
      border: { color: '#E63B86', width: 1 }
    },
    needleTail: {
      length: '20%',
      linearGradient: pointerLinearGradient
    }
  }
  ]
}
},
provide: {
  circulargauge: [Gradient]
}
};
</script>
<style>
  .wrapper {
    max-width: 500px;
    margin: 0 auto;
  }
</style>

```

{% previewsample "page.domainurl/code-snippet/circular-gauge/getting-started-cs47" %}

### Radial Gradient

Using radial gradient, colors will be applied in circular progression. The inner circle position of the radial gradient will be set using the [innerPosition](#) property. The outer circle position of the radial gradient can be set using the [outerPosition](#) property. The color stop values such as color, opacity and offset are set using [colorStop](#) property.

The radial gradient can be applied to all pointer types like marker, range bar and needle. To do so, follow the below code sample.

#### APP.VUE

```
<template>
  <div id="app">
    <div class='wrapper'>
      <ejs-circulargauge>
        <e-axes>
          <e-axis :pointers= 'pointers' :startAngle = 270
: endAngle= 90 :lineStyle= 'lineStyle' :labelStyle= 'labelStyle' :majorTicks=
'majorTicks' :minorTicks= 'minorTicks' :radius= 'radius' :minimum= 0
:maximum= 100>
            </e-axis>
          </e-axes>
        </ejs-circulargauge>
      </div>
    </div>
  </template>
<script>
import Vue from 'vue';
import { CircularGaugePlugin, Gradient } from "@syncfusion/ej2-vue-
circulargauge";
Vue.use(CircularGaugePlugin);
export default {
  data: function () {
    var pointerRadialGradient = {
      radius: '50%',
      innerPosition: { x: '50%', y: '50%' },
      outerPosition: { x: '50%', y: '50%' },
      colorStop: [
        { color: '#FEF3F9', offset: '0%', opacity: 0.9 },
        { color: '#E63B86', offset: '60%', opacity: 0.9 } ]
    }
    return {
      lineStyle: { width: 3, color: '#E63B86' },
      labelStyle: {
        font: { size: '0px' }
      }, majorTicks: {
        height: 0
      }, minorTicks: {
        height: 0
      },
      radius: '90%',
      pointers: [{
        radius: '80%',
        value: 80,
        animation: { enable: true, duration: 1000 },
        pointerWidth: 10,
        radialGradient: pointerRadialGradient,
        cap: {
          radius: 8,
          color: 'white',
          border: {
            color: '#E63B86',
            width: 1
          }
        }
      }
    ]
  }
}
```

```

    },
    needleTail: {
      length: '20%',
      radialGradient: pointerRadialGradient
    }
  }, {
    radius: '60%', value: 40,
    animation: { duration: 1000 },
    pointerWidth: 10,
    radialGradient: pointerRadialGradient,
    cap: {
      radius: 8, color: 'white',
      border: { color: '#E63B86', width: 1 }
    },
    needleTail: {
      length: '20%',
      radialGradient: pointerRadialGradient
    }
  }
]]
}
},
provide: {
  circulargauge: [Gradient]
}
};
</script>
<style>
.wrapper {
  max-width: 500px;
  margin: 0 auto;
}
</style>

```

{% previewsample "page.domainurl/code-snippet/circular-gauge/getting-started-cs48" %}

## Gauge annotations in Vue Circular gauge component

Annotations are used to mark a specific area of interest in the gauge with texts, shapes or images.

### Content

You can place any custom element on the axis area by assigning the id of the element to

[content](#) property of [annotation](#) object.

Note: To use annotation feature, we need to inject **Annotations** module using **CircularGauge.Inject(Annotations)** method.

### APP.VUE

```

<template>
  <div id="app">
    <div class='wrapper'>
      <ejs-circulargauge>
        <e-axes>
          <e-axis :annotations='annotations'>
            <e-pointers>

```

```

        <e-pointer value=50 ></e-pointer>
      </e-pointers>
    </e-axis>
  </e-axes>
</ejs-circulargauge>
</div>
</div>
</template>
<script>
import Vue from "vue";
import { CircularGaugePlugin, Annotations } from "@syncfusion/ej2-vue-circulargauge";
let contentVue = Vue.component("contentTemplate", {
  template: '<div><span style="font-size:10px; color:#424242; font-family:Regular">pointer Value: 50</span></div>',
  data() {
    return {
      data: {}
    };
  }
});
let contentTemplate = function() {
  return { template: contentVue };
};
Vue.use(CircularGaugePlugin);
export default {
  data: function() {
    return {
      annotations: [{
        content: contentTemplate,
        zIndex: '1'
      }]
    },
    provide: {
      circulargauge: [Annotations]
    }
  };
</script>
<style>
  .wrapper {
    max-width: 300px;
    margin: 0 auto;
  }
</style>

```

{% previewsample "page.domainurl/code-snippet/circular-gauge/getting-started-cs1" %}

### Position

Annotation can be placed around the axis by using [radius](#)

and [angle](#) property.

For example, if the angle is 90 degree and the radius is 110%, then the annotation, will be placed at the right side of the axis.



Radius of the annotation takes value either in pixel or percentage. By setting value in percentage, annotation gets its position with respect to its axis radius.

#### APP.VUE

```
<template>
  <div id="app">
    <div class='wrapper'>
      <ejs-circulargauge>
        <e-axes>
          <e-axis :annotations='annotations'>
            <e-pointers>
              <e-pointer value=50 ></e-pointer>
            </e-pointers>
          </e-axis>
        </e-axes>
      </ejs-circulargauge>
    </div>
  </div>
</template>
<script>
import Vue from "vue";
import { CircularGaugePlugin, Annotations } from "@syncfusion/ej2-vue-circulargauge";
Vue.use(CircularGaugePlugin);
let contentVue = Vue.component("contentTemplate", {
  template: '<div><span style="font-size:10px; color:#424242; font-family:Regular">pointer Value: 50</span></div>',
  data() {
    return {
      data: {}
    };
  }
});
let contentTemplate = function() {
  return { template: contentVue };
};
export default {
  data:function(){
    return {
      annotations: [{
        content: contentTemplate,
        angle: 90,
        radius: '150%',
        zIndex: '1'
      }]
    }
  },
  provide: {
    circulargauge: [Annotations]
  }
};
</script>
<style>
  .wrapper {
    max-width: 300px;
    margin: 0 auto;
  }
</style>
```

```
}
</style>
```

```
{% previewsample "page.domainurl/code-snippet/circular-gauge/getting-started-cs2" %}
```

### Sub Gauge

As the annotation allows you to place any custom element, we can initialize a gauge to the element and can

be used to place that in another gauge.

### APP.VUE

```
<template>
  <div id="app">
    <div class='wrapper'>
      <ejs-circulargauge :axes='axes'>
      </ejs-circulargauge>
    </div>
  </div>
</template>
<script>
import Vue from "vue";
import { CircularGaugePlugin, Annotations } from "@syncfusion/ej2-vue-circulargauge";
Vue.use(CircularGaugePlugin);
export default {
  data: function() {
    return {
      axes: [{
        minimum: 0,
        maximum: 12,
        startAngle: 0,
        endAngle: 360,
        lineStyle: { width: 0 },
        ranges: [{
          start: 0, end: 3,
          color: 'rgba(29,29,29,0.7)'
        }, {
          start: 3, end: 12,
          color: 'rgba(168,145,102,0.1)'
        }],
        annotations: [{
          angle: 270,
          radius: '40%',
          content: '<div id="subGauge"
style="width:90px;height:90px;"></div>'
        }, {
          angle: 90,
          radius: '40%',
          content: '<div id="time"><span>6:30 PM</span></div>',
          zIndex: '1'
        }],
        labelStyle: {
          hiddenLabel: 'First'
        }
      }],
    }
  },
}
```

```

    pointers: [{
      pointerWidth: 5,
      radius: '40%',
      value: 6.5,
      color: 'rgb(29,29,29)',
      border: { width: 1, color: 'rgb(29,29,29)' },
      cap: {
        color: 'rgb(29,29,29)',
        radius: 0,
        border: {
          width: 0.2,
          color: 'red'
        }
      },
      needleTail: {
        length: '0%'
      },
      animation: {
        enable: false
      }
    },
    {
      radius: '60%',
      pointerWidth: 5,
      color: 'rgb(29,29,29)',
      border: {
        width: 1,
        color: 'rgb(29,29,29)'
      },
      value: 6,
      cap: {
        color: 'rgb(29,29,29)',
        radius: 0,
        border: {
          width: 0.2,
          color: 'red'
        }
      },
      needleTail: {
        length: '0%'
      },
      animation: {
        enable: false
      }
    },
    {
      radius: '70%',
      pointerWidth: 4,
      value: 9.8,
      color: 'rgba(168,145,102,1)',
      cap: {
        color: 'rgba(168,145,102,1)',
        radius: 4,
        border: {
          width: 0.2,
          color: 'rgba(168,145,102,1)'
        }
      }
    }
  ]

```

```

        },
        needleTail: {
            color: 'rgba(168,145,102,1)',
            length: '20%'
        },
        animation: {
            enable: false,
            duration: 500
        }
    }
}]]
    }
},
provide: {
    circulargauge: [Annotations]
}
};
</script>
<style>
    .wrapper {
        max-width: 300px;
        margin: 0 auto;
    }
</style>

```

{% previewsample "page.domainurl/code-snippet/circular-gauge/getting-started-cs3" %}

See also

- [Tooltip for Annotation](#)

### Animation in Vue Circular Gauge component

All of the elements in the Circular Gauge, such as the axis lines, ticks, labels, ranges, pointers, and annotations, can be animated sequentially by using the [animationDuration](#) property. The animation for the Circular Gauge is enabled when the `animationDuration` property is set to an appropriate value in milliseconds, providing a smooth rendering effect for the component. If the `animationDuration` property is set to `0`, which is the default value, the animation effect is disabled. If the animation is enabled, the component will behave in the following order.

1. The axis line will be animated in the rendering direction (clockwise or anticlockwise).
2. Each tick line and label will then be animated.
3. If available, ranges will be animated.
4. If available, pointers will be animated in the same way as [pointer animation](#).
5. If available, annotations will be animated.

The animation of the Circular Gauge is demonstrated in the following example.

#### APP.VUE

```

<template>
  <div id="app">
    <div class='wrapper'>

```

```

    <ejs-circulargauge :animationDuration=2000
:background='background' style='display:block' align='center' id='gauge'>
    <e-axes>
        <e-axis :radius='gaugeRadius' :annotations='annotations'
:startAngle='startAngle' :endAngle='endAngle' :majorTicks='majorTicks'
:lineStyle='lineStyle' :minorTicks='minorTicks' :labelStyle='labelStyle'>
            <e-pointers>
                <e-pointer :value='value' :radius='pointerRadius'
:color='color' :pointerWidth='pointerWidth' :cap='cap'
:needleTail='needleTail'></e-pointer>
            </e-pointers>
            <e-ranges>
                <e-range start="0" end="30" color="#E63B86"
startWidth="22" endWidth="22" radius="60%"
:linearGradient="linearGradient"></e-range>
                <e-range start="30" end="60" color="#E0E0E0"
startWidth="22" endWidth="22" radius="60%"></e-range>
            </e-ranges>
        </e-axis>
    </e-axes>
</ejs-circulargauge>
</div>
</div>
</template>
<script>
import Vue from 'vue';
import { CircularGaugePlugin, Annotations, Gradient } from '@syncfusion/ej2-
vue-circulargauge';
Vue.use(CircularGaugePlugin);
export default {
    data() {
        return {
            annotations:[{
                angle:165,
                radius:'35%',
                content:'<div style="font-size:18px;margin-left: -20px;margin-
top: -12px; color:#9DD55A">60</div>',
                zIndex: '1'
            }],
            linearGradient: {
                startValue: '0%',
                endValue: '100%',
                colorStop: [
                    { color: '#9e40dc', offset: '0%', opacity: 1 },
                    { color: '#d93c95', offset: '70%', opacity: 1 },
                ],
            },
            gaugeRadius: '80%',
            startAngle: 230,
            endAngle: 130,
            background:'transparent',
            majorTicks: {
                offset: 5
            },
            lineStyle: {width: 8, color: '#E0E0E0' },
            minorTicks: {
                offset: 5
            }
        }
    }
}

```

```

    },
    labelStyle: {
      font: {
        fontFamily: 'inherit'
      },
      offset: -1
    },
    value: 60,
    pointerRadius: '60%',
    pointerWidth: 7,
    color: '#c06c84',
    cap: {
      radius: 8,
      color: '#c06c84',
      border: { width: 0 }
    },
    needleTail: {
      length: '0%',
    }
  },
  provide: {
    circulargauge: [Annotations, Gradient]
  },
}
</script>
<style>
.wrapper {
  max-width: 400px;
  margin: 0 auto;
}
</style>

```

{% previewsample "page.domainurl/code-snippet/circular-gauge/getting-started-cs70" %}

Only the pointer of the Circular Gauge can be animated individually, not the axis lines, ticks, labels, ranges, and annotations. You can refer this [link](#) to enable only pointer animation.

### Gauge legend in Vue Circular gauge component

Legend provides valuable information for interpreting what the circular gauge axis range displays, and they can be represented in various colors, shapes, and other identifiers based on the data. It gives a breakdown of what each symbol represents in the axis range of circular gauge.

You can add the legend for circular gauge ranges by setting the visible property of `legendSettings` to true.

#### Legend customization

Customization option is also provided for the legend shape, alignment, and position.

##### Position and alignment

The position of the legend is used to place legend in various positions. You can use the `position` property in `legendSettings`. Based on the position, the legend item will be aligned. The following options are available to customize the legend position:

- Top
- Bottom
- Left
- Right
- Custom
- Auto

The legend alignment is used to align the legend items in specific location. You can use the alignment property in `legendSettings` to align the legend items. The following options are available to customize the legend alignment:

- Near
- Center
- Far

The legends can also be positioned to absolute position using the `location.x` and `location.y` properties available in `legendSettings`.

#### *Legend size*

The legend size can be modified using the height and width properties in `legendSettings`.

#### *Legend opacity*

To specify the transparency for legend shape, set the opacity property in `legendSettings`.

#### *Legend shape*

To change the legend item shape, specify the desired shape in the shape property of the legend. By default, the shape of the legend is circle.

It also supports the following shapes:

- Rectangle
- Diamond
- Triangle
- InvertedTriangle
- Image
- Circle

You can customize a shape using the `shapeWidth` and `shapeHeight` properties.

#### *Legend padding*

You can control the spacing between the legend items using the padding option of the legend. The default value of padding is 5.

#### *Legend border*

You can customize the legend border using the border option in the legend. The legend border can be customized using the border color and width properties.

<!-- markdownlint-disable MD009 -->

#### *Font of the legend text*

The font of the legend item text can be customized using the following properties:

- fontFamily
- fontStyle
- fontWeight
- opacity
- color
- size

The following code example shows how to add legend in the gauge.

#### **APP.VUE**

```
<template>
  <div id="app">
    <div class='wrapper'>
      <ejs-circulargauge :legendSettings= 'legendSettings'>
        <e-axes>
          <e-axis minimum=0 maximum=100 :majorTicks= 'majorTicks'
:minorTicks= 'minorTicks' :labelStyle= 'labelStyle' :ranges='ranges'></e-
axis>
        </e-axes>
      </ejs-circulargauge>
    </div>
  </div>
</template>
<script>
import Vue from 'vue';
import { CircularGaugePlugin, Legend } from "@syncfusion/ej2-vue-
circulargauge";
Vue.use(CircularGaugePlugin);
export default {
  data: function () {
    return {
      majorTicks: {
        useRangeColor: true
      },
      minorTicks: {
        useRangeColor: true
      },
      labelStyle: {
        useRangeColor: true
      },
      ranges: [{
        start: 0,
        end: 25,
        radius: '108%'
      }, {
        start: 25,
        end: 50,
        radius: '108%'
      }, {
        start: 50,
        end: 75,
        radius: '108%'
      }, {
        start: 75,
        end: 100,
```



```

        radius: '108%'
      }],
      legendSettings : {
        visible: true,
        shapeWidth:30,
        shapeHeight:30,
        padding:15,
        border: {
          color:'green',
          width:3
        }
      }
    },
    provide: {
      circulargauge: [Legend]
    }
  };
</script>
<style>
  .wrapper {
    max-width: 300px;
    margin: 0 auto;
  }
</style>

```

{% previewsample "page.domainurl/code-snippet/circular-gauge/getting-started-cs31" %}

### Toggle option in legend

The toggle option has been provided for legend. So, if you toggle the legend, the given color will be changed to the corresponding circular gauge range. You can enable the toggle option using **toggleVisibility** in the **legendSettings** property.

### APP.VUE

```

<template>
  <div id="app">
    <div class='wrapper'>
      <ejs-circulargauge :legendSettings= 'legendSettings'>
        <e-axes>
          <e-axis minimum=0 maximum=100 :majorTicks= 'majorTicks'
:minorTicks= 'minorTicks' :labelStyle= 'labelStyle' :ranges='ranges'></e-
axis>
        </e-axes>
      </ejs-circulargauge>
    </div>
  </div>
</template>
<script>
import Vue from 'vue';
import { CircularGaugePlugin, Legend } from "@syncfusion/ej2-vue-
circulargauge";
Vue.use(CircularGaugePlugin);
export default {
  data: function () {
    return {

```

```

        majorTicks: {
            useRangeColor: true
        },
        minorTicks: {
            useRangeColor: true
        },
        labelStyle: {
            useRangeColor: true
        },
        ranges: [{
            start: 0,
            end: 25,
            radius: '108%'
        }, {
            start: 25,
            end: 50,
            radius: '108%'
        }, {
            start: 50,
            end: 75,
            radius: '108%'
        }, {
            start: 75,
            end: 100,
            radius: '108%'
        }],
        legendSettings : {
            visible: true,
            toggleVisibility: true
        }
    },
    provide: {
        circulargauge: [Legend]
    }
};
</script>
<style>
    .wrapper {
        max-width: 300px;
        margin: 0 auto;
    }
</style>

```

{% previewsample "page.domainurl/code-snippet/circular-gauge/getting-started-cs32" %}

### Paging support in legend

By default, paging will be enabled if the legend items exceed the legend bounds. You can view each legend item by navigating between the pages using navigation buttons.

### APP.VUE

```

<template>
    <div id="app">
        <div class='wrapper'>
            <ejs-circulargauge :legendSettings= 'legendSettings'>

```

```

        <e-axes>
            <e-axis minimum=0 maximum=100 :majorTicks= 'majorTicks'
:minorTicks= 'minorTicks' :labelStyle= 'labelStyle' :ranges='ranges'></e-
axis>
        </e-axes>
    </ejs-circulargauge>
</div>
</template>
<script>
import Vue from 'vue';
import { CircularGaugePlugin, Legend } from "@syncfusion/ej2-vue-
circulargauge";
Vue.use(CircularGaugePlugin);
export default {
    data: function () {
        return {
            majorTicks: {
                useRangeColor: true
            },
            minorTicks: {
                useRangeColor: true
            },
            labelStyle: {
                useRangeColor: true
            },
            ranges: [{
                start: 0,
                end: 25,
                radius: '108%'
            }, {
                start: 25,
                end: 50,
                radius: '108%'
            }, {
                start: 50,
                end: 75,
                radius: '108%'
            }, {
                start: 75,
                end: 100,
                radius: '108%'
            }
        ],
            legendSettings : {
                visible: true,
                height: '50'
            }
        }
    },
    provide: {
        circulargauge: [Legend]
    }
};
</script>
<style>
    .wrapper {
        max-width: 300px;

```

```
margin: 0 auto;
}
</style>
```

{% previewsample "page.domainurl/code-snippet/circular-gauge/getting-started-cs33" %}

### Legend text customization

You can customize the legend text using `legendText` property in `ranges`.

#### APP.VUE

```
<template>
  <div id="app">
    <div class='wrapper'>
      <ejs-circulargauge :legendSettings= 'legendSettings'>
        <e-axes>
          <e-axis minimum=0 maximum=100 :majorTicks= 'majorTicks'
:minorTicks= 'minorTicks' :labelStyle= 'labelStyle' :ranges='ranges'></e-
axis>
        </e-axes>
      </ejs-circulargauge>
    </div>
  </div>
</template>
<script>
import Vue from 'vue';
import { CircularGaugePlugin, Legend } from "@syncfusion/ej2-vue-
circulargauge";
Vue.use(CircularGaugePlugin);
export default {
  data: function () {
    return {
      majorTicks: {
        useRangeColor: true
      },
      minorTicks: {
        useRangeColor: true
      },
      labelStyle: {
        useRangeColor: true
      },
      ranges: [{
        start: 0,
        end: 25,
        radius: '108%',
        legendText: 'light air'
      }, {
        start: 25,
        end: 50,
        radius: '108%',
        legendText: 'light air'
      }, {
        start: 50,
        end: 75,
        radius: '108%',
        legendText: 'light breeze'
      }
    ]
  }
}
```

```

      }, {
        start: 75,
        end: 100,
        radius: '108%',
        legendText: 'Gentle breeze'
      }],
      legendSettings : {
        visible: true,
      }
    },
    provide: {
      circulargauge: [Legend]
    }
  };
</script>
<style>
  .wrapper {
    max-width: 300px;
    margin: 0 auto;
  }
</style>

```

{% previewsample "page.domainurl/code-snippet/circular-gauge/getting-started-cs34" %}

**legendRendering** event will be triggered before rendering each legend item, using this event you can customize needed legend items using following arguments.

Argument name	Description
fill	Specifies the legend shape color
text	Specifies the current legend text
shape	Customize the shape of the legends
name	Specifies the name of the event
cancel	Set to true, to cancel the event status

## Gauge user interaction in Vue Circular gauge component

### Tooltip for pointer

Circular gauge will displays the pointer details through [tooltip](#),

when the mouse is moved over the pointer.

<!-- markdownlint-disable MD036 -->

### Tooltip for ranges

Circular gauge displays the information about the ranges through tooltip when hovering the mouse over the ranges. You can enable this feature by setting the type property of tooltip to 'Range' in the array collection.

### Tooltip customization for ranges

To customize the range tooltip, use the `rangeSettings` property in tooltip. The following options are available to customize the range tooltip:

- `fill` - Specifies the range tooltip fill color.
- `textStyle` - Specifies the range tooltip text style.
- `format` - Specifies the range content format.
- `template` - Specifies the custom template for tooltip.
- `enableAnimation` - Animates as it moves from one point to another.
- `border` - Specifies the tooltip border.
- `showMouseAtPosition` - Displays the position of the tooltip on the cursor position.

### Tooltip for annotation

Circular gauge displays the information about the annotations through tooltip when hovering the mouse over the annotation. You can enable this feature by setting the type property of tooltip to 'Annotation' in the array collection.

### Tooltip customization for annotation

To customize the annotation tooltip, use the `annotationSettings` property in tooltip. The following options are available to customize the annotation tooltip:

- `fill` - Specifies the annotation tooltip fill color.
- `textStyle` - Specifies the annotation tooltip text style.
- `format` - Specifies the annotation content format.
- `template` - Specifies the tooltip content with custom template.
- `enableAnimation` - Animates as it moves from one point to another.
- `border` - Specifies the tooltip border.

The following code example shows the tooltip for the pointer, ranges and annotation.

### APP.VUE

```
<template>
<div id="app">
  <div class='wrapper'>
    <ejs-circulargauge style='display:block' align='center'
id='tooltip-container' :enablePointerDrag='enablePointerDrag'
:tooltip='tooltip'>
      <e-axes>
        <e-axis :annotations='annotations' :radius='gaugeradius'
:startDate='startDate' minimum=0 maximum=120 :endDate='endDate'
:majorTicks='majorTicks' :lineStyle='lineStyle' :minorTicks='minorTicks'
:labelStyle='labelStyle' :ranges='ranges'>
          <e-pointers>
            <e-pointer :value='value' :cap='cap'
:radius='pointerRadius' :color='color' :animation='animation'></e-pointer>
          </e-pointers>
        </e-axis>
      </e-axes>
    </ejs-circulargauge>
  </div>
</div>
</template>
```

```
<script>
import Vue from 'vue';
import {CircularGaugePlugin, GaugeTooltip, Annotations } from
"@syncfusion/ej2-vue-circulargauge";
Vue.use(CircularGaugePlugin);
export default {
  data: function () {
    return {
      annotations: [{
        content: 'CircularGauge', zIndex:'1', angle: 180
      }],
      gaugeradius: '90%',
      startAngle: 240,
      endAngle: 120,
      lineStyle: {
        width: 0
      },
      majorTicks: {
        color: 'white',
        offset: -5,
        height: 12
      },
      minorTicks: {
        width: 0
      },
      labelStyle: {
        useRangeColor: true,
        font: {
          color: '#424242',
          size: '13px',
          fontFamily: 'Roboto'
        }
      },
      value: 70,
      pointerRadius: '60%',
      color: '#33BCBD',
      cap: {
        radius: 10,
        border: {
          color: '#33BCBD',
          width: 5
        }
      },
      animation: {
        enable: true,
        duration: 1500
      },
      ranges: [{
        start: 0,
        end: 50,
        startWidth: 10,
        endWidth: 10,
        radius: '102%',
        color: '#3A5DC8',
      }, {
        start: 50,
        end: 120,
```

```

        radius: '102%',
        startWidth: 10,
        endWidth: 10,
        color: '#33BCBD',
      }],
      tooltip: {
        type: ['Pointer', 'Range', 'Annotation'],
        enable: true,
        enableAnimation: false,
        annotationSettings: { template: '<div>CircularGauge</div>' },
        rangeSettings: { fill: 'red' }
      },
      enablePointerDrag: true
    }
  },
  provide: {
    circulargauge: [GaugeTooltip, Annotations]
  },
};
</script>
<style>
.wrapper {
  max-width: 300px;
  margin: 0 auto;
}
#templateWrap img {
  border-radius: 30px;
  width: 30px;
  height: 30px;
  margin: 0 auto;
}
#templateWrap .des {
  float: right;
  padding-left: 10px;
  line-height: 30px;
}
</style>

```

{% previewsample "page.domainurl/code-snippet/circular-gauge/getting-started-cs60" %}

### Template

Any HTML elements can be displayed in the tooltip by using the

[template](#) property of the tooltip.

### APP.VUE

```

<template>
  <div id="app">
    <div class='wrapper'>
      <ejs-circulargauge :tooltip='tooltip' >
        <e-axes>
          <e-axis >
            <e-pointers>
              <e-pointer value=70 ></e-pointer>
            </e-pointers>
          </e-axis>

```



```

        </e-axes>
      </ejs-circulargauge>
    </div>
  </div>
</template>
<script>
import Vue from 'vue';
import { CircularGaugePlugin, GaugeTooltip } from '@syncfusion/ej2-vue-circulargauge';
Vue.use(CircularGaugePlugin);
export default {
  data: function () {
    return {
      tooltip: {
        enable: true,
        template: '<div>Pointer: 70 </div>'
      }
    },
    provide: {
      circulargauge: [GaugeTooltip]
    },
  };
</script>
<style>
.wrapper {
  max-width: 300px;
  margin: 0 auto;
}
</style>

```

{% previewsample "page.domainurl/code-snippet/circular-gauge/getting-started-cs61" %}

### Pointer Drag

Pointers can be dragged over the axis value. This can be achieved by clicking and dragging the pointer. To enable or disable the pointer drag, you can use

[enablePointerDrag](#) property.

### APP.VUE

```

<template>
  <div id="app">
    <div class='wrapper'>
      <ejs-circulargauge id='tooltip-container' enablePointerDrag=
'enablePointerDrag' :tooltipRender='tooltipRender' :tooltip='tooltip' >
        <e-axes>
          <e-axis>
            <e-pointers>
              <e-pointer value=70 ></e-pointer>
            </e-pointers>
          </e-axis>
        </e-axes>
      </ejs-circulargauge>
    </div>
  </div>

```

```

</template>
<script>
import Vue from 'vue';
import { CircularGaugePlugin, GaugeTooltip } from "@syncfusion/ej2-vue-circulargauge";
Vue.use(CircularGaugePlugin);
export default {
  data: function () {
    return {
      enablePointerDrag: true
      tooltip: {
        enable: true,
        template: '<div id='templateWrap'><div class='des'>pointer
<span>${Math.round(value)} </span></div></div>'
      }
    }
  },
  provide: {
    circulargauge: [GaugeTooltip]
  },
  methods: {
    tooltipRender: function (args) {
      let cotainerObj = document.getElementById('tooltip-container');
      let value = args.pointer.currentValue;
      debugger;
      cotainerObj.ej2_instances[0].setPointerValue(0, 0, value);
    }
  }
};
</script>
<style>
.wrapper {
  max-width: 300px;
  margin: 0 auto;
}
#templateWrap .des {
  float: right;
  padding-left: 10px;
  line-height: 30px;
}
</style>

```

{% previewsample "page.domainurl/code-snippet/circular-gauge/getting-started-cs62" %}

[changes](#)

#### APP.VUE

```

<template>
<div id="app">
  <div class='wrapper'>
    <ejs-circulargauge style='display:block' align='center' id='tooltip-
container' :title='title' :titleStyle='titleStyle'
:enablePointerDrag='enablePointerDrag' :tooltipRender='tooltipRender'
:tooltip='tooltip'>
      <e-axes>

```

```

        <e-axis :radius='gaugeradius' :startAngle='startAngle'
minimum=0 maximum=120 :endAngle='endAngle' :majorTicks='majorTicks'
:lineStyle='lineStyle' :minorTicks='minorTicks' :labelStyle='labelStyle'
:ranges='ranges'>
            <e-pointers>
                <e-pointer :value='value' :cap='cap'
:radius='pointerRadius' :color='color' :animation='animation'></e-pointer>
            </e-pointers>
        </e-axis>
    </e-axes>
</ejs-circulargauge>
</div>
</div>
</template>
<script>
import Vue from 'vue';
import {CircularGaugePlugin, GaugeTooltip} from "@syncfusion/ej2-vue-
circulargauge";
Vue.use(CircularGaugePlugin);
export default {
    data: function () {
        return {
            title: 'Tooltip Customization',
            titleStyle: {
                size: '15px',
                color: 'grey'
            },
            gaugeradius: '90%',
            startAngle: 240,
            endAngle: 120,
            lineStyle: {
                width: 0
            },
            majorTicks: {
                color: 'white',
                offset: -5,
                height: 12
            },
            minorTicks: {
                width: 0
            },
            labelStyle: {
                useRangeColor: true,
                font: {
                    color: '#424242',
                    size: '13px',
                    fontFamily: 'Roboto'
                }
            },
            value: 70,
            pointerRadius: '60%',
            color: '#33BCBD',
            cap: {
                radius: 10,
                border: {
                    color: '#33BCBD',
                    width: 5
                }
            }
        }
    }
}

```

```

    },
    animation: {
      enable: true,
      duration: 1500
    },
    ranges: [{
      start: 0,
      end: 50,
      startWidth: 10,
      endWidth: 10,
      radius: '102%',
      color: '#3A5DC8',
    }, {
      start: 50,
      end: 120,
      radius: '102%',
      startWidth: 10,
      endWidth: 10,
      color: '#33BCBD',
    }],
    tooltip: {
      enable: true,
      fill: 'transparent',
      template: "<div id='templateWrap'><img
src='src/circulargauge/images/range1.png'/><img
src='src/circulargauge/images/range3.png'/><div
class='des'><span>${Math.round(pointers[0].value)} MPH</span></div></div>",
      border: {
        color: '#33BCBD',
        width: 2
      }
    },
    enablePointerDrag: true
  }
},
provide: {
  circulargauge: [GaugeTooltip]
},
methods: {
  load: function (args) {
    let selectedTheme = location.hash.split("/") [1];
    selectedTheme = selectedTheme ? selectedTheme : "Material";
    args.gauge.theme =
      selectedTheme.charAt(0).toUpperCase() +
selectedTheme.slice(1);
  },
  tooltipRender: function (args) {
    let color;
    let cotainerObj = document.getElementById('tooltip-container');
    let value = args.pointer.currentValue;
    debugger;
    let content = args.content;
    if (value >= 0 && value <= 50) {
      let color = '#3A5DC8';
      content.children[1].remove();
      args.textStyle.color = color;
    }
  }
}

```

```

        args.border.color = color;

cotaainerObj.ej2_instances[0].axes[0].pointers[0].animation.enable = false;
        cotaainerObj.ej2_instances[0].axes[0].pointers[0].color =
color;

cotaainerObj.ej2_instances[0].axes[0].pointers[0].cap.border.color = color;
        cotaainerObj.ej2_instances[0].setPointerValue(0, 0, value);
    } else {
        let color = '#33BCBD';
        content.children[0].remove();
        args.textStyle.color = color;
        args.border.color = color;

cotaainerObj.ej2_instances[0].axes[0].pointers[0].animation.enable = false;
        cotaainerObj.ej2_instances[0].axes[0].pointers[0].color =
color;

cotaainerObj.ej2_instances[0].axes[0].pointers[0].cap.border.color = color;
        cotaainerObj.ej2_instances[0].setPointerValue(0, 0, value);
    }
}
}
};
</script>
<style>
    .wrapper {
        max-width: 300px;
        margin: 0 auto;
    }
    #templateWrap img {
        border-radius: 30px;
        width: 30px;
        height: 30px;
        margin: 0 auto;
    }
    #templateWrap .des {
        float: right;
        padding-left: 10px;
        line-height: 30px;
    }
</style>

```

{% previewsample "page.domainurl/code-snippet/circular-gauge/getting-started-cs63" %}

## Print and export in Vue Circular Gauge component

### Print

The Circular Gauge can be printed directly from the browser by calling the [print](#) method. More information on the arguments for this method can be found [here](#).

To use the print functionality, inject the **Print** module using the **provide** option and set the property **allowPrint** in the Circular Gauge to **true**.

### APP.VUE

```

<template>
  <div id="app">
    <div class='wrapper'>
      <ejs-button id='print' isToggle="true" v-on:click.native='clickPrint'>
Print
      </ejs-button>
      <ejs-circulargauge id="gauge" ref="gauge" allowPrint="true">
      </ejs-circulargauge>
    </div>
  </div>
</template>
<script>
import Vue from 'vue';
import { CircularGaugePlugin, Print } from "@syncfusion/ej2-vue-
circulargauge";
import { ButtonPlugin } from '@syncfusion/ej2-vue-buttons';
Vue.use(CircularGaugePlugin);
Vue.use(ButtonPlugin);
export default {
  data() {
    return {
    },
  },
  methods: {
    clickPrint: function (args) {
      this.$refs.gauge.ej2Instances.print();
    }
  },
  provide: {
    circulargauge: [Print],
  }
};
</script>
<style>
.wrapper {
  max-width: 300px;
  margin: 0 auto;
}
</style>

```

{% previewsample "page.domainurl/code-snippet/circular-gauge/getting-started-cs49" %}

## Export

### Image Export

The Circular Gauge can be exported as an image in **JPEG**, **PNG** and **SVG** formats. To use the image export functionality, inject the **ImageExport** module using the **provide** option and set the Circular Gauge's [allowImageExport](#) property to **true**. When the [export](#) method is used, the Circular Gauge will be exported as an image. The **export** method requires the following parameters for image export.

Parameter	Description
type	Specifies the export type, which accepts values such as <b>JPEG</b> , <b>PNG</b> and <b>SVG</b> for image export.

- | fileName | Specifies the file name of the exported image. |
- | orientation | This parameter is not required for image export. So, the value must be **null**. |
- | allowDownload | Specifies whether the exported image file should be downloaded or not. When the value is set to **true**, the file will be downloaded. When set to **false**, the base64 string of the file will be returned. This is an optional parameter, with the default value set to **true**. |

**APP.VUE**

```

<template>
  <div id="app">
    <div class='wrapper'>
      <ejs-button id='export' isToggle="true" v-
on:click.native='clickExport'> Export </ejs-button>
      <ejs-circulargauge id="gauge" ref="gauge" allowImageExport="true">
      </ejs-circulargauge>
    </div>
  </div>
</template>
<script>
import Vue from 'vue';
import { CircularGaugePlugin, ImageExport } from "@syncfusion/ej2-vue-
circulargauge";
import { ButtonPlugin } from '@syncfusion/ej2-vue-buttons';
Vue.use(CircularGaugePlugin);
Vue.use(ButtonPlugin);
export default {
  data() {
    return {
    },
    methods: {
      clickExport: function (args) {
        this.$refs.gauge.ej2Instances.export('PNG', 'Gauge');
      }
    },
    provide: {
      circulargauge: [ImageExport],
    }
  };
</script>
<style>
.wrapper {
  max-width: 300px;
  margin: 0 auto;
}
</style>

```

{% previewsample "page.domainurl/code-snippet/circular-gauge/getting-started-cs50" %}

The base64 string of the exported image files can only be received using the **export** method for JPEG and PNG formats. The **allowDownload** parameter in the **export** method must be set to **false**, as explained in the table above.

**APP.VUE**

```

<template>
  <div id="app">
    <div class='wrapper'>
      <ejs-button id='export' isToggle="true" v-
on:click.native='clickExport'> Export </ejs-button>
      <ejs-circulargauge id="gauge" ref="gauge" allowImageExport="true">
      </ejs-circulargauge>
    </div>
  </div>
</template>
<script>
import Vue from 'vue';
import { CircularGaugePlugin, ImageExport } from "@syncfusion/ej2-vue-
circulargauge";
import { ButtonPlugin } from '@syncfusion/ej2-vue-buttons';
import { PdfPageOrientation } from '@syncfusion/ej2-pdf-export';
Vue.use(CircularGaugePlugin);
Vue.use(ButtonPlugin);
export default {
  data() {
    return {
    },
  },
  methods: {
    clickExport: function (args) {
      this.$refs.gauge.ej2Instances.export('PNG', 'Gauge',
PdfPageOrientation.Landscape, false).then((data) => {
        document.writeln(data);
      })
    },
  },
  provide: {
    circulargauge: [ImageExport]
  }
};
</script>
<style>
.wrapper {
  max-width: 300px;
  margin: 0 auto;
}
</style>

```

{% previewsample "page.domainurl/code-snippet/circular-gauge/getting-started-cs68" %}

#### PDF Export

The Circular Gauge can be exported as a PDF document using [export](#) method. To use the PDF export functionality, inject the PdfExport module using the [provide](#) option and set the Circular Gauge's [allowPdfExport](#) property to **true**. The [export](#) method requires the following parameters for PDF export.

Parameter	Description
type	Specifies the export type, which accepts values such as <b>PDF</b> for pdf export.



- | **fileName** | Specifies the file name of the exported PDF document. |
- | **orientation** | Specifies the orientation of the exported PDF document which accepts the values such as **Portrait** and **Landscape**. The default value is **Portrait**. |
- | **allowDownload** | This is an optional parameter, with the default value set to **true**. This parameter is supported only for the image export. The exported PDF document cannot be returned as base64 string. |

**APP.VUE**

```

<template>
  <div id="app">
    <div class='wrapper'>
      <ejs-button id='export' isToggle="true" v-
on:click.native='clickExport'> Export </ejs-button>
      <ejs-circulargauge id="gauge" ref="gauge" allowPdfExport="true">
      </ejs-circulargauge>
    </div>
  </div>
</template>
<script>
import Vue from 'vue';
import { CircularGaugePlugin, PdfExport } from "@syncfusion/ej2-vue-
circulargauge";
import { ButtonPlugin } from '@syncfusion/ej2-vue-buttons';
import { PdfPageOrientation } from '@syncfusion/ej2-pdf-export';
Vue.use(CircularGaugePlugin);
Vue.use(ButtonPlugin);
export default {
  data() {
    return {
    },
  },
  methods: {
    clickExport: function (args) {
      this.$refs.gauge.ej2Instances.export('PDF', 'Gauge',
PdfPageOrientation.Portrait);
    }
  },
  provide: {
    circulargauge: [PdfExport],
  }
};
</script>
<style>
.wrapper {
  max-width: 300px;
  margin: 0 auto;
}
</style>

```

{% previewsample "page.domainurl/code-snippet/circular-gauge/getting-started-cs69" %}

## Gauge appearance in Vue Circular gauge component

### Gauge Title

Circular gauge can be given a title by using [title](#) property, to show the information about the gauge.

Title can be customized by using [titleStyle](#) property in gauge.

#### APP.VUE

```
<template>
  <div id="app">
    <div class='wrapper'>
      <ejs-circulargauge title= 'Speedometer' :titleStyle=
'titleStyle'>
    </ejs-circulargauge>
    </div>
  </div>
</template>
<script>
import Vue from 'vue';
import { CircularGaugePlugin } from "@syncfusion/ej2-vue-circulargauge";
Vue.use(CircularGaugePlugin);
export default {
  data: function () {
    return {
      titleStyle: {
        color: '#27d5ff'
      }
    }
  };
};
</script>
<style>
.wrapper {
  max-width: 300px;
  margin: 0 auto;
}
</style>
```

{% previewsample "page.domainurl/code-snippet/circular-gauge/getting-started-cs4" %}

### Gauge Position

<!-- markdownlint-disable MD036 -->

Gauge can be positioned anywhere in the container with the help of

[centerX](#) and

[centerY](#)

property and it accepts values either in percentage or in pixels.

The default value of the [centerX](#) and

[centerY](#) property is 50%, which means gauge will get rendered to the centre of the container.

#### In Pixel

You can set the mid point of the gauge in pixel as demonstrated below,

**APP.VUE**

```

<template>
  <div id="app">
    <div class='wrapper'>
      <ejs-circulargauge>
        <e-axes>
          <e-axis startAngle= 90 endAngle= 180 :lineStyle=
'lineStyle'>
            </e-axis>
          </e-axes>
        </ejs-circulargauge>
      </div>
    </div>
  </template>
<script>
import Vue from 'vue';
import { CircularGaugePlugin } from "@syncfusion/ej2-vue-circulargauge";
Vue.use(CircularGaugePlugin);
export default {
  data: function () {
    return {
      lineStyle: {
        width: 2,
        color: '#F8F8F8'
      }
    }
  }
};
</script>
<style>
  .wrapper {
    max-width: 200px;
    max-height: 100px;
    margin: 0px;
  }
</style>

```

{% previewsample "page.domainurl/code-snippet/circular-gauge/getting-started-cs5" %}

*In Percentage*

By setting the value in percentage, gauge gets its mid point with respect to its plot area.

For example, when the [centerX](#)

value as '0%' and [centerY](#) value is '50%', gauge will get positioned at the top left corner of the plot area.

**APP.VUE**

```

<template>
  <div id="app">
    <div class='wrapper'>
      <ejs-circulargauge>
        <e-axes>
          <e-axis startAngle= 0 endAngle= 180 :lineStyle=
'lineStyle'>
            </e-axis>

```

```

        </e-axes>
      </ejs-circulargauge>
    </div>
  </div>
</template>
<script>
import Vue from 'vue';
import { CircularGaugePlugin } from '@syncfusion/ej2-vue-circulargauge';
Vue.use(CircularGaugePlugin);
export default {
  data: function () {
    return {
      lineStyle: {
        width: 2,
        color: '#F8F8F8'
      }
    }
  }
};
</script>
<style>
.wrapper {
  max-width: 300px;
  max-height: 100px;
  margin: 0px;
}
</style>

```

{% previewsample "page.domainurl/code-snippet/circular-gauge/getting-started-cs6" %}

## Area Customization

*Customize the gauge background*

Using [background](#) and

[border](#) properties, you can change the background color and border of the circular gauge.

## APP.VUE

```

<template>
  <div id="app">
    <div class='wrapper'>
      <ejs-circulargauge>
        <e-axes>
          <e-axis startAngle= 230 endAngle= 130 maximum= 120
radius= '90%':pointers = 'pointers' :ranges='ranges' :lineStyle= 'lineStyle'
:minorTicks= 'minorTicks' :majorTicks = 'majorTicks'>
            </e-axis>
          </e-axes>
        </ejs-circulargauge>
      </div>
    </div>
  </template>
<script>
import Vue from 'vue';
import { CircularGaugePlugin } from '@syncfusion/ej2-vue-circulargauge';
Vue.use(CircularGaugePlugin);

```

```

export default {
  data: function () {
    return {
      majorTicks: {
        width: 1, color: '#8c8c8c'
      },
      lineStyle: { width: 2 },
      minorTicks: {
        width: 1, color: '#8c8c8c'
      },
      pointers: [{
        value: 60,
        radius: '60%'
      }],
      ranges: [{
        start: 0,
        end: 70,
        radius: '110%',
        strokeWidth: 10
      }, {
        start: 70,
        end: 110,
        radius: '110%',
        strokeWidth: 10
      }, {
        start: 110,
        end: 120,
        radius: '110%',
        strokeWidth: 10
      }
    ]
  }
};
</script>
<style>
  .wrapper {
    max-width: 300px;
    margin: 0 auto;
  }
</style>

```

{% previewsample "page.domainurl/code-snippet/circular-gauge/getting-started-cs7" %}

### Gauge Margin

You can set margin for gauge from its container through

[margin](#) property.

### APP.VUE

```

<template>
  <div id="app">
    <div class='wrapper'>
      <ejs-circulargauge background= 'skyblue' :border= 'border'
      :margin= 'margin' >
        <e-axes>

```

```

        <e-axis startAngle= 230 endAngle= 130 maximum= 120
radius= '90%':pointers = 'pointers' :ranges='ranges' :lineStyle= 'lineStyle'
:minorTicks= 'minorTicks' :majorTicks = 'majorTicks'>
        </e-axis>
    </e-axes>
</ejs-circulargauge>
</div>
</div>
</template>
<script>
import Vue from 'vue';
import { CircularGaugePlugin } from "@syncfusion/ej2-vue-circulargauge";
Vue.use(CircularGaugePlugin);
export default {
    data: function () {
        return {
            border: {color: "#FF0000", width: 2},
            margin: { left: 40, right: 40, top: 40, bottom: 40 },
            majorTicks: {
                width: 1, color: '#8c8c8c'
            },
            lineStyle: { width: 2 },
            minorTicks: {
                width: 1, color: '#8c8c8c'
            },
            pointers: [{
                value: 60,
                radius: '60%'
            }],
            ranges: [{
                start: 0,
                end: 70,
                radius: '110%',
                strokeWidth: 10
            }, {
                start: 70,
                end: 110,
                radius: '110%',
                strokeWidth: 10
            }, {
                start: 110,
                end: 120,
                radius: '110%',
                strokeWidth: 10
            }
        ]
    }
};
</script>
<style>
    .wrapper {
        max-width: 400px;
        max-height:100px;
        margin: 0 auto;
    }
</style>

```

{% previewsample "page.domainurl/code-snippet/circular-gauge/getting-started-cs8" %}

### Radius calculation based on angles

Render semi or quarter circular gauges by modifying the start and end angles. By enabling the radius based on angle option, the radius of circular gauge will be calculated based on the start and end angles to avoid excess white space.

#### APP.VUE

```
<template>
  <div id="app">
    <div class='wrapper'>
      <ejs-circulargauge>
        <e-axes>
          <e-axis startAngle= 270 endAngle= 90 radius= '80%'
:lineStyle= 'lineStyle'>
            </e-axis>
          </e-axes>
        </ejs-circulargauge>
      </div>
    </div>
  </template>
<script>
import Vue from 'vue';
import { CircularGaugePlugin } from "@syncfusion/ej2-vue-circulargauge";
Vue.use(CircularGaugePlugin);
export default {
  data: function () {
    return {
      lineStyle: {
        width: 2,
        color: '#F8F8F8'
      },
    }
  }
};
</script>
<style>
.wrapper {
  max-width: 300px;
  max-height: 100px;
  margin: 0px;
}
</style>
```

{% previewsample "page.domainurl/code-snippet/circular-gauge/getting-started-cs9" %}

### Accessibility in Vue Circular gauge component

Circular Gauge has built-in accessibility features like screen reading and WAI-ARIA attributes.

#### WAI-ARIA attributes

The Circular Gauge component followed the [WAI-ARIA](#) patterns to meet the accessibility. The following ARIA attributes are used in the Circular Gauge component:

| Attributes | Purpose |

| --- | --- |

| **role=region** | It is specified in the pointer where the interactive drag and drop function is supported to update the pointer value. |

| **aria-label** | Provides an accessible name for the axis labels, legend title, legend item label, text pointer and annotation. |

### Screen reading in Circular Gauge

Accessibility in the Circular Gauge component ensures that all users, regardless of ability or disability, can use screen reading. The following Circular Gauge elements will be read aloud using screen reading software, such as Narrator for Windows.

| Elements | Description |

| --- | --- |

| Axis labels | Reads the axis labels of the Circular Gauge. |

| Legend title | Reads the title of the legend in the Circular Gauge. |

| Legend item label | Reads the label of the legend item in the Circular Gauge. |

| Text pointer | Reads the text content shown as a pointer in Circular Gauge. |

| Annotation | Reads the content specified in the annotation. |

### Ensuring accessibility

The Circular Gauge component's accessibility levels are ensured through an [accessibility-checker](#) and [axe-core](#) software tools during automated testing.

The accessibility compliance of the Circular Gauge component is shown in the following sample. Open the [sample](#) in a new window to evaluate the accessibility of the Circular Gauge component with accessibility tools.

```
{% previewsample "https://ej2.syncfusion.com/accessibility/circular-gauge.html" %}
```

See also

- [Accessibility in Syncfusion Vue components](#)

### Internationalization in Vue Circular Gauge component

Circular Gauge provides internationalization support for below elements.

- Axis Labels
- Tooltip

For more information about number formatter, you can refer [internationalization](#).

### Globalization

Globalization is the process of designing and developing a component that works in different cultures/locales.



Internationalization library is used to globalize number in Circular Gauge component using [format](#) property in [labelStyle](#).

#### Numeric Format

In the below example, axis labels are globalized to **EUR**.

#### APP.VUE

```
<template>
<div id="app">
  <div class='wrapper'>
    <ejs-circulargauge :axes='axes'>
    </ejs-circulargauge>
  </div>
</div>
</template>
<script>
import Vue from 'vue';
import { CircularGaugePlugin } from "@syncfusion/ej2-vue-circulargauge";
import { loadCldr, L10n, setCulture, setCurrencyCode } from
'@syncfusion/ej2-base';
setCulture('de');
setCurrencyCode('EUR');
Vue.use(CircularGaugePlugin);
export default {
  data: function () {
    return {
      axes: [{
        labelStyle: {
          position: 'Inside',
          //Label format set as currency.
          format: 'c'
        }
      }]
    }
  }
};
</script>
<style>
.wrapper {
  max-width: 300px;
  margin: 0 auto;
}
</style>
```

{% previewsample "page.domainurl/code-snippet/circular-gauge/getting-started-cs66" %}

#### Right-to-left

Circular Gauge can render its elements from right to left, which improves the user experience for certain language users. To do so, set the [enableRtl](#) property to **true**. When this property is enabled, elements such as the tooltip and legend will be rendered from right to left. Meanwhile, the axis can be rendered from right to left by setting the [direction](#) property to **AntiClockWise**. For more information on axis, click [here](#).

The following example illustrates the right to left rendering of the Circular Gauge.

**APP.VUE**

```

<template>
<div id="app">
  <div class='wrapper'>
    <ejs-circulargauge :axes='axes' :legendSettings='legendSettings'
:tooltip='tooltip' :enableRtl='enableRtl'>
    </ejs-circulargauge>
  </div>
</div>
</template>
<script>
import Vue from 'vue';
import { CircularGaugePlugin, GaugeTooltip, Legend } from "@syncfusion/ej2-
vue-circulargauge";
Vue.use(CircularGaugePlugin);
export default {
  data: function () {
    return {
      enableRtl: true,
      tooltip: {
        type: ['Pointer', 'Range'],
        format: 'Pointer : {value} ',
        enable: true,
        enableAnimation: false
      },
      legendSettings:{
        visible:true,
        position:'Right'
      },
      axes: [{
        direction: 'AntiClockWise',
        lineStyle: { width: 10, color: 'transparent' },
        labelStyle: {
          position: 'Inside', useRangeColor: false,
          font: {
            size: '12px',
            color: '#424242',
            fontFamily: 'Roboto',
            fontStyle: 'Regular'
          }
        },
      },
      majorTicks: {
        height: 10,
        offset: 5,
        color: '#9E9E9E'
      },
      minorTicks: { height: 0 },
      startAngle: 210,
      endAngle: 150,
      minimum: 0,
      maximum: 120,
      radius: '80%',
      ranges: [{
        start: 0,
        end: 40,
        color: '#30B32D'
      }
    ]
  }
}

```

```

    },
    {
      start: 40,
      end: 80,
      color: '#FFDD00'
    },
    {
      start: 80,
      end: 120,
      color: '#F03E3E'
    }
  ],
  pointers: [{
    animation: { enable: false },
    value: 65,
    radius: '60%',
    color: '#757575',
    pointerWidth: 8,
    cap: {
      radius: 7,
      color: '#757575'
    },
    needleTail: {
      length: '18%'
    }
  }
  ]
},
provide: {
  circulargauge: [GaugeTooltip, Legend]
}
};
</script>
<style>
  .wrapper {
    max-width: 300px;
    margin: 0 auto;
  }
</style>

```

{% previewsample "page.domainurl/code-snippet/circular-gauge/getting-started-cs67" %}

## ColorPicker

### Getting Started with the Vue Color picker Component in Vue 2

This article provides a step-by-step guide for setting up a Vue 2 project using [Vue-CLI](#) and integrating the Syncfusion Vue Color picker component using the [Composition API](#) / [Options API](#).

#### Prerequisites

[System requirements for Syncfusion Vue UI components](#)

#### Dependencies

The list of dependencies required to use the ColorPicker component in your application is given below:

```
`javascript`
```

```
|-- @syncfusion/ej2-vue-inputs
|-- @syncfusion/ej2-vue-base
|-- @syncfusion/ej2-base
|-- @syncfusion/ej2-inputs
|-- @syncfusion/ej2-buttons
|-- @syncfusion/ej2-popups
|-- @syncfusion/ej2-splitbuttons
`
```

### Setting up the Vue 2 project

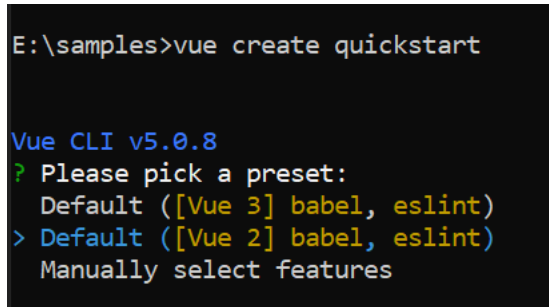
To generate a Vue 2 project using Vue-CLI, use the [vue create](#) command. Follow these steps to install Vue CLI and create a new project:

```
`bash
npm install -g @vue/cli
vue create quickstart
cd quickstart
npm run serve
`
```

or

```
`bash
yarn global add @vue/cli
vue create quickstart
cd quickstart
yarn run serve
`
```

When creating a new project, choose the option **Default ([Vue 2] babel, eslint)** from the menu.



```
E:\samples>vue create quickstart

Vue CLI v5.0.8
? Please pick a preset:
  Default ([Vue 3] babel, eslint)
> Default ([Vue 2] babel, eslint)
  Manually select features
```

Once the **quickstart** project is set up with default settings, proceed to add Syncfusion components to the project.

### Add Syncfusion Vue packages

Syncfusion packages are available at [npmjs.com](https://www.npmjs.com). To use Vue components, install the required npm package.

This article uses the [Vue Color picker component](#) as an example. Install the `@syncfusion/ej2-vue-inputs` package by running the following command:

```
`bash
```

```
npm install @syncfusion/ej2-vue-inputs --save
```

```
,
```

or

```
`bash
```

```
yarn add @syncfusion/ej2-vue-inputs
```

```
,
```

### Import Syncfusion CSS styles

You can import themes for the Syncfusion Vue component in various ways, such as using CSS or SASS styles from npm packages, CDN, [CRG](#) and [Theme Studio](#). Refer to [themes topic](#) to know more about built-in themes and different ways to refer to themes in a Vue project.

In this article, the `Material` theme is applied using CSS styles, which are available in installed packages. The necessary `Material` CSS styles for the Color picker component and its dependents were imported into the `<style>` section of `src/App.vue` file.

#### ~/SRC/APP.VUE

```
<style>
@import '../node_modules/@syncfusion/ej2-base/styles/material.css';
@import '../node_modules/@syncfusion/ej2-buttons/styles/material.css';
@import '../node_modules/@syncfusion/ej2-popups/styles/material.css';
@import '../node_modules/@syncfusion/ej2-splitbuttons/styles/material.css';
@import '../node_modules/@syncfusion/ej2-inputs/styles/material.css';
</style>
```

### Add Syncfusion Vue component

Follow the below steps to add the Vue Color picker component using `Composition API` or `Options API`:

1\ First, import and register the Color picker component in the `script` section of the `src/App.vue` file. If you are using the `Composition API`, you should add the `setup` attribute to the `script` tag to indicate that Vue will be using the `Composition API`.

#### COMPOSITION API (~/SRC/APP.VUE)

```
<script>
import { ColorPickerComponent as EjsColorpicker } from '@syncfusion/ej2-vue-inputs';
</script>
```

#### OPTIONS API (~/SRC/APP.VUE)

```
<script>
import { ColorPickerComponent } from '@syncfusion/ej2-vue-inputs';
export default {
  components: {
    'ejs-colorpicker': ColorPickerComponent
  }
}
</script>
```

2\ In the **template** section, define the Color picker component.

#### **COMPOSITION API (~SRC/APP.VUE)**

```
<template>
<div class='wrap'>
<h4>Choose Color</h4>
<ejs-colorpicker></ejs-colorpicker>
</div>
</template>
```

Here is the summarized code for the above steps in the **src/App.vue** file:

#### **COMPOSITION API (~SRC/APP.VUE)**

```
<template>
<div class='wrap'>
  <h4>Choose Color</h4>
  <ejs-colorpicker></ejs-colorpicker>
</div>
</template>
<script setup>
import { ColorPickerComponent as EjsColorpicker } from '@syncfusion/ej2-vue-inputs';
import { enableRipple } from '@syncfusion/ej2-base';
enableRipple(true);
</script>
<style>
@import '../node_modules/@syncfusion/ej2-base/styles/material.css';
@import '../node_modules/@syncfusion/ej2-buttons/styles/material.css';
@import '../node_modules/@syncfusion/ej2-popups/styles/material.css';
@import '../node_modules/@syncfusion/ej2-splitbuttons/styles/material.css';
@import '../node_modules/@syncfusion/ej2-inputs/styles/material.css';
.wrap {
  margin: 0 auto;
  width: 300px;
  text-align: center;
}
</style>
```

#### **OPTIONS API (~SRC/APP.VUE)**

```
<template>
<div class='wrap'>
  <h4>Choose Color</h4>
  <ejs-colorpicker></ejs-colorpicker>
</div>
```

```

</template>
<script>
import { ColorPickerComponent } from '@syncfusion/ej2-vue-inputs';
import { enableRipple } from '@syncfusion/ej2-base';
enableRipple(true);
export default {
  components: {
    'ejs-colorpicker': ColorPickerComponent
  },
}
</script>
<style>
@import '../node_modules/@syncfusion/ej2-base/styles/material.css';
@import '../node_modules/@syncfusion/ej2-buttons/styles/material.css';
@import '../node_modules/@syncfusion/ej2-popups/styles/material.css';
@import '../node_modules/@syncfusion/ej2-splitbuttons/styles/material.css';
@import '../node_modules/@syncfusion/ej2-inputs/styles/material.css';
.wrap {
  margin: 0 auto;
  width: 300px;
  text-align: center;
}
</style>

```

### Run the project

To run the project, use the following command:

```
`bash
```

```
npm run serve
```

```
,
```

or

```
`bash
```

```
yarn run serve
```

```
,
```

```
{% previewsample "page.domainurl/code-snippet/color-picker/default-cs2" %}
```

### Inline type

By default, the ColorPicker will be rendered using SplitButton and open the pop-up to access the ColorPicker. To render the ColorPicker container alone and to access it directly, render it as inline. It can be achieved by setting the [inline](#) property to `true`.

The following sample shows the inline type rendering of ColorPicker.

### COMPOSITION API (~SRC/APP.VUE)

```

<template>
  <div class='wrap'>
    <h4>Choose Color</h4>
    <ejs-colorpicker :inline="true" :showButtons="false"></ejs-colorpicker>
  </div>
</template>

```

```

<script setup>
import { ColorPickerComponent as EjsColorpicker } from '@syncfusion/ej2-vue-inputs';
import { enableRipple } from '@syncfusion/ej2-base';
enableRipple(true);
</script>
<style>
@import '../node_modules/@syncfusion/ej2-base/styles/material.css';
@import '../node_modules/@syncfusion/ej2-buttons/styles/material.css';
@import '../node_modules/@syncfusion/ej2-popups/styles/material.css';
@import '../node_modules/@syncfusion/ej2-splitbuttons/styles/material.css';
@import '../node_modules/@syncfusion/ej2-inputs/styles/material.css';
.wrap {
  margin: 0 auto;
  width: 300px;
  text-align: center;
}
</style>

```

### OPTIONS API (~SRC/APP.VUE)

```

<template>
  <div class='wrap'>
    <h4>Choose Color</h4>
    <ejs-colorpicker :inline="true" :showButtons="false"></ejs-colorpicker>
  </div>
</template>
<script>
import { ColorPickerComponent } from '@syncfusion/ej2-vue-inputs';
import { enableRipple } from '@syncfusion/ej2-base';
enableRipple(true);
export default {
  components: {
    'ejs-colorpicker': ColorPickerComponent
  }
}
</script>
<style>
@import '../node_modules/@syncfusion/ej2-base/styles/material.css';
@import '../node_modules/@syncfusion/ej2-buttons/styles/material.css';
@import '../node_modules/@syncfusion/ej2-popups/styles/material.css';
@import '../node_modules/@syncfusion/ej2-splitbuttons/styles/material.css';
@import '../node_modules/@syncfusion/ej2-inputs/styles/material.css';
.wrap {
  margin: 0 auto;
  width: 300px;
  text-align: center;
}
</style>

```

{% previewsample "page.domainurl/code-snippet/color-picker/default-cs3" %}

> The `showButtons` property is disabled in this sample because the control buttons are not needed for inline type. To know about the control buttons functionality, refer to the [showButtons](#) sample.



## See Also

- [Set color value](#)
- [ColorPicker customization](#)

## Getting Started with the Vue ColorPicker Component in Vue 3

This article provides a step-by-step guide for setting up a [Vite](#) project with a JavaScript environment and integrating the Syncfusion Vue ColorPicker component using the [Composition API](#) / [Options API](#).

The **Composition API** is a new feature introduced in Vue.js 3 that provides an alternative way to organize and reuse component logic. It allows developers to write components as functions that use smaller, reusable functions called composition functions to manage their properties and behavior.

The **Options API** is the traditional way of writing Vue.js components, where the component logic is organized into a series of options that define the component's properties and behavior. These options include data, methods, computed properties, watchers, lifecycle hooks, and more.

### Prerequisites

#### [System requirements for Syncfusion Vue UI components](#)

#### Set up the Vite project

A recommended approach for beginning with Vue is to scaffold a project using [Vite](#). To create a new Vite project, use one of the commands that are specific to either NPM or Yarn.

```
`bash
npm create vite@latest
```

```
,
```

or

```
`bash
yarn create vite
```

```
,
```

Using one of the above commands will lead you to set up additional configurations for the project as below:

1. Define the project name: We can specify the name of the project directly. Let's specify the name of the project as **my-project** for this article.

```
`bash
? Project name: » my-project
```

```
,
```

2. Select **Vue** as the framework. It will create a Vue 3 project.

```
`bash
? Select a framework: » - Use arrow-keys. Return to submit.
```

Vanilla

Vue

React

Preact

Lit

Svelte

Others

,

3. Choose **JavaScript** as the framework variant to build this Vite project using JavaScript and Vue.

```
`bash
```

? Select a variant: » - Use arrow-keys. Return to submit.

JavaScript

TypeScript

Customize with create-vue ↗

Nuxt ↗

,

4. Upon completing the aforementioned steps to create the **my-project**, run the following command to install its dependencies:

```
`bash
```

```
cd my-project
```

```
npm install
```

,

or

```
`bash
```

```
cd my-project
```

```
yarn install
```

,

Now that **my-project** is ready to run with default settings, let's add Syncfusion components to the project.

#### Add Syncfusion Vue packages

Syncfusion Vue component packages are available at [npmjs.com](https://www.npmjs.com). To use Syncfusion Vue components in the project, install the corresponding npm package.

This article uses the [Vue ColorPicker component](#) as an example. To use the Vue ColorPicker component in the project, the **@syncfusion/ej2-vue-inputs** package needs to be installed using the following command:

```
`bash
```

```
npm install @syncfusion/ej2-vue-inputs --save
```

```
,
```

```
or
```

```
`bash
```

```
yarn add @syncfusion/ej2-vue-inputs
```

```
,
```

### Import Syncfusion CSS styles

You can import themes for the Syncfusion Vue component in various ways, such as using CSS or SASS styles from npm packages, CDN, [CRG](#) and [Theme Studio](#). Refer to [themes topic](#) to know more about built-in themes and different ways to refer to themes in a Vue project.

In this article, **Material** theme is applied using CSS styles, which are available in installed packages. The necessary **Material** CSS styles for the ColorPicker component and its dependents were imported into the `<style>` section of **src/App.vue** file.

#### ~/SRC/APP.VUE

```
<style>
@import '../node_modules/@syncfusion/ej2-base/styles/material.css';
@import '../node_modules/@syncfusion/ej2-buttons/styles/material.css';
@import '../node_modules/@syncfusion/ej2-popups/styles/material.css';
@import '../node_modules/@syncfusion/ej2-splitbuttons/styles/material.css';
@import '../node_modules/@syncfusion/ej2-inputs/styles/material.css';
</style>
```

The order of importing CSS styles should be in line with its dependency graph.

### Add Syncfusion Vue component

Follow the below steps to add the Vue ColorPicker component using **Composition API** or **Options API**:

1.First, import and register the ColorPicker component and its child directives in the **script** section of the **src/App.vue** file. If you are using the **Composition API**, you should add the **setup** attribute to the **script** tag to indicate that Vue will be using the **Composition API**.

#### COMPOSITION API (~/SRC/APP.VUE)

```
<script setup>
import { ColorPickerComponent as EjsColorpicker } from "@syncfusion/ej2-vue-inputs";
</script>
```

#### OPTIONS API (~/SRC/APP.VUE)

```
<script>
import { ColorPickerComponent } from "@syncfusion/ej2-vue-inputs";
//Component registration
export default {
  name: "App",
  components: {
    "ejs-colorpicker": ColorPickerComponent
  }
}
```

```
}  
}  
</script>
```

Here is the summarized code for the above steps in the **src/App.vue** file:

### **COMPOSITION API (~SRC/APP.VUE)**

```
<template>  
<ejs-colorpicker></ejs-colorpicker>  
</template>  
<script setup>  
import { ColorPickerComponent as EjsColorpicker } from "@syncfusion/ej2-vue-inputs";  
</script>  
<style>  
@import '../node_modules/@syncfusion/ej2-base/styles/material.css';  
@import '../node_modules/@syncfusion/ej2-buttons/styles/material.css';  
@import '../node_modules/@syncfusion/ej2-popups/styles/material.css';  
@import '../node_modules/@syncfusion/ej2-splitbuttons/styles/material.css';  
@import '../node_modules/@syncfusion/ej2-inputs/styles/material.css';  
</style>
```

### **OPTIONS API (~SRC/APP.VUE)**

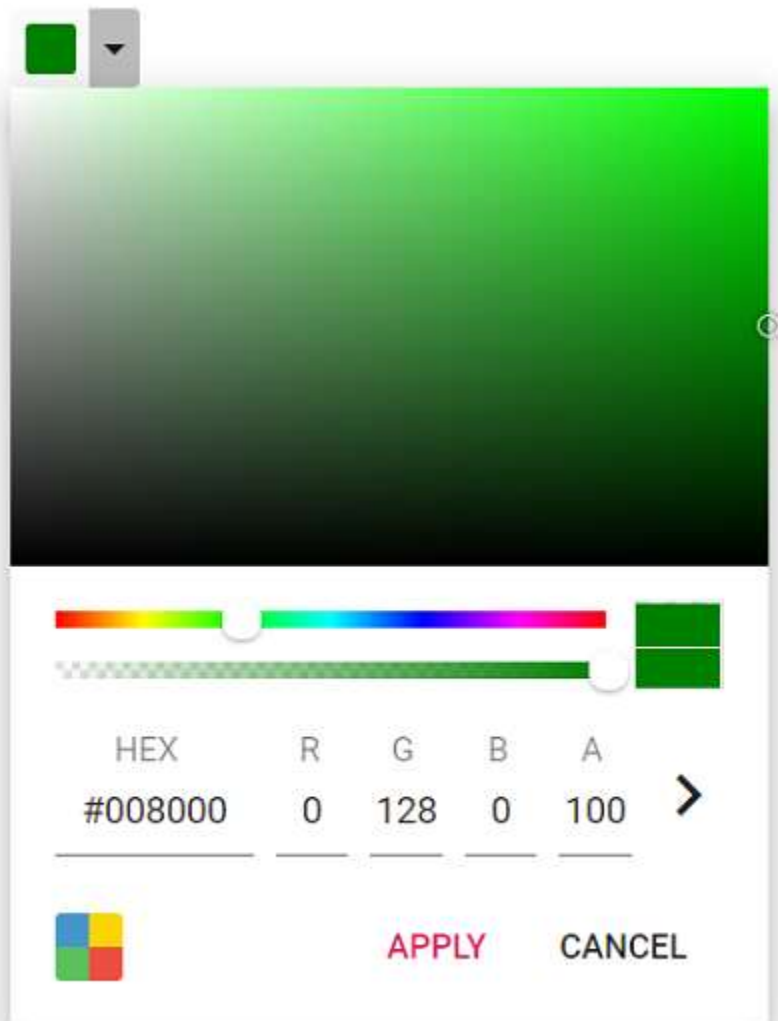
```
<template>  
<ejs-colorpicker></ejs-colorpicker>  
</template>  
<script>  
import { ColorPickerComponent } from "@syncfusion/ej2-vue-inputs";  
// Component registration  
export default {  
  name: "App",  
  // Declaring component and its directives  
  components: {  
    "ejs-colorpicker": ColorPickerComponent  
  },  
  // Bound properties declarations  
  data() {  
    return {  
    };  
  }  
};  
</script>  
<style>  
@import '../node_modules/@syncfusion/ej2-base/styles/material.css';  
@import '../node_modules/@syncfusion/ej2-buttons/styles/material.css';  
@import '../node_modules/@syncfusion/ej2-popups/styles/material.css';  
@import '../node_modules/@syncfusion/ej2-splitbuttons/styles/material.css';  
@import '../node_modules/@syncfusion/ej2-inputs/styles/material.css';  
</style>
```

### Run the project

To run the project, use the following command:

```
`bash
npm run dev
`
or
`bash
yarn run dev
`
```

The output will appear as follows:



For migrating from Vue 2 to Vue 3, refer to the [migration](#) documentation.

See also

- [Getting Started with Vue UI Components using Composition API and TypeScript](#)
- [Getting Started with Vue UI Components using Options API and TypeScript](#)

## Mode and value in Vue Color picker component

### Rendering palette at initial load

By default, the **Picker** area will be rendered at initial load. To render the **Palette** area while opening the ColorPicker pop-up, and specify the [mode](#) property as **Palette**.

In the following sample, it will render the **Palette** at initial load.

#### APP.VUE

```
<template>
<div class='wrap'>
  <h4>Choose Color</h4>
  <ejs-colorpicker mode="Palette"></ejs-colorpicker>
</div>
</template>
<script>
import Vue from 'vue';
import { ColorPickerPlugin } from '@syncfusion/ej2-vue-inputs';
import { enableRipple } from '@syncfusion/ej2-base';
enableRipple(true);
Vue.use(ColorPickerPlugin);
export default {}
</script>
<style>
@import '../node_modules/@syncfusion/ej2-base/styles/material.css';
@import '../node_modules/@syncfusion/ej2-buttons/styles/material.css';
@import '../node_modules/@syncfusion/ej2-popups/styles/material.css';
@import '../node_modules/@syncfusion/ej2-splitbuttons/styles/material.css';
@import '../node_modules/@syncfusion/ej2-inputs/styles/material.css';
.wrap {
  margin: 0 auto;
  width: 300px;
  text-align: center;
}
</style>
```

{% previewsample "page.domainurl/code-snippet/color-picker/default-cs18" %}

### Color value

The [value](#) property can be used to specify the color value to the ColorPicker. It supports either **three** or **six** digit hex codes. To include **opacity**, set the color value as **four** or **eight** digit hex code.

In the following sample, the color value sets as **four** digit hex code, the last digit represents the **opacity** value.

#### APP.VUE

```
<template>
<div class='wrap'>
  <h4>Choose Color</h4>
  <ejs-colorpicker value="035a" :modeSwitcher="false"></ejs-colorpicker>
</div>
</template>
<script>
import Vue from 'vue';
```

```

import { ColorPickerPlugin } from '@syncfusion/ej2-vue-inputs';
import { enableRipple } from '@syncfusion/ej2-base';
enableRipple(true);
Vue.use(ColorPickerPlugin);
export default {}
</script>
<style>
@import '../node_modules/@syncfusion/ej2-base/styles/material.css';
@import '../node_modules/@syncfusion/ej2-buttons/styles/material.css';
@import '../node_modules/@syncfusion/ej2-popups/styles/material.css';
@import '../node_modules/@syncfusion/ej2-splitbuttons/styles/material.css';
@import '../node_modules/@syncfusion/ej2-inputs/styles/material.css';
.wrap {
  margin: 0 auto;
  width: 300px;
  text-align: center;
}
</style>

```

{% previewsample "page.domainurl/code-snippet/color-picker/default-cs19" %}

> The [value](#) property supports hex code with or without # prefix.

See Also

- [How to render palette alone](#)
- [Custom palette](#)
- [No color support in palette](#)

## Localization in Vue Color picker component

### Localization

The **Localization** library allows you to localize default text content of the ColorPicker. The ColorPicker component has static text for **control buttons (apply / cancel)** and **mode switcher** that can be changed to other cultures (Arabic, Deutsch, French, etc.) by defining the [locale](#) value and translation object.

The following list of properties and its values are used in the ColorPicker.

Locale key words | Text

Apply | Apply

Cancel | Cancel

ModeSwitcher | Switch Mode

### Loading translations

To load translation object in an application use **load** function of **L10n** class.

The below example demonstrates the ColorPicker in **Deutsch** culture.

### APP.VUE

```

<template>
<div class='wrap'>

```

```

    <h4>Choose Color</h4>
    <ejs-colorpicker locale='de-DE'></ejs-colorpicker>
</div>
</template>
<script>
import Vue from 'vue';
import { ColorPickerPlugin } from '@syncfusion/ej2-vue-inputs';
import { enableRipple, L10n } from '@syncfusion/ej2-base';
enableRipple(true);
L10n.load({
  'de-DE': {
    'colorpicker': {
      'Apply': 'Anwenden',
      'Cancel': 'Abbrechen',
      'ModeSwitcher': 'Modus wechseln'
    }
  }
});
Vue.use(ColorPickerPlugin);
export default {}
</script>
<style>
@import '../node_modules/@syncfusion/ej2-base/styles/material.css';
@import '../node_modules/@syncfusion/ej2-buttons/styles/material.css';
@import '../node_modules/@syncfusion/ej2-popups/styles/material.css';
@import '../node_modules/@syncfusion/ej2-splitbuttons/styles/material.css';
@import '../node_modules/@syncfusion/ej2-inputs/styles/material.css';
.wrap {
  margin: 0 auto;
  width: 300px;
  text-align: center;
}
</style>

```

{% previewsample "page.domainurl/code-snippet/color-picker/default-cs16" %}

### Right to Left - RTL

ColorPicker component has **RTL** support. It helps to render the ColorPicker from right-to-left direction.

It improves the user experiences and accessibility for users who use right-to-left languages(Arabic, Farsi, Urdu, etc). This can be achieved by setting the [enableRtl](#) property to **true**.

The following example illustrates how to enable right-to-left support in ColorPicker component.

### APP.VUE

```

<template>
<div class='wrap'>
  <h4>Choose Color</h4>
  <ejs-colorpicker :enableRtl="true" locale='ar-AE'></ejs-colorpicker>
</div>
</template>
<script>
import Vue from 'vue';
import { ColorPickerPlugin } from '@syncfusion/ej2-vue-inputs';
import { enableRipple, L10n } from '@syncfusion/ej2-base';
enableRipple(true);

```



```

L10n.load({
  'ar-AE': {
    'colorpicker': {
      'Apply': 'تطبيق',
      'Cancel': 'إلغاء',
      'ModeSwitcher': 'مفتاح كهربائي الوضع'
    }
  }
});
Vue.use(ColorPickerPlugin);
export default {}
</script>
<style>
@import '../node_modules/@syncfusion/ej2-base/styles/material.css';
@import '../node_modules/@syncfusion/ej2-buttons/styles/material.css';
@import '../node_modules/@syncfusion/ej2-popups/styles/material.css';
@import '../node_modules/@syncfusion/ej2-splitbuttons/styles/material.css';
@import '../node_modules/@syncfusion/ej2-inputs/styles/material.css';
.wrap {
  margin: 0 auto;
  width: 300px;
  text-align: center;
}
</style>

```

{% previewsample "page.domainurl/code-snippet/color-picker/default-cs17" %}

See Also

- [More information about localization](#)

### Accessibility in Vue Color picker component

The Color picker component followed the accessibility guidelines and standards, including [ADA, Section 508](#), [WCAG 2.2](#) standards, and [WCAG roles](#) that are commonly used to evaluate accessibility.

The accessibility compliance for the Color picker component is outlined below.

| Accessibility Criteria | Compatibility |

| -- | -- |

| [WCAG 2.2](#) Support |  |

| [Section 508](#) Support |  |

| Screen Reader Support |  |

| Right-To-Left Support |  |

| Color Contrast |  |

| Mobile Device Support |  |

| Keyboard Navigation Support |  |

| [Accessibility Checker](#) Validation |  |

| [Axe-core](#) Accessibility Validation |  |

<style>

```
.post .post-content img {
display: inline-block;
margin: 0.5em 0;
}
```

</style>

<div> - All features of the component meet the requirement.</div>

<div> - Some features of the component do not meet the requirement.</div>

<div> - The component does not meet the requirement.</div>

### WAI-ARIA attributes

The Color picker component followed the WAI-ARIA patterns to meet the accessibility. The following ARIA attributes are used in the Color picker component:

| Attributes | Purpose |

| --- | --- |

| **role** | Indicates the Color picker component as **color** and the tiles as **gridcell** in the color palette. |

| **aria-label** | Indicates the accessible name for the tiles. |

| **aria-selected** | Indicates the current selected state of the tile. |

| **aria-haspopup** | Indicates the availability of the popup element. |

| **aria-expanded** | Indicates whether the popup can be expanded or collapsed, as well as indicates whether its current state is expanded or collapsed. |

| **aria-owns** | Identifies an elements in order to define a visual, functional, or contextual parent/child relationship between DOM elements where the DOM hierarchy cannot be used to represent the relationship. |

| **aria-disabled** | Indicates that the element is perceivable but disabled, so it is not editable or otherwise operable. |

### Keyboard interaction

The Color picker component followed the keyboard interaction guideline, making it easy for people who use assistive technologies (AT) and those who completely rely on keyboard navigation. The following keyboard shortcuts are supported by the Color picker component.

| **Press** | **To do this** |

| --- | --- |

| **Up Arrow** | Moves the handler/tile up from the current position. |

| **Down Arrow** | Moves the handler/tile down from the current position. |

| **Left Arrow** | Moves the handler/tile left from the current position. |

| **Right Arrow** | Moves the handler/tile right from the current position. |

| **Enter** | Apply the selected color value. |

| **Tab** | To focus the next focusable element in the Color picker popup. |

### Ensuring accessibility

The Color picker component's accessibility levels are ensured through an [accessibility-checker](#) and [axe-core](#) software tools during automated testing.

The accessibility compliance of the Color picker component is shown in the following sample. Open the [sample](#) in a new window to evaluate the accessibility of the Color picker component with accessibility tools.

```
{% previewsample "page.domainurl/code-snippet/color-picker/default-cs1" %}
```

See also

- [Accessibility in Syncfusion Vue components](#)

### Two way binding in Vue Color picker component

It can be achieved by using the `v-model` directive in vue. In the following sample the color value is selected in one ColorPicker will automatically changes in the other ColorPicker. It will update in the other ColorPicker using value property.

#### APP.VUE

```
<template>
<div>
<div class='wrapper1'>
  <h4>Choose Color</h4>
  <ejs-colorpicker v-model="value"></ejs-colorpicker>
</div>
<div class="wrapper2">
  <h4>Choose Color</h4>
  <ejs-colorpicker v-model="value"></ejs-colorpicker>
</template>
</div>
</div>
<script>
import Vue from 'vue';
import { ColorPickerPlugin } from '@syncfusion/ej2-vue-inputs';
```

```

import { enableRipple } from '@syncfusion/ej2-base';
enableRipple(true);
Vue.use(ColorPickerPlugin);
export default {
  data() {
    return {
      value: null
    }
  }
}
</script>
<style>
@import '../node_modules/@syncfusion/ej2-base/styles/material.css';
@import '../node_modules/@syncfusion/ej2-buttons/styles/material.css';
@import '../node_modules/@syncfusion/ej2-popups/styles/material.css';
@import '../node_modules/@syncfusion/ej2-splitbuttons/styles/material.css';
@import '../node_modules/@syncfusion/ej2-inputs/styles/material.css';
.wrapper1 {
  /* margin: 0 auto;
  width: 300px;
  text-align: center; */
  float: left;
  padding: 10px 0 0 160px;
}
.wrapper2 {
  float: right;
  padding: 10px 330px 0 0;
}
</style>

```

{% previewsample "page.domainurl/code-snippet/color-picker/default-cs20" %}

### Style and appearance in Vue Color picker component

To modify the ColorPicker appearance, you need to override the default CSS of ColorPicker component. Please find the list of CSS classes and its corresponding section in ColorPicker component. Also, you have an option to create your own custom theme for the controls using our [Theme Studio](#).

#### CSS Class | Purpose of Class

- |.e-custom-picker .e-container .e-handler|To customized Color Picker selection handler
- |.color-picker.e-dropdown-popup ul .e-container|To customize the Color Picker container
- |.color-picker.e-dropdown-popup ul .e-item.e-palette-item|To customize the Color Picker palette item
- |.color-picker.e-dropdown-popup .e-container .e-switch|To customize the Color Picker switch control
- |.color-picker.e-dropdown-popup .e-container .e-slider-preview|To customize the Color Picker slider control

### How To

#### Hide control buttons in Vue Color picker component

ColorPicker can be rendered without control buttons (Apply/Cancel). In this case, while selecting a color, the

ColorPicker pop-up is closed and selected colors can be applied directly. To hide control buttons, set the [showButtons](#) property to `false`.

#### APP.VUE

```
<template>
<div class='wrap'>
  <h4>Choose Color</h4>
  <ejs-colorpicker :showButtons="false"></ejs-colorpicker>
</div>
</template>
<script>
import Vue from 'vue';
import { ColorPickerPlugin } from '@syncfusion/ej2-vue-inputs';
import { enableRipple } from '@syncfusion/ej2-base';
enableRipple(true);
Vue.use(ColorPickerPlugin);
export default {}
</script>
<style>
@import '../node_modules/@syncfusion/ej2-base/styles/material.css';
@import '../node_modules/@syncfusion/ej2-buttons/styles/material.css';
@import '../node_modules/@syncfusion/ej2-popups/styles/material.css';
@import '../node_modules/@syncfusion/ej2-splitbuttons/styles/material.css';
@import '../node_modules/@syncfusion/ej2-inputs/styles/material.css';
.wrap {
  margin: 0 auto;
  width: 300px;
  text-align: center;
}
</style>
```

{% previewsample "page.domainurl/code-snippet/color-picker/default-cs14" %}

Render palette alone in Vue Color picker component

To render the **Palette** alone in ColorPicker, specify the [mode](#) property as **Palette**, and set the [modeSwitcher](#) property to `false`.

In the following sample, the [showButtons](#) property is disabled to hide the control buttons and it renders only the **Palette** area.

#### APP.VUE

```
<template>
<div class='wrap'>
  <h4>Choose Color</h4>
  <ejs-colorpicker mode="Palette" :showButtons="false"
:modeSwitcher="false"></ejs-colorpicker>
</div>
</template>
<script>
import Vue from 'vue';
import { ColorPickerPlugin } from '@syncfusion/ej2-vue-inputs';
import { enableRipple } from '@syncfusion/ej2-base';
enableRipple(true);
Vue.use(ColorPickerPlugin);
```

```

export default {}
</script>
<style>
@import '../node_modules/@syncfusion/ej2-base/styles/material.css';
@import '../node_modules/@syncfusion/ej2-buttons/styles/material.css';
@import '../node_modules/@syncfusion/ej2-popups/styles/material.css';
@import '../node_modules/@syncfusion/ej2-splitbuttons/styles/material.css';
@import '../node_modules/@syncfusion/ej2-inputs/styles/material.css';
.wrap {
  margin: 0 auto;
  width: 300px;
  text-align: center;
}
</style>

```

{% previewsample "page.domainurl/code-snippet/color-picker/default-cs15" %}

> To render Picker alone specify the [mode](#) property as 'Picker'.

### Colorpicker in dropdownbutton in Vue Color picker component

This section explains about how to render the ColorPicker in DropDownButton. The [target](#) property of the DropDownButton helps to achieve this scenario. To know about the usage of [target](#) property refer to [Popup templating](#) section.

In the below sample, the color picker is rendered as inline type by setting [inline](#) property as `true` and the rendered color picker wrapper is passed as a [target](#) to the DropDownButton to achieve the above scenario.

### APP.VUE

```

<template>
<div class='wrap'>
  <h4>Choose Color</h4>
  <ejs-colorpicker :inline="true" :modeSwitcher="false"
:change="onChange"></ejs-colorpicker>
  <ejs-dropdownbutton id="dropdownbtn" ref="ddb" target=".e-colorpicker-
wrapper" iconCss="e-dropdownbtn-preview" :beforeClose="beforeDdbClose"
:open="ddbOpen"></ejs-dropdownbutton>
</div>
</template>
<script>
import Vue from 'vue';
import { ColorPickerPlugin } from '@syncfusion/ej2-vue-inputs';
import { DropDownButtonPlugin } from '@syncfusion/ej2-vue-splitbuttons';
import { enableRipple } from '@syncfusion/ej2-base';
enableRipple(true);
Vue.use(ColorPickerPlugin);
Vue.use(DropDownButtonPlugin);
export default {
  methods: {
    onChange: function(args) {

this.$refs.ddb.ej2Instances.element.children[0].style.backgroundColor =
args.currentValue.rgba;
      this.closePopup();
    },
  },

```

```

        ddbOpen: function(args) {
            args.element.querySelector('.e-cancel').addEventListener('click',
this.closePopup);
            this.tooltip();
        },
        beforeDdbClose: function(args) {
            args.element.querySelector('.e-
cancel').removeEventListener('click', this.closePopup);
        },
        closePopup: function() {
            this.$refs.ddb.ej2Instances.toggle()
        },
        tooltip: function() {
            var zindex = (document.getElementsByClassName('e-color-picker-
tooltip')[0]).style.zIndex;
            var zindexIntValue = parseInt(zindex) + 2;
            (document.getElementsByClassName('e-color-picker-
tooltip')[0]).style.zIndex = zindexIntValue.toString();
        }
    }
}
</script>
<style>
@import '../node_modules/@syncfusion/ej2-base/styles/material.css';
@import '../node_modules/@syncfusion/ej2-buttons/styles/material.css';
@import '../node_modules/@syncfusion/ej2-popups/styles/material.css';
@import '../node_modules/@syncfusion/ej2-splitbuttons/styles/material.css';
@import '../node_modules/@syncfusion/ej2-inputs/styles/material.css';
.wrap {
    margin: 0 auto;
    width: 300px;
    text-align: center;
}
/* DropDownButton preview customization */
#dropdownbtn .e-btn-icon.e-dropdownbtn-preview {
    background-color: #008000;
    height: 18px;
    width: 18px;
    margin-top: 0;
}
#dropdownbtn {
    padding: 4px;
}
</style>

```

```
{% previewsample "page.domainurl/code-snippet/color-picker/default-cs4" %}
```

## Customize colorpicker in Vue Color picker component

### Custom palette

By default, the Palette will be rendered with default colors. To load custom colors in the palette, specify the colors in the [presetColors](#) property. To customize the color palette, add a custom class to palette tiles using [BeforeTileRender](#) event.

The following sample demonstrates the above functionalities.

### APP.VUE

```

<template>
<div class='wrap'>
  <div id="preview"></div>
  <h4>Choose Color</h4>
  <ejs-colorpicker mode="Palette" value="#ba68c8" :columns="4"
:inline="true" :showButtons="false" :modeSwitcher="false"
:presetColors="customColors" :beforeTileRender="tileRender"
:change="onChange"></ejs-colorpicker>
</div>
</template>
<script>
import Vue from 'vue';
import { ColorPickerPlugin } from '@syncfusion/ej2-vue-inputs';
import { enableRipple } from '@syncfusion/ej2-base';
enableRipple(true);
Vue.use(ColorPickerPlugin);
export default {
data() {
  return {
    customColors: {
      'custom1': ['#ef9a9a', '#e57373', '#ef5350', '#f44336',
'#f48fb1', '#f06292',
'#ec407a', '#e91e63', '#ce93d8', '#ba68c8', '#ab47bc',
'#9c27b0', '#b39ddb',
'#9575cd', '#7e57c2', '#673ab7'],
      'custom2': ['#9fa8da', '#7986cb', '#5c6bc0', '#3f51b5',
'#90caf9', '#64b5f6',
'#42a5f5', '#2196f3', '#81d4fa', '#4fc3f7', '#29b6f6',
'#03a9f4',
'#80ddea', '#4dd0e1', '#26c6da', '#00bcd4'],
      'custom3': ['#80cbc4', '#4db6ac', '#26a69a', '#009688',
'#a5d6a7', '#81c784',
'#66bb6a', '#4caf50', '#c5e1a5', '#aed581', '#9ccc65',
'#8bc34a', '#e6ee9c',
'#dce775', '#d4e157', '#cddc39']
    },
  };
},
methods: {
  onChange: function(args) {
    document.getElementById("preview").style.backgroundColor =
args.currentValue.hex;
  },
  tileRender: function(args) {
    args.element.classList.add("e-icons");
    args.element.classList.add("e-custom-tile");
  }
}
}
</script>
<style>
@import '../node_modules/@syncfusion/ej2-base/styles/material.css';
@import '../node_modules/@syncfusion/ej2-buttons/styles/material.css';
@import '../node_modules/@syncfusion/ej2-popups/styles/material.css';
@import '../node_modules/@syncfusion/ej2-splitbuttons/styles/material.css';
@import '../node_modules/@syncfusion/ej2-inputs/styles/material.css';
.e-container {

```



```

    background-color: transparent;
    border-color: transparent;
    box-shadow: none;
  }
  /* Tile customization styles */
  .e-container .e-palette .e-custom-tile {
    border: 0;
    color: #fff;
    height: 36px;
    font-size: 18px;
    width: 36px;
    line-height: 36px;
    border-radius: 50%;
    margin: 2px 5px;
  }
  .e-container .e-custom-palette.e-palette-group {
    height: 182px;
  }
  /* Selected state icon */
  .e-container .e-palette .e-custom-tile.e-selected::before {
    content: '\e933';
  }
  .e-container .e-palette .e-custom-tile.e-selected {
    outline: none;
  }
  .wrap {
    margin: 0 auto;
    width: 300px;
    text-align: center;
  }
  #preview {
    background-color: #ba68c8;
    height: 50px;
    width: 100%;
  }
</style>

```

```
{% previewsample "page.domainurl/code-snippet/color-picker/default-cs5" %}
```

#### Hide input area from picker

By default, the input area will be rendered in ColorPicker. To hide the input area from it, add `e-hide-value` class to ColorPicker using the [cssClass](#) property.

In the following sample, the ColorPicker is rendered without input area.

#### APP.VUE

```

<template>
<div class='wrap'>
  <h4>Choose Color</h4>
  <ejs-colorpicker cssClass="e-hide-value" :modeSwitcher="false"></ejs-
colorpicker>
</div>
</template>
<script>
import Vue from 'vue';

```

```

import { ColorPickerPlugin } from '@syncfusion/ej2-vue-inputs';
import { enableRipple } from '@syncfusion/ej2-base';
enableRipple(true);
Vue.use(ColorPickerPlugin);
export default {}
</script>
<style>
@import '../node_modules/@syncfusion/ej2-base/styles/material.css';
@import '../node_modules/@syncfusion/ej2-buttons/styles/material.css';
@import '../node_modules/@syncfusion/ej2-popups/styles/material.css';
@import '../node_modules/@syncfusion/ej2-splitbuttons/styles/material.css';
@import '../node_modules/@syncfusion/ej2-inputs/styles/material.css';
.wrap {
  margin: 0 auto;
  width: 300px;
  text-align: center;
}
</style>

```

{% previewsample "page.domainurl/code-snippet/color-picker/default-cs6" %}

#### Custom handle

Color picker handle shape and UI can be customized. Here, we have customized the handle as **svg icon**. The same way you can customize the handle based on your requirement.

The following sample show the customized color picker handle.

#### APP.VUE

```

<template>
<div class='wrap'>
  <h4>Choose Color</h4>
  <ejs-colorpicker cssClass="e-custom-picker" value="#344aee"
:modeSwitcher="false" :open="onOpen"></ejs-colorpicker>
</div>
</template>
<script>
import Vue from 'vue';
import { ColorPickerPlugin } from '@syncfusion/ej2-vue-inputs';
import { enableRipple } from '@syncfusion/ej2-base';
enableRipple(true);
Vue.use(ColorPickerPlugin);
export default {
  methods: {
    onOpen: function(args) {
      args.element.querySelector('.e-handler').classList.add('e-icons');
    }
  }
}
</script>
<style>
@import '../node_modules/@syncfusion/ej2-base/styles/material.css';
@import '../node_modules/@syncfusion/ej2-buttons/styles/material.css';
@import '../node_modules/@syncfusion/ej2-popups/styles/material.css';
@import '../node_modules/@syncfusion/ej2-splitbuttons/styles/material.css';
@import '../node_modules/@syncfusion/ej2-inputs/styles/material.css';

```

```

.wrap {
  margin: 0 auto;
  width: 300px;
  text-align: center;
}
/* To hide the handle balloon preview */
.e-color-picker-tooltip.e-popup.e-popup-open {
  display: none;
}
/* Handle customization styles */
.e-custom-picker .e-container .e-hsv-container .e-handler {
  background: transparent
url('data:image/svg+xml;base64,PD94bWwgdmVyc2lvbj0iMS4wIiBlbmNvZGluZz0idXRmLTgiPz4KPCFetLSBHZW51cmF0b3I6IEFkb2JlIElsbHVzdHJhdG9yIDIyLjEuMCwgU1ZHIEV4cG9ydCBQbHVnLUluIC4gU1ZHIFZlcnNpb246IDYyMDAgQnVpbGQgMCkgIC0tPgo8c3ZnIHZlcnNpb249IjEuMSIgaWQ9IkkxeWVyXzEiIHhtbG5zPSJodHRwOi8vd3d3LnczLm9yZy8yMDAwL3N2ZyIgeG1sbmM6eGxpbnMs9Imh0dHA6Ly93d3d3cudzMub3JnLzE5OTkveGxpbnMsIiHg9IjBweCIgeT0iMHB4IgoJIHZpZXh0c3g9IjAgMCAxNiAxNiIgc3R5bGU9ImVuYWJsZS1iYWNRZ3JvdW5kOm5ldyAwIDAgMTYgMTY7IiB4bWw6c3BhY2U9InByZXNlcnZlIj4KPHN0eWxlIHR5cGU9InRleHQvY3NzIj4KCS5zdDB7ZmlsbDojRkZGRkZGO30KPC9zdHlsZT4KPGC+Cgk8cG9seWdvbiBjbGFzcz0ic3QwIiBwb2ludHM9IjE2LDYgMTAsNiAxMCwwIDYsMCA2LDYgMCw2IDAsMTAgNiwxMCA2LDE2IDEwLDE2IDEwLDEwIDE2LDEwIAkiLz4KPC9nPgo8cGF0aCBkPSJNMTAsNlYwSDZ2NkgwdjRoNnY2aDR2LTZoNlY2SDEweibNMtUsOUg5djZINlY5SDFWN2g2VjFoMnY2aDZWOXoiLz4KPC9zdmc+Cg==');
  font-size: 16px;
  height: 16px;
  line-height: 16px;
  margin-left: -8px;
  margin-top: -8px;
  border: none;
  box-shadow: none;
  width: 16px;
}
</style>

```

{% previewsample "page.domainurl/code-snippet/color-picker/default-cs7" %}

#### Custom primary button

By default, the applied color will be updated in primary button of the color picker. You can customize that as **icon**.

In the following sample, the **picker** icon is added to primary button and using [change](#) event the selected color will be updated in bottom portion of the icon.

#### APP.VUE

```

<template>
<div class='wrap'>
  <h4>Choose Color</h4>
  <ejs-colorpicker id="element" :change="onChange"></ejs-colorpicker>
</div>
</template>
<script>
import Vue from 'vue';
import { ColorPickerPlugin } from '@syncfusion/ej2-vue-inputs';
import { enableRipple } from '@syncfusion/ej2-base';
enableRipple(true);

```

```

Vue.use(ColorPickerPlugin);
export default {
  methods: {
    onChange: function(args) {
      document.querySelector('.e-colorpicker-wrapper .e-selected-color').style.borderBottomColor = args.currentValue.rgba;
    }
  },
  mounted: function() {
    var previewIcon = document.querySelector('.e-colorpicker-wrapper .e-selected-color');
    previewIcon.classList.add("e-icons");
  }
};
</script>
<style>
@import '../node_modules/@syncfusion/ej2-base/styles/material.css';
@import '../node_modules/@syncfusion/ej2-buttons/styles/material.css';
@import '../node_modules/@syncfusion/ej2-popups/styles/material.css';
@import '../node_modules/@syncfusion/ej2-splitbuttons/styles/material.css';
@import '../node_modules/@syncfusion/ej2-inputs/styles/material.css';
.wrap {
  margin: 0 auto;
  width: 300px;
  text-align: center;
}
/* Icon customization */
.e-colorpicker-wrapper #element+.e-split-btn-wrapper .e-split-btn .e-selected-color {
  background: none;
  border-bottom-style: solid;
  border-bottom-width: 3px;
  width: 14px;
  margin: 0px 2px;
  border-bottom-color: #008000;
}
.e-colorpicker-wrapper #element+.e-split-btn-wrapper .e-split-btn .e-selected-color .e-split-preview {
  display: none;
}
.e-colorpicker-wrapper #element+.e-split-btn-wrapper .e-split-btn .e-selected-color::before {
  content: '\e35c';
}
</style>

```

{% previewsample "page.domainurl/code-snippet/color-picker/default-cs8" %}

> The Essential JS 2 provides a set of icons that can be loaded by applying **e-icons** class name to the element. You can also use third party icon to customize the primary button.

#### Display hex code in input

The color picker input element can be showcased in the place of primary button. The applied color hex code will be updated in the primary button input.

The following sample shows the color picker with input.

**APP.VUE**

```

<template>
<div class='wrap'>
  <h4>Choose Color</h4>
  <ejs-colorpicker id="element" type="text" class="e-input"></ejs-
colorpicker>
</div>
</template>
<script>
import Vue from 'vue';
import { ColorPickerPlugin } from '@syncfusion/ej2-vue-inputs';
Vue.use(ColorPickerPlugin);
export default {
  mounted: function() {
    var target = document.getElementById("element");
    target.nextElementSibling.insertBefore(target,
target.nextElementSibling.children[1]);
  }
};
</script>
<style>
@import '../node_modules/@syncfusion/ej2-base/styles/material.css';
@import '../node_modules/@syncfusion/ej2-buttons/styles/material.css';
@import '../node_modules/@syncfusion/ej2-popups/styles/material.css';
@import '../node_modules/@syncfusion/ej2-splitbuttons/styles/material.css';
@import '../node_modules/@syncfusion/ej2-inputs/styles/material.css';
.wrap {
  margin: 0 auto;
  width: 300px;
  text-align: center;
}
/* Input element customization */
.e-colorpicker-wrapper .e-split-btn-wrapper #element.e-input {
  height: 16px;
  margin: 0;
  opacity: 1;
  position: initial;
  width: 75px;
}
/* To hide primary button */
.e-colorpicker-wrapper .e-split-btn-wrapper .e-split-btn {
  display: none;
}
/* Secondary button customization */
.e-colorpicker-wrapper .e-split-btn-wrapper .e-btn.e-dropdown-btn {
  background: transparent;
  border-color: transparent;
  border-bottom-color: rgba(0, 0, 0, 0.42);
}
.e-colorpicker-wrapper .e-split-btn-wrapper .e-input:focus+.e-btn.e-
dropdown-btn {
  padding-bottom: 3px;
  border-bottom-width: 2px;
  border-bottom-color: #e3165b;
}
.e-colorpicker-wrapper .e-split-btn-wrapper .e-btn.e-dropdown-btn .e-caret {

```

```

    transform: rotate(0deg);
    transition: transform 200ms ease-in-out;
  }
.e-colorpicker-wrapper .e-split-btn-wrapper .e-btn.e-dropdown-btn.e-active
.e-caret {
  transform: rotate(180deg);
}
</style>

```

```
{% previewsample "page.domainurl/code-snippet/color-picker/default-cs9" %}
```

### Custom UI

The color picker UI can be customized in all possible ways. The following sample shows the excel like UI customization with help of SplitButton and Dialog component. In that by clicking the more colors option from color palette, the dialog contains color picker will open.

### APP.VUE

```

<template>
<div class='wrap'>
  <ul id="target" tabindex="0">
    <li class="e-item e-palette-item">
      <ejs-colorpicker ref="palette" id="palette" mode="Palette"
:inline="true" :showButtons="false" :modeSwitcher="false"
:change="onPaletteChange"></ejs-colorpicker>
    </li>
    <li class="e-item" tabindex="-1">
      <span class="e-menu-icon"></span>
      More colors...
    </li>
  </ul>
  <h4>Select color</h4>
  <ejs-splitbutton id="splitbtn" ref="splitBtn" iconCss="e-icons e-font-
icon" target="#target" :open="onDdPopupOpen"
:beforeClose="onBeforeDdPopupClose"></ejs-splitbutton>
    <ejs-dialog id="pickerdlg" ref="pickerDlg" cssClass="e-dlg-picker"
:isModal="true" :target="target" :width="width" :height="height"
:visible="false" :animationSettings="animationSettings"
:content="pickerDlgContent" :open="pickerDlgOpen"
:overlayClick="closePickerDlg"></ejs-dialog>
  </div>
</template>
<script>
import Vue from 'vue';
import { ColorPickerPlugin } from '@syncfusion/ej2-vue-inputs';
import { SplitButtonPlugin } from '@syncfusion/ej2-vue-splitbuttons';
import { DialogPlugin } from '@syncfusion/ej2-vue-popups';
Vue.use(ColorPickerPlugin);
Vue.use(SplitButtonPlugin);
Vue.use(DialogPlugin);
var pickerDlgContent = Vue.component("demo", {
  template: '<div class="dialogContent"><ejs-colorpicker id="picker"
:inline="true" :modeSwitcher="false" :change="onPickerChange"></ejs-
colorpicker></div>',
  data() {
    return {

```

```

        data: {}
    };
},
methods: {
    onPickerChange: function(args) {
document.getElementById('splitbtn').children[0].style.borderBottomColor =
args.currentValue.rgba;
        document.getElementById('pickerdlg').ej2_instances[0].hide();
    }
}
});
export default {
    data() {
        return {
            target: ".wrap",
            width: "270px",
            height: "336px",
            animationSettings: { effect: 'Zoom' },
            pickerDlgContent: function () {
                return { template : pickerDlgContent }
            }
        };
    },
    methods: {
        onPaletteChange: function(args) {
document.getElementById('splitbtn').children[0].style.borderBottomColor =
args.currentValue.rgba;
        },
        onDdPopupOpen: function(args) {
            args.element.children[1].addEventListener('click',
this.openPickerDlg);
        },
        onBeforeDdPopupClose: function(args) {
            args.element.children[1].removeEventListener('click',
this.openPickerDlg);
        },
        pickerDlgOpen: function() {
            var colorPickerInst =
document.getElementById('picker').ej2_instances[0];
            colorPickerInst.refresh();
            colorPickerInst.element.nextElementSibling.querySelector('.e-ctrl-
btn .e-cancel').addEventListener('click', this.closePickerDlg);
        },
        openPickerDlg: function() {
            this.$refs.pickerDlg.ej2Instances.show();
        },
        closePickerDlg: function() {
            this.$refs.pickerDlg.ej2Instances.hide();
        }
    }
}
};
</script>
<style>
@import '../node_modules/@syncfusion/ej2-base/styles/material.css';
@import '../node_modules/@syncfusion/ej2-buttons/styles/material.css';

```

```

@import '../node_modules/@syncfusion/ej2-popups/styles/material.css';
@import '../node_modules/@syncfusion/ej2-splitbuttons/styles/material.css';
@import '../node_modules/@syncfusion/ej2-inputs/styles/material.css';
.wrap {
  margin: 0 auto;
  min-height: 345px;
  text-align: center;
}
/* Primary button icon preview */
.e-btn-icon.e-font-icon {
  border-bottom-style: solid;
  border-bottom-width: 3px;
}
/* Primary button icon */
.e-btn-icon.e-font-icon::before {
  content: '\e34c';
}
.e-colorpicker-wrapper.e-hide-palette {
  display: none;
}
.e-dropdown-popup ul .e-item:first-child.e-palette-item {
  height: auto;
  padding: 0;
}
.e-dlg-picker.e-dialog .e-dlg-content {
  padding: 0;
  background-color: transparent;
}
/* Sets ColorPicker height */
.e-dlg-picker.e-dialog {
  max-height: 336px !important;
  background-color: transparent;
}
/* More colors li icon customization */
.e-dropdown-popup ul .e-item:last-child .e-menu-icon {
  height: 24px;
  margin-top: 6px;
  width: 24px;
  background-image: linear-gradient(to bottom, #fff 0, #000 100%);
  background-color: #0450c2;
  background-blend-mode: hard-light;
}
</style>

```

{% previewsample "page.domainurl/code-snippet/color-picker/default-cs10" %}

### Handle no color support in Vue Color picker component

The ColorPicker component supports no color functionality. By clicking the no color tile from palette, the selected color becomes **empty** and considered as no color has been selected from color picker.

#### Default no color

To achieve this, set [noColor](#) property as **true**.

In the following sample, the first tile of the color palette represents the no color tile. By clicking the no color tile you can achieve the above functionalities.



**APP.VUE**

```

<template>
<div class='wrap'>
  <div id='preview'></div>
  <h4>Select Color</h4>
  <ejs-colorpicker id="element" value="#ba68c8" mode="Palette"
:noColor="true" :showButtons="false" :modeSwitcher="false"
:change="onChange"></ejs-colorpicker>
</div>
</template>
<script>
import Vue from 'vue';
import { ColorPickerPlugin } from '@syncfusion/ej2-vue-inputs';
import { enableRipple } from '@syncfusion/ej2-base';
enableRipple(true);
Vue.use(ColorPickerPlugin);
export default {
  methods: {
    onChange: function(args) {
      var preview = document.getElementById('preview');
      preview.style.backgroundColor = args.currentValue.hex;
      preview.textContent = args.currentValue.hex ? args.currentValue.hex :
'No color';
    }
  },
  mounted: function() {
    var preview = document.getElementById('preview');
    preview.style.backgroundColor = '#ba68c8';
    preview.textContent = '#ba68c8';
  }
};
</script>
<style>
@import '../node_modules/@syncfusion/ej2-base/styles/material.css';
@import '../node_modules/@syncfusion/ej2-buttons/styles/material.css';
@import '../node_modules/@syncfusion/ej2-popups/styles/material.css';
@import '../node_modules/@syncfusion/ej2-splitbuttons/styles/material.css';
@import '../node_modules/@syncfusion/ej2-inputs/styles/material.css';
.wrap {
  margin: 0 auto;
  width: 300px;
  text-align: center;
}
#preview {
  border: 1px solid;
  height: 40px;
  line-height: 40px;
}
h4, #preview {
  font-family: 'Helvetica Neue', 'Helvetica', 'Arial', 'sans-serif';
  font-size: 14px;
}
</style>

```

```
{% previewsample "page.domainurl/code-snippet/color-picker/default-cs12" %}
```

If the [noColor](#) property is enabled, make sure to disable the [modeswitcher](#) property.

#### Custom no color

The following sample show the color palette with custom no color option.

#### APP.VUE

```
<template>
<div class='wrap'>
  <ul id="target" tabindex="0">
    <li class="e-item e-palette-item">
      <ejs-colorpicker id="element" ref="colorPicker" value="#f44336"
mode="Palette" :inline="true" :columns="4" :showButtons="false"
:modeSwitcher="false" :presetColors="presets"
:beforeTileRender="beforeTileRender" :change="onChange"></ejs-colorpicker>
    </li>
    <li class="e-item" id="no-color" tabindex="-1">
      <span class="e-menu-icon e-nocolor"></span>
      No color
    </li>
  </ul>
  <div>
    <div id='preview'></div>
    <h4>Select Color</h4>
    <ejs-splitbutton id="splitbtn" ref="splitBtn" iconCss="e-cp-icons e-
picker-icon" target="#target"></ejs-splitbutton>
  </div>
</div>
</template>
<script>
import Vue from 'vue';
import { ColorPickerPlugin } from '@syncfusion/ej2-vue-inputs';
import { SplitButtonPlugin } from '@syncfusion/ej2-vue-splitbuttons';
Vue.use(ColorPickerPlugin);
Vue.use(SplitButtonPlugin);
export default {
  data() {
    return {
      presets: {
        'custom': ['#f44336', '#e91e63', '#9c27b0', '#673ab7', '#2196f3',
'#03a9f4', '#00bcd4', '#009688', '#8bc34a', '#cddc39',
'#ffeb3b', '#ffc107']
      }
    };
  },
  methods: {
    onChange: function(args) {
      document.querySelector(".e-split-btn .e-picker-
icon").style.borderBottomColor = args.currentValue.hex;
      var preview = document.getElementById('preview');
      var splitBtnObj = this.$refs.splitBtn.ej2Instances;
      preview.style.backgroundColor = args.currentValue.hex;
      preview.textContent = args.currentValue.hex;
      if (splitBtnObj.element.getAttribute("aria-expanded")) {
        splitBtnObj.toggle();
        splitBtnObj.element.focus();
      }
    }
  }
}
```

```

    },
    beforeTileRender: function(args) {
        args.element.classList.add('e-custom-tile');
    }
},
mounted: function() {
    var preview = document.getElementById('preview');
    preview.style.backgroundColor = '#ba68c8';
    preview.textContent = '#ba68c8';
    document.getElementById('no-color').onclick = function() {
        //sets color picker value property to null
        this.$refs.colorPicker.ej2Instances.setProperties({ 'value': '' },
true);
        document.querySelector('.e-split-btn .e-picker-
icon').style.borderBottomColor = 'transparent';
        preview.textContent = 'No color';
        preview.style.backgroundColor = 'transparent';
    }
}
};
</script>
<style>
@import '../node_modules/@syncfusion/ej2-base/styles/material.css';
@import '../node_modules/@syncfusion/ej2-buttons/styles/material.css';
@import '../node_modules/@syncfusion/ej2-popups/styles/material.css';
@import '../node_modules/@syncfusion/ej2-splitbuttons/styles/material.css';
@import '../node_modules/@syncfusion/ej2-inputs/styles/material.css';
/* Preview area styles */
#preview {
    border: 1px solid;
    height: 40px;
    line-height: 40px;
    width: 100%;
}
@font-face {
    font-family: 'paint';
    src:
url(data:application/x-font-ttf;charset=utf-
8;base64,AAEAAAAKAIAAAwAgTlMvMj0gSRIAAAEoAAAAVmNtYXdDnEOdVAAABiAAAADZnbHlmIZD
+uwAAAcgAAADMaGVhZBKhhHQAADQAAAAANmhoZWEHjANrAAAArAAAACRobXR4B+j/8wAAAYAAAAA
IbG9jYQBMAAAAAHAAAAABmlheHABDgBKAAABCAAAACBuYW1ln6hzswAAApQAAAINcG9zdEkLMmU
AAASkAAAAANgABAAADUv9qAFoEAP/z//4D6gABAAAAAAAAAAAAAAAAAAgABAAAAAQAAAZfc6F8
PPPUACwPoAAAAANfSn9kAAAAA19Kf2f/z//wD6gPhAAAAACAACAAAAAAAAAAEAAAACAD4AAgAAAAA
AAgAAAAoACgAAAP8AAAAAAAAAQp0AZAABQAAANoCvAAAAIwCegK8AAAB4AAxAQIAAAIABQMAAAA
AAAAAAAAAAAAAAAAAAAAAAAAUGZFZABA5wDnAANS/2oAWgPhAJYAAAABAAAAAAAAABAAAAAPo//M
AAAACAAAAAwAAABQAAwABAAAFAAEACIAAAAEAAQAAQAA5wD//wAA5wD//wAAAAEABAAAAEAAAA
AAAAAZgAAAAAL/8//8A+oD4QAKAD0AAAEWBgcATc1JiQHJTMmNjceARcVJx4BBx4BFQ4BIiYnNDY
3PgEvAS4BIw4BBwEGHgI3AT4BLwE1LgEnDgEDEiRlCgulCxP+8RT+GyYDQFxoZQwTBQEDDxEBJzo
nAREOCqkPJQ4cDBcdAf6oG1a3nx8BWQ4RHKAdeG1oWwHTLHVwYVml6Kx1BHEqfwYFqWUHEX4tDAo
cEx0nJx0RHgoVUDQpDgsBFAH+px2guFUaAVkNOiCgCXnhCAWOAAAAAAAAAAEgDeAAEAAAAAAAAQA
AAEAAAAAAAAEABQABAAEAAAAAAAAIABwAGAAEAAAAAAAAAMABQANAAEAAAAAAAAQABQASAAEAAAAAAU
ACwAXAAEAAAAAAAYABQAIAAEAAAAAAAOALAAANAAEAAAAAAASAEgBTAAMAAQQJAAAAAgBIAAMAAQQ
JAAEACgBnAAMAAQQJAAIADgBxAAMAAQQJAAAMACgB/AAMAAQQJAAQACgCJAAMAAQQJAAUAFgCTAAM
AAQQJAAAYACgCpAAMAAQQJAAoAWACzAAMAAQQJAAASAJAELIHBhaW50UmVndWxhcncBhaW50cGFpbncR
WZXJzaW9uIDEuMHBhaW50Rm9udCBnZW51cmF0ZWQgdXNpbmcgU3luY2Zlc2lvbiBNZXRYbyBTdHV
kaW93d3cuc3luY2Zlc2lvbi5jb20AIABwAGEAaQBuaHQAUgBlAGcAdQBsAGEACgBwAGEAaQBuaHQ
ACABhAGkAbgB0AFYAZQByAHMAaQBvAG4AIAAxAAC4AMABwAGEAaQBuaHQARgBvAG4AdAAgAGcAZQB

```

```

uAGUACgBhAQZQBkACAAAdQBzAGkAbgBnACAAUwB5AG4AYwBmAUAcwBpAG8AbgAgAE0AZQB0AHT
AbwAgAFMAdABlAGQAAQBvAHcAdwB3AC4AcwB5AG4AYwBmAUAcwBpAG8AbgAuAGMAbwBtAAAAAAAI
AAAAAAAAAACgAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAGCAQMADHBhaW50LWJlY2tldAAAAAA=)
format('truetype');
font-weight: normal;
font-style: normal;
}
.e-cp-icons {
font-family: 'paint' !important;
speak: none;
font-size: 55px;
font-style: normal;
font-weight: normal;
font-variant: normal;
text-transform: none;
line-height: 1;
-webkit-font-smoothing: antialiased;
-moz-osx-font-smoothing: grayscale;
}
.wrap {
margin: 0 auto;
width: 300px;
text-align: center;
}
.e-dropdown-popup ul#target {
padding: 0;
}
.e-dropdown-popup ul .e-item.e-palette-item {
height: auto;
padding: 0;
}
.e-btn-icon.e-picker-icon {
border-bottom-color: #f44336;
border-bottom-style: solid;
border-bottom-width: 3px;
}
/* Picker icon */
.e-btn-icon.e-picker-icon::before {
content: '\e700';
}
/* No color li styles */
.e-dropdown-popup ul .e-item .e-menu-icon.e-nocolor {
height: 22px;
margin-top: 8px;
width: 22px;
background: transparent
}
url('data:image/svg+xml;base64,PD94bWwgdmlld0JveD0iMCAwIDYgNiIgdmVyc
TgiPz4KPHN2ZyB3aWR0aD0iNnB4IiBoZWlnaHQ9IjZweCIgdmlld0JveD0iMCAwIDYgNiIgdmVyc
21vbG90iMS4xIiB4bWxucz0iaHR0cDovL3d3dy53My5vcmcvMjAwMC9zdmciIHhtbG5zOnhsaW5rP
SJodHRwOi8vd3d3LnczLm9yZy8xOTk5L3hsaW5rIj4KICAgIDwhLS0gr2VuZXJhdG9yOiBTa2V0Y
2ggNTAgKDU0OTgzKSAatIGh0dHA6Ly93d3cuYm9oZW1pYW5jb2RpbmcuY29tL3NrZXRjaCAatLT4KI
CAgIDx0aXRzT5Hcm9lcCA5PC90aXRzT4KICAgIDxkZXNjPkNyZWZ0ZWQgd2l0aCBTa2V0Y2guP
C9kZXNjPgogICAgPGRlZnM+PC9kZWZzPgogICAgPGcgaWQ9IiBhZ2UtMSIgc3Ryb2t1PSJub251I
iBzdHJva2Utd2lkdGg9IjEiIGZpbGw9Im5vbWUiIGZpbGwtcnVsZT0iZXZlbnM9kZCI+CiAgICAgI
CAgPGcgaWQ9Ikd3b3VwLTkiPgogICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIC
WxsPSIjRTBFMEUwIiB4PSIwIiB5PSIwIiB3aWR0aD0iMyIgaGVpZ2h0PSIzIj48L3JlY3Q+CiAgI
CAgICAgICAgIDxyZWNOIGlkpSJSZWN0YW5nbGUtMTETQ29weS0yIiBmaWxsPSIjRkZGRkZGIiB4P

```

```

SIwIiB5PSIzIiB3aWR0aD0iMyIgaGVpZ2h0PSIzIj48L3JlY3Q+CiAgICAgICAgICAgIDxyZWNOI
GlkPSJSZWN0YW5nbGUtMTETQ29weSIgZmlsbD0iI0ZGRkZGRiIgeD0iMyIgeT0iMCIgd2lkdGg9I
jMiIGhlaWdodD0iMyI+PC9yZWNOPgogICAgICAgICAgICA8cmVjdCBpZD0iUmVjdGFuZ2xlLTExL
UNvcHktMyIgaGVpZ2h0PSIzIj48L3JlY3Q+CiAgICAgICAgICAgIDwvZz4KICAgIDwvZz4KPC9zdmc+');
}
/* Tile customization */
.e-container.e-palette.e-tile.e-custom-tile {
  height: 24px;
  width: 24px;
  margin: 4px;
}
h4, #preview {
  font-family: 'Helvetica Neue', 'Helvetica', 'Arial', 'sans-serif';
  font-size: 14px;
}
</style>

```

{% previewsample "page.domainurl/code-snippet/color-picker/default-cs13" %}

### Disabled in Vue Color picker component

To achieve disabled state in ColorPicker, set the [disabled](#) property to `true`. The ColorPicker pop-up cannot be accessed in disabled state.

The following example shows the `disabled` state of ColorPicker component.

### APP.VUE

```

<template>
<div class='wrap'>
  <h4>Choose Color</h4>
  <ejs-colorpicker :disabled="true"></ejs-colorpicker>
</div>
</template>
<script>
import Vue from 'vue';
import { ColorPickerPlugin } from '@syncfusion/ej2-vue-inputs';
import { enableRipple } from '@syncfusion/ej2-base';
enableRipple(true);
Vue.use(ColorPickerPlugin);
export default {}
</script>
<style>
@import '../node_modules/@syncfusion/ej2-base/styles/material.css';
@import '../node_modules/@syncfusion/ej2-buttons/styles/material.css';
@import '../node_modules/@syncfusion/ej2-popups/styles/material.css';
@import '../node_modules/@syncfusion/ej2-splitbuttons/styles/material.css';
@import '../node_modules/@syncfusion/ej2-inputs/styles/material.css';
.wrap {
  margin: 0 auto;
  width: 300px;
  text-align: center;
}
</style>

```

```
{% previewsample "page.domainurl/code-snippet/color-picker/default-cs11" %}
```

## ComboBox

### Getting Started with the Vue Combo box Component in Vue 2

This article provides a step-by-step guide for setting up a Vue 2 project using [Vue-CLI](#) and integrating the Syncfusion Vue Combo box component using the [Composition API](#) / [Options API](#)[Link to the Video](#).

To get start quickly with ComboBox component using Vue CLI, you can check on this video:

Prerequisites

[System requirements for Syncfusion Vue UI components](#)

#### Setting up the Vue 2 project

To generate a Vue 2 project using Vue-CLI, use the [vue create](#) command. Follow these steps to install Vue CLI and create a new project:

```
`bash
```

```
npm install -g @vue/cli
```

```
vue create quickstart
```

```
cd quickstart
```

```
npm run serve
```

```
,
```

or

```
`bash
```

```
yarn global add @vue/cli
```

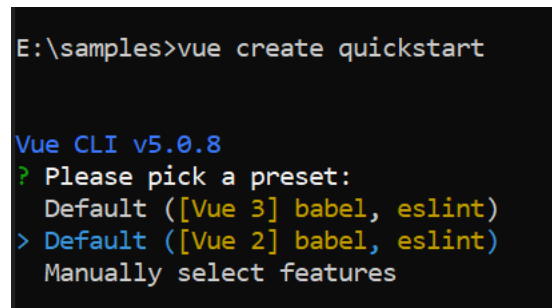
```
vue create quickstart
```

```
cd quickstart
```

```
yarn run serve
```

```
,
```

When creating a new project, choose the option **Default ([Vue 2] babel, eslint)** from the menu.



```
E:\samples>vue create quickstart

Vue CLI v5.0.8
? Please pick a preset:
  Default ([Vue 3] babel, eslint)
> Default ([Vue 2] babel, eslint)
  Manually select features
```

Once the **quickstart** project is set up with default settings, proceed to add Syncfusion components to the project

### Add Syncfusion Vue packages

Syncfusion packages are available at [npmjs.com](https://www.npmjs.com). To use Vue components, install the required npm package.

This article uses the [Vue Combo box component](#) as an example. Install the `@syncfusion/ej2-vue-dropdowns` package by running the following command:

```
`bash
```

```
npm install @syncfusion/ej2-vue-dropdowns --save
```

```
,
```

or

```
`bash
```

```
yarn add @syncfusion/ej2-vue-dropdowns
```

```
,
```

### Import Syncfusion CSS styles

You can import themes for the Syncfusion Vue component in various ways, such as using CSS or SASS styles from npm packages, CDN, [CRG](#) and [Theme Studio](#). Refer to [themes topic](#) to know more about built-in themes and different ways to refer to themes in a Vue project.

In this article, the `Material` theme is applied using CSS styles, which are available in installed packages. The necessary `Material` CSS styles for the Combo box component and its dependents were imported into the `<style>` section of `src/App.vue` file.

#### ~/SRC/APP.VUE

```
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-inputs/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-dropdowns/styles/material.css";
</style>
```

### Add Syncfusion Vue component

Follow the below steps to add the Vue Combo box component using `Composition API` or `Options API`:

1\ First, import and register the Combo box component in the `script` section of the `src/App.vue` file. If you are using the `Composition API`, you should add the `setup` attribute to the `script` tag to indicate that Vue will be using the `Composition API`.

#### COMPOSITION API (~/SRC/APP.VUE)

```
<script setup>
import { ComboBoxComponent as EjsCombobox } from "@syncfusion/ej2-vue-
dropdowns";
</script>
```

#### OPTIONS API (~/SRC/APP.VUE)

```
<script>
import { ComboBoxComponent } from "@syncfusion/ej2-vue-dropdowns";
```

```
export default {
  components: {
    'ejs-combobox': ComboBoxComponent
  }
}
</script>
```

2\ In the **template** section, define the Combo box component with the [dataSource](#) and [placeholder](#) property.

### ~/SRC/APP.VUE

```
<template>
<div id="app">
<div id='container' style="margin:50px auto 0; width:250px;">
<br>
<ejs-combobox id='combobox' :dataSource='sportsData' placeholder='Select a
game'></ejs-combobox>
</div>
</div>
</template>
```

### Binding data source

After initialization, populate the ComboBox with data using the **dataSource** property. Here, an array of string values is passed to the ComboBox component.

### COMPOSITION API (~/SRC/APP.VUE)

```
<template>
<div id="app">
<div id='container' style="margin:50px auto 0; width:250px;">
<br>
<ejs-combobox id='combobox' :dataSource='sportsData' placeholder='Select a
game'></ejs-combobox>
</div>
</div>
</template>
<script setup>
import { ComboBoxComponent as EjsCombobox } from "@syncfusion/ej2-vue-
dropdowns";
const sportsData = ['Badminton', 'Cricket', 'Football', 'Golf', 'Tennis'];
</script>
```

### OPTIONS API (~/SRC/APP.VUE)

```
<template>
<div id="app">
<div id='container' style="margin:50px auto 0; width:250px;">
<br>
<ejs-combobox id='combobox' :dataSource='sportsData' placeholder='Select a
game'></ejs-combobox>
</div>
</div>
</template>
<script>
```



```
import { ComboBoxComponent } from "@syncfusion/ej2-vue-dropdowns";
export default {
  components: {
    'ejs-combobox': ComboBoxComponent
  },
  data () {
    return {
      sportsData: ['Badminton', 'Cricket', 'Football', 'Golf', 'Tennis']
    }
  }
}
</script>
```

Here is the summarized code for the above steps in the **src/App.vue** file:

### **COMPOSITION API (~SRC/APP.VUE)**

```
<template>
  <div id="app">
    <div id='container' style="margin:50px auto 0; width:250px;">
      <br>
      <ejs-combobox id='combobox' :dataSource='sportsData'
placeholder='Select a game'></ejs-combobox>
    </div>
  </div>
</template>
<script setup>
import { ComboBoxComponent as EjsCombobox } from "@syncfusion/ej2-vue-
dropdowns";
const sportsData = ['Badminton', 'Cricket', 'Football', 'Golf', 'Tennis'];
</script>
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-inputs/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-dropdowns/styles/material.css";
</style>
```

### **OPTIONS API (~SRC/APP.VUE)**

```
<template>
  <div id="app">
    <div id='container' style="margin:50px auto 0; width:250px;">
      <br>
      <ejs-combobox id='combobox' :dataSource='sportsData'
placeholder='Select a game'></ejs-combobox>
    </div>
  </div>
</template>
<script>
import { ComboBoxComponent } from "@syncfusion/ej2-vue-dropdowns";
export default {
  components: {
    'ejs-combobox': ComboBoxComponent
  },
  data () {
    return {
```

```

        sportsData: ['Badminton', 'Cricket', 'Football', 'Golf', 'Tennis']
      }
    }
  }
</script>
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-inputs/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-dropdowns/styles/material.css";
</style>

```

### Run the application

To run the application, use the following command:

```
`bash
```

```
npm run serve
```

```
,
```

or

```
`bash
```

```
yarn run serve
```

```
,
```

```
{% previewsample "page.domainurl/code-snippet/combobox/getting-started/getting-started-cs1" %}
```

### Custom values

The ComboBox allows the user to give input as custom value which is not required to present in predefined set of values. By default, this support is enabled by [allowCustom](#) property. In this case, both text field and value field considered as same. The custom value will be sent to post back handler when a form is about to be submitted.

### COMPOSITION API (~SRC/APP.VUE)

```

<template>
  <div id="app">
    <div id='container' style="margin:50px auto 0; width:250px;">
      <br>
      <ejs-combobox id='combobox' :dataSource='sportsData'
:fields='fields' :allowCustom='allowCustom' placeholder='Select a
game'></ejs-combobox>
    </div>
  </div>
</template>
<script setup>
import { ComboBoxComponent as EjsCombobox } from "@syncfusion/ej2-vue-
dropdowns";
const sportsData = [
  { Id: 'game1', Game: 'Badminton' },
  { Id: 'game2', Game: 'Football' },
  { Id: 'game3', Game: 'Tennis' }];
const fields = { text: 'Game', value: 'Id' };
const allowCustom = true;

```

```
</script>
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-inputs/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-dropdowns/styles/material.css";
</style>
```

### OPTIONS API (~SRC/APP.VUE)

```
<template>
  <div id="app">
    <div id='container' style="margin:50px auto 0; width:250px;">
      <br>
      <ejs-combobox id='combobox' :dataSource='sportsData'
:fields='fields' :allowCustom='allowCustom' placeholder='Select a
game'></ejs-combobox>
    </div>
  </div>
</template>
<script>
import { ComboBoxComponent } from "@syncfusion/ej2-vue-dropdowns";
export default {
  components: {
    'ejs-combobox': ComboBoxComponent
  },
  data () {
    return {
      sportsData: [
        { Id: 'game1', Game: 'Badminton' },
        { Id: 'game2', Game: 'Football' },
        { Id: 'game3', Game: 'Tennis' }
      ],
      fields: { text: 'Game', value: 'Id' },
      allowCustom: true
    }
  }
}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-inputs/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-dropdowns/styles/material.css";
</style>
```

{% previewsample "page.domainurl/code-snippet/combobox/getting-started/custom-cs1" %}

### Configure the popup list

By default, the width of the popup list automatically adjusts according to the ComboBox input element's width, and the height of the popup list has '300px'.

The height and width of the popup list can also be customized using the [popupHeight](#) and [popupWidth](#) property respectively.

In the following sample, popup list's width and height have configured.

### COMPOSITION API (~SRC/APP.VUE)

```
<template>
  <div id="app">
    <div id='container' style="margin:50px auto 0; width:250px;">
      <br>
      <ejs-combobox id='combobox' :dataSource='sportsData'
popupHeight="200px" popupWidth="250px" placeholder="select a game"></ejs-
combobox>
    </div>
  </div>
</template>
<script setup>
import { ComboBoxComponent as EjsCombobox } from "@syncfusion/ej2-vue-
dropdowns";
const sportsData = ['Badminton', 'Basketball', 'Cricket', 'Football',
'Golf', 'Hockey', 'Rugby', 'Snooker', 'Tennis'];
</script>
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-inputs/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-dropdowns/styles/material.css";
</style>
```

### OPTIONS API (~SRC/APP.VUE)

```
<template>
  <div id="app">
    <div id='container' style="margin:50px auto 0; width:250px;">
      <br>
      <ejs-combobox id='combobox' :dataSource='sportsData'
popupHeight="200px" popupWidth="250px" placeholder="select a game"></ejs-
combobox>
    </div>
  </div>
</template>
<script>
import { ComboBoxComponent } from "@syncfusion/ej2-vue-dropdowns";
export default {
  components: {
    'ejs-combobox': ComboBoxComponent
  },
  data () {
    return {
      sportsData: ['Badminton', 'Basketball', 'Cricket', 'Football', 'Golf',
'Hockey', 'Rugby', 'Snooker', 'Tennis']
    }
  }
}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-inputs/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-dropdowns/styles/material.css";
</style>
```

{% previewsample "page.domainurl/code-snippet/combobox/getting-started/popup-cs2" %}

You can refer to our [Vue ComboBox](#) feature tour page for its groundbreaking feature representations.

See Also

- [How to bind the data](#)

## Getting Started with the Vue ComboBox Component in Vue 3

This article provides a step-by-step guide for setting up a [Vite](#) project with a JavaScript environment and integrating the Syncfusion Vue ComboBox component using the [Composition API](#) / [Options API](#).

The **Composition API** is a new feature introduced in Vue.js 3 that provides an alternative way to organize and reuse component logic. It allows developers to write components as functions that use smaller, reusable functions called composition functions to manage their properties and behavior.

The **Options API** is the traditional way of writing Vue.js components, where the component logic is organized into a series of options that define the component's properties and behavior. These options include data, methods, computed properties, watchers, lifecycle hooks, and more.

### Prerequisites

#### [System requirements for Syncfusion Vue UI components](#)

#### Set up the Vite project

A recommended approach for beginning with Vue is to scaffold a project using [Vite](#). To create a new Vite project, use one of the commands that are specific to either NPM or Yarn.

```
`bash
```

```
npm create vite@latest
```

```
,
```

or

```
`bash
```

```
yarn create vite
```

```
,
```

Using one of the above commands will lead you to set up additional configurations for the project as below:

1. Define the project name: We can specify the name of the project directly. Let's specify the name of the project as **my-project** for this article.

```
`bash
```

```
? Project name: » my-project
```

```
,
```

2. Select **Vue** as the framework. It will create a Vue 3 project.

```
`bash
```

```
? Select a framework: » - Use arrow-keys. Return to submit.
```

Vanilla

Vue

React

Preact

Lit

Svelte

Others

,

3. Choose **JavaScript** as the framework variant to build this Vite project using JavaScript and Vue.

``bash`

? Select a variant: » - Use arrow-keys. Return to submit.

JavaScript

TypeScript

Customize with create-vue ↗

Nuxt ↗

,

4. Upon completing the aforementioned steps to create the **my-project**, run the following command to install its dependencies:

``bash`

`cd my-project`

`npm install`

,

or

``bash`

`cd my-project`

`yarn install`

,

Now that **my-project** is ready to run with default settings, let's add Syncfusion components to the project.

#### Add Syncfusion Vue packages

Syncfusion Vue component packages are available at [npmjs.com](https://www.npmjs.com). To use Syncfusion Vue components in the project, install the corresponding npm package.

This article uses the [Vue ComboBox component](#) as an example. To use the Vue ComboBox component in the project, the `@syncfusion/ej2-vue-dropdowns` package needs to be installed using the following command:

``bash`

```
npm install @syncfusion/ej2-vue-dropdowns --save
```

```
,
```

or

```
`bash
```

```
yarn add @syncfusion/ej2-vue-dropdowns
```

```
,
```

### Import Syncfusion CSS styles

You can import themes for the Syncfusion Vue component in various ways, such as using CSS or SASS styles from npm packages, CDN, [CRG](#) and [Theme Studio](#). Refer to [themes topic](#) to know more about built-in themes and different ways to refer to themes in a Vue project.

In this article, **Material** theme is applied using CSS styles, which are available in installed packages. The necessary **Material** CSS styles for the ComboBox component and its dependents were imported into the `<style>` section of **src/App.vue** file.

#### ~/SRC/APP.VUE

```
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-inputs/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-dropdowns/styles/material.css";
</style>
```

The order of importing CSS styles should be in line with its dependency graph.

### Add Syncfusion Vue component

Follow the below steps to add the Vue ComboBox component using **Composition API** or **Options API**:

1.First, import and register the ComboBox component in the **script** section of the **src/App.vue** file. If you are using the **Composition API**, you should add the **setup** attribute to the **script** tag to indicate that Vue will be using the **Composition API**.

#### COMPOSITION API (~/SRC/APP.VUE)

```
<script setup>
import { ComboBoxComponent as EjsCombobox } from "@syncfusion/ej2-vue-
dropdowns";
</script>
```

#### OPTIONS API (~/SRC/APP.VUE)

```
<script>
import { ComboBoxComponent } from "@syncfusion/ej2-vue-dropdowns";
//Component registration
export default {
  name: "App",
  components: {
    'ejs-combobox' : ComboBoxComponent,
  }
}
```

```
</script>
```

2. In the **template** section, define the ComboBox component with the [dataSource](#) property and column definitions.

### ~/SRC/APP.VUE

```
<template>
<div class="control_wrapper">
<ejs-combobox id='combobox' :dataSource='sportsData' placeholder='Select a
game'></ejs-combobox>
</div>
</template>
```

Here is the summarized code for the above steps in the **src/App.vue** file:

### COMPOSITION API (~/SRC/APP.VUE)

```
<template>
<div class="control_wrapper">
<ejs-combobox id='combobox' :dataSource='data[0].sportsData'
placeholder='Select a game'></ejs-combobox>
</div>
</template>
<script setup>
import { ComboBoxComponent as EjsCombobox } from "@syncfusion/ej2-vue-
dropdowns";
const data = [{ waterMark : 'e.g. Basketball',
sportsData: ['Badminton', 'Cricket', 'Football', 'Golf', 'Tennis'] }]
</script>
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-inputs/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-dropdowns/styles/material.css";
</style>
```

### OPTIONS API (~/SRC/APP.VUE)

```
<template>
<div class="control_wrapper">
<ejs-combobox id='combobox' :dataSource='sportsData' placeholder='Select a
game'></ejs-combobox>
</div>
</template>
<script>
import { ComboBoxComponent } from "@syncfusion/ej2-vue-dropdowns";
//Component registration
export default {
name: 'App',
components: {
"ejs-combobox": ComboBoxComponent
},
data () {
return {
waterMark : 'e.g. Basketball',
```



```
sportsData: ['Badminton', 'Cricket', 'Football', 'Golf', 'Tennis']
}
}
}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-inputs/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-dropdowns/styles/material.css";
</style>
```

### Run the project

To run the project, use the following command:

```
`bash
```

```
npm run dev
```

```
,
```

or

```
`bash
```

```
yarn run dev
```

```
,
```

### Custom values

The ComboBox allows the user to give input as custom value which is not required to present in predefined set of values. By default, this support is enabled by [allowCustom](#) property. In this case, both text field and value field considered as same.

The custom value will be sent to post back handler when a form is about to be submitted.

### COMPOSITION API (~SRC/APP.VUE)

```
<template>
<div id="app">
<div id='container' style="margin:50px auto 0; width:250px;">
<br>
<ejs-combobox id='combobox' :dataSource='data[0].sportsData'
:fields='data[0].fields' :allowCustom='data[0].allowCustom'
placeholder='Select a game'></ejs-combobox>
</div>
</div>
</template>
<script setup>
import { ComboBoxComponent as EjsCombobox } from "@syncfusion/ej2-vue-
dropdowns";
const data = [{ sportsData: [{ Id: 'game1', Game: 'Badminton' },
{ Id: 'game2', Game: 'Football' },
{ Id: 'game3', Game: 'Tennis' } ]},
fields: { text: 'Game', value: 'Id' },
allowCustom: true}]
</script>
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
```

```
@import "../node_modules/@syncfusion/ej2-inputs/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-dropdowns/styles/material.css";
</style>
```

### OPTIONS API (~SRC/APP.VUE)

```
<template>
<div id="app">
<div id='container' style="margin:50px auto 0; width:250px;">
<br>
<ejs-combobox id='combobox' :dataSource='sportsData' :fields='fields'
:allowCustom='allowCustom' placeholder='Select a game'></ejs-combobox>
</div>
</div>
</template>
<script>
import { ComboBoxComponent } from "@syncfusion/ej2-vue-dropdowns";
//Component registration
export default {
name: 'App',
components: {
"ejs-combobox": ComboBoxComponent
},
data () {
return {
sportsData: [
{ Id: 'game1', Game: 'Badminton' },
{ Id: 'game2', Game: 'Football' },
{ Id: 'game3', Game: 'Tennis' }
],
fields: { text: 'Game', value: 'Id' },
allowCustom: true
}
}
}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-inputs/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-dropdowns/styles/material.css";
</style>
```

### Configure the popup list

By default, the width of the popup list automatically adjusts according to the ComboBox input element's width, and the height of the popup list has '300px'.

The height and width of the popup list can also be customized using the [popupHeight](#) and [popupWidth](#) property respectively.

In the following sample, popup list's width and height have configured.

### APP.VUE

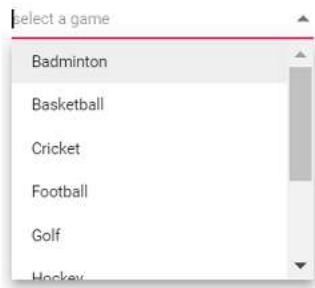
```
<template>
<div id="app">
<div id='container' style="margin:50px auto 0; width:250px;">
```

```

        <br>
        <ejs-combobox id='combobox' :dataSource='sportsData'
popupHeight="200px" popupWidth="250px" placeholder="select a game"></ejs-
combobox>
    </div>
</div>
</template>
<script>
    import { ComboBoxComponent } from "@syncfusion/ej2-vue-dropdowns";
    //Component registration
    export default {
        name: 'App',
        components: {
            "ejs-combobox": ComboBoxComponent
        },
        data () {
            return {
                sportsData: ['Badminton', 'Basketball', 'Cricket',
'Football', 'Golf', 'Hockey', 'Rugby', 'Snooker', 'Tennis']
            }
        }
    }
</script>
<style>
    @import "../node_modules/@syncfusion/ej2-base/styles/material.css";
    @import "../node_modules/@syncfusion/ej2-inputs/styles/material.css";
    @import "../node_modules/@syncfusion/ej2-vue-
dropdowns/styles/material.css";
</style>

```

Output be like the below.



See Also

- [How to bind the data](#)

## Data binding in Vue Combo box component

The ComboBox loads the data either from local data sources or remote data services using the [dataSource](#) property. It supports the data type of [array](#) or [DataManager](#).

The ComboBox also supports different kinds of data services such as OData, OData V4, and Web API, and data formats such as XML, JSON, and JSONP with the help of [DataManager](#) adaptors.

| Fields | Type | Description |

|-----|-----|-----|

| text | **string** | Specifies the display text of each list item. |

| value | **number or string** | Specifies the hidden data value which mapped to each list item that should be unique. |

| groupBy | **string** | Specifies the category under which the list item needs to be grouped. |

| iconCss | **string** | Specifies the icon class of each list item. |

When binding complex data to the ComboBox, fields should be mapped correctly. Otherwise, the selected item remains undefined.

### Binding local data

Local data can be represented in two ways as described below.

#### 1. Array of simple data

The ComboBox has support to load array of primitive data such as strings and numbers. Here, both value and text field act the same.

### APP.VUE

```
<template>
  <div id="app">
    <div id='container' style="margin:50px auto 0; width:250px;">
      <br>
      <ejs-combobox id='combobox' :dataSource='sportsData'
placeholder="Select a game"></ejs-combobox>
    </div>
  </div>
</template>
<script>
import Vue from 'vue';
import { ComboBoxPlugin } from "@syncfusion/ej2-vue-dropdowns";
Vue.use(ComboBoxPlugin);
export default {
  data () {
    return {
      sportsData: ['Badminton', 'Cricket', 'Football', 'Golf', 'Tennis']
    }
  }
}
</script>
<style>
@import
"https://ej2.syncfusion.com/vue/documentation/../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-inputs/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-dropdowns/styles/material.css";
</style>
```

{% previewsample "page.domainurl/code-snippet/combobox/data-binding/simple-cs1" %}

## 2. Array of JSON data

The ComboBox can generate its list items through an array of complex data. For this, the appropriate columns should be mapped to the [fields](#) property.

In the following example, **Id** column and **Game** column from complex data have been mapped to the **value** field and **text** field, respectively.

### APP.VUE

```
<template>
  <div id="app">
    <div id='container' style="margin:50px auto 0; width:250px;">
      <br>
      <ejs-combobox id='combobox' :dataSource='sportsData'
:fields='fields' placeholder="Select a game"></ejs-combobox>
    </div>
  </div>
</template>
<script>
import Vue from 'vue';
import { ComboBoxPlugin } from "@syncfusion/ej2-vue-dropdowns";
Vue.use(ComboBoxPlugin);
export default {
  data () {
    return {
      sportsData: [
        { Id: 'game1', Game: 'Badminton' },
        { Id: 'game2', Game: 'Football' },
        { Id: 'game3', Game: 'Tennis' }
      ],
      fields : { text: 'Game', value: 'Id' }
    }
  }
}
</script>
<style>
@import
"https://ej2.syncfusion.com/vue/documentation/../../node_modules/@syncfusion/ej
2-base/styles/material.css";
@import "../../node_modules/@syncfusion/ej2-inputs/styles/material.css";
@import "../../node_modules/@syncfusion/ej2-vue-
dropdowns/styles/material.css";
</style>
```

{% previewsample "page.domainurl/code-snippet/combobox/data-binding/json-cs1" %}

## 3. Array of Complex data

The ComboBox can generate its list items through an array of complex data. For this, the appropriate columns should be mapped to the [fields](#) property.

In the following example, **Code.Id** column and **Country.Name** column from complex data have been mapped to the **value** field and **text** field, respectively.

### APP.VUE

```
<template>
```

```

<div id="app">
  <div id='container' style="margin:50px auto 0; width:250px;">
    <br>
    <ejs-combobox id='combobox' :dataSource='countriesData'
:fields='fields' placeholder="Select a Country"></ejs-combobox>
  </div>
</div>
</template>
<script>
import Vue from 'vue';
import { ComboBoxPlugin } from "@syncfusion/ej2-vue-dropdowns";
Vue.use(ComboBoxPlugin);
export default {
  data () {
    return {
      countriesData: [
        { Country: { Name: 'Australia' }, Code: { Id: 'AU' } },
        { Country: { Name: 'Bermuda' }, Code: { Id: 'BM' } },
        { Country: { Name: 'Canada' }, Code: { Id: 'CA' } },
        { Country: { Name: 'Cameroon' }, Code: { Id: 'CM' } },
        { Country: { Name: 'Denmark' }, Code: { Id: 'DK' } },
        { Country: { Name: 'France' }, Code: { Id: 'FR' } }
      ],
      fields : { text: 'Country.Name', value: 'Code.Id' }
    }
  }
}
</script>
<style>
@import
"https://ej2.syncfusion.com/vue/documentation/../../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../../node_modules/@syncfusion/ej2-inputs/styles/material.css";
@import "../../node_modules/@syncfusion/ej2-vue-
dropdowns/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/combobox/data-binding/complex-cs1" %}

### Binding remote data

The ComboBox supports retrieval of data from remote data services with the help of **DataManager** component. The [Query](#) property is used to fetch data from the database and bind it to the ComboBox.

The following sample displays the first 6 contacts from “Customers” table of the **Northwind** Data Service.

### APP.VUE

```

<template>
  <div id="app">
    <div id='container' style="margin:50px auto 0; width:250px;">
      <br>
      <ejs-combobox id='combobox' :dataSource='customerData'
:query='query' :fields='fields' sortOrder='Ascending' placeholder="Select a
Customer"></ejs-combobox>
    </div>
  </div>

```

```

</div>
</template>
<script>
import Vue from 'vue';
import { ComboBoxPlugin } from "@syncfusion/ej2-vue-dropdowns";
Vue.use(ComboBoxPlugin);
import { Query, DataManager, ODataV4Adaptor } from '@syncfusion/ej2-data';
export default {
  data () {
    return {
      customerData : new DataManager({
        url: 'https://services.odata.org/V4/Northwind/Northwind.svc/',
        adaptor: new ODataV4Adaptor,
        crossDomain: true
      }),
      query : new Query().from('Customers').select(['ContactName',
'CustomerID']).take(6) ,
      fields : { text: 'ContactName', value: 'CustomerID' } ,
    }
  }
}
</script>
<style>
@import "../../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../../node_modules/@syncfusion/ej2-inputs/styles/material.css";
@import "../../node_modules/@syncfusion/ej2-vue-
dropdowns/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/combobox/data-binding/remote-cs1" %}

See Also

- [How to achieve cascading](#)
- [How to load data using template](#)
- [How to group the data using header](#)
- [How to filter the bound dataLink to the Video](#)

## Templates in Vue Combo box component

The ComboBox has been provided with several options to customize each list items, group title, header, and footer elements.

To customize vue ComboBox items Using templates, you can check on this video:

### Item template

The content of each list item within the ComboBox can be customized with the help of [itemTemplate](#) property.

In the following sample, each list item is split into two columns to display relevant data's.

### APP.VUE

```

<template>
  <div id="app">
    <div id='container' style="margin:50px auto 0; width:250px;">

```

```

        <br>
        <ejs-dropdownlist id='dropdownlist' placeholder='Select an employee'
sortOrder='Ascending' :itemTemplate='itemTemplate' :dataSource='dataSource'
:query='query' :fields='fields'></ejs-dropdownlist>
    </div>
</div>
</template>
<script>
import Vue from 'vue';
import { DropDownListPlugin } from "@syncfusion/ej2-vue-dropdowns";
Vue.use(DropDownListPlugin);
import { DataManager, Query, ODataV4Adaptor } from "@syncfusion/ej2-data";
var itemVue = Vue.component("itemTemplate", {
    template: `<span><span class='name'>{{data.FirstName}}</span><span class
='city'>{{data.City}}</span></span>`,
    data() {
        return {
            data: {}
        };
    }
});
export default {
    data () {
        return {
            itemTemplate : function(e) {
                return {
                    template: itemVue
                };
            },
            query : new Query().from('Employees').select(['FirstName', 'City',
'EmployeeID']).take(6),
            dataSource : new DataManager({
                url: 'https://services.odata.org/V4/Northwind/Northwind.svc/',
                adaptor: new ODataV4Adaptor,
                crossDomain: true
            }),
            fields: { text: 'FirstName', value: 'EmployeeID' }
        }
    }
}
</script>
<style>
@import "../../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../../node_modules/@syncfusion/ej2-inputs/styles/material.css";
@import "../../node_modules/@syncfusion/ej2-vue-
dropdowns/styles/material.css";
.city {
    right: 15px;
    position: absolute;
}
.combobox {
    width: 30%;
    margin: 0 auto;
}
#app {
    color: #008cff;
    height: 40px;

```



```
position: absolute;
top: 10%;
width: 90%;
}
```

{% previewsample "page.domainurl/code-snippet/combobox/templates/item-cs1" %}

### Group template

The group header title under which appropriate sub-items are categorized can also be customize with the help of [groupTemplate](#) property. This template is common for both inline and floating group header template.

In the following sample, employees are grouped according to their city.

### APP.VUE

```
<template>
  <div id="app">
    <div class='combobox'>
      <ejs-combobox id='combobox' sortOrder="Ascending"
:dataSource='employeeData' :groupTemplate='groupTemplate' :fields='fields'
:query='query' placeholder="Select an employee"></ejs-combobox>
    </div>
  </div>
</template>
<script>
import Vue from 'vue';
import { ComboBoxPlugin } from "@syncfusion/ej2-vue-dropdowns";
Vue.use(ComboBoxPlugin);
import { Query, DataManager, Predicate, ODataV4Adaptor } from
'@syncfusion/ej2-data';
var groupVue = Vue.component("groupTemplate", {
  template: `<strong>{{data.City}}</strong>`,
  data() {
    return {
      data: {}
    };
  }
});
export default {
  data () {
    return {
      employeeData : new DataManager({
        url: 'https://services.odata.org/V4/Northwind/Northwind.svc/',
        adaptor: new ODataV4Adaptor,
        crossDomain: true
      }),
      query : new Query().from('Employees').select(['FirstName', 'City',
'EmployeeID']).take(6).where(new Predicate('City', 'equal',
'London').or('City', 'equal', 'Seattle')),
      fields : { text: 'FirstName', value: 'EmployeeID', groupBy: 'City'
    },
    groupTemplate : function(e) {
      return {
        template: groupVue
      }
    }
  }
};
```

```

        },
    },
}
}
</script>
<style>
@import "../../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../../node_modules/@syncfusion/ej2-inputs/styles/material.css";
@import "../../node_modules/@syncfusion/ej2-vue-
dropdowns/styles/material.css";
#app {
    color: #008cff;
    height: 40px;
    position: absolute;
    top: 10%;
    width: 90%;
}
.combobox {
    width: 30%;
    margin: 0 auto;
}
</style>

```

{% previewsample "page.domainurl/code-snippet/combobox/templates/group-cs1" %}

### Header template

The header element is shown statically at the top of the popup list items within the ComboBox, and any custom element can be placed as a header element using the [headerTemplate](#) property.

In the following sample, the list items and its headers are designed and displayed as two columns similar to multiple columns of the grid.

### APP.VUE

```

<template>
  <div id="app">
    <div class='combobox'>
      <br>
      <ejs-combobox id='combobox' sortOrder="Ascending"
:dataSource='employeeData' :itemTemplate='itemTemplate'
:headerTemplate='headerTemplate' :fields='fields' :query='query'
placeholder="Select an employee"></ejs-combobox>
    </div>
  </div>
</template>
<script>
import Vue from 'vue';
import { ComboBoxPlugin } from "@syncfusion/ej2-vue-dropdowns";
Vue.use(ComboBoxPlugin);
import { Query, DataManager, Predicate, ODataV4Adaptor } from
'@syncfusion/ej2-data';
export default {
  data () {
    return {
      employeeData : new DataManager({

```

```

        url: 'https://services.odata.org/V4/Northwind/Northwind.svc/',
        adaptor: new ODataV4Adaptor,
        crossDomain: true
    )),
    query : new Query().from('Employees').select(['FirstName', 'City',
'EmployeeID']).take(6),
    fields : { text: 'FirstName', value: 'EmployeeID' },
    headerTemplate : "<span class='head'><span
class='name'>Name</span><span class='city'>City</span></span>"
    itemTemplate : "<span class='item' ><span
class='name'>${FirstName}</span><span class='city'>${City}</span></span>"
    }
    }
}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-inputs/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-
dropdowns/styles/material.css";
.head,
.item {
    display: table;
    width: 100%;
    margin: auto;
}
.head {
    height: 40px;
    font-size: 15px;
    font-weight: 600;
    text-indent: 1.067em;
}
.name,
.city {
    display: table-cell;
    vertical-align: middle;
    width: 50%;
}
#app {
    color: #008cff;
    height: 40px;
    position: absolute;
    top: 10%;
    width: 90%;
}
.combobox {
    width: 30%;
    margin: 0 auto;
}
</style>

```

{% previewsample "page.domainurl/code-snippet/combobox/templates/header-cs1" %}

### Footer template

The ComboBox has options to show a footer element at the bottom of the list items in the popup list. Here, you can place any custom element as a footer element using the [footerTemplate](#) property.

In the following sample, footer element displays the total number of list items present in the ComboBox.

### APP.VUE

```
<template>
  <div id="app">
    <div id='container' style="margin:50px auto 0; width:250px;">
      <br>
      <ejs-combobox id='combobox' :dataSource='sportsData'
:footerTemplate='footerTemplate' placeholder="Select a game"></ejs-combobox>
    </div>
  </div>
</template>
<script>
import Vue from 'vue';
import { ComboBoxPlugin } from "@syncfusion/ej2-vue-dropdowns";
Vue.use(ComboBoxPlugin);
var footerVue = Vue.component("footerTemplate", {
  template: `<span class='foot'> Total list item: 4</span>`,
  data() {
    return {
      data: {}
    };
  }
});
export default {
  data () {
    var data = ["BasketBall", "Cricket", "Football", "Golf"];
    return {
      sportsData: data,
      footerTemplate : function(e) {
        return {
          template: footerVue
        }
      }
    }
  }
}
</script>
<style>
@import ".../node_modules/@syncfusion/ej2-base/styles/material.css";
@import ".../node_modules/@syncfusion/ej2-inputs/styles/material.css";
@import ".../node_modules/@syncfusion/ej2-vue-
dropdowns/styles/material.css";
.foot {
  text-indent: 1.2em;
  display: block;
  font-size: 15px;
  line-height: 40px;
  border-top: 1px solid #e0e0e0;
}
</style>
```

{% previewsample "page.domainurl/code-snippet/combobox/templates/footer-cs1" %}

### No records template

The ComboBox is provided with support to custom design the popup list content when no data is found and no matches found on search with the help of [noRecordsTemplate](#) property.

In the following sample, popup list content displays the notification of no data available.

#### APP.VUE

```
<template>
  <div id="app">
    <div id='container' style="margin:50px auto 0; width:250px;">
      <br>
      <ejs-combobox id='combobox' :dataSource='sportsData'
:noRecordsTemplate='noRecordsTemplate' placeholder="Select an item"></ejs-
combobox>
    </div>
  </div>
</template>
<script>
import Vue from 'vue';
import { ComboBoxPlugin } from "@syncfusion/ej2-vue-dropdowns";
Vue.use(ComboBoxPlugin);
var footerVue = Vue.component("noRecordsTemplate", {
  template: `<span class='norecord'> NO DATA AVAILABLE</span>`,
  data() {
    return {
      data: {}
    };
  }
});
export default {
  data () {
    var data = [];
    return {
      sportsData: data,
      noRecordsTemplate : function(e) {
        return {
          template: footerVue
        }
      }
    }
  }
}
</script>
<style>
@import "../../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../../node_modules/@syncfusion/ej2-inputs/styles/material.css";
@import "../../node_modules/@syncfusion/ej2-vue-
dropdowns/styles/material.css";
</style>
```

{% previewsample "page.domainurl/code-snippet/combobox/templates/empty-cs1" %}

### Action failure template

There is also an option to custom design the popup list content when the data fetch request fails at the remote server. This can be achieved using the [actionFailureTemplate](#) property.

In the following sample, when the data fetch request fails, the ComboBox displays the notification.

### APP.VUE

```
<template>
  <div id="app">
    <div id='container' style="margin:50px auto 0; width:250px;">
      <br>
      <ejs-combobox id='combobox' :dataSource='customerData'
      :actionFailureTemplate='actionFailureTemplate' :fields='fields'
      :query='query' placeholder="Select a customer"></ejs-combobox>
    </div>
  </div>
</template>
<script>
import Vue from 'vue';
import { ComboBoxPlugin } from "@syncfusion/ej2-vue-dropdowns";
Vue.use(ComboBoxPlugin);
import { Query, DataManager, Predicate, ODataV4Adaptor } from
'syncfusion/ej2-data';
var footerVue = Vue.component("actionFailureTemplate", {
  template: `<span class='action-failure'> Data fetch get fails</span>`,
  data() {
    return {
      data: {}
    };
  }
});
export default {
  data () {
    return {
      customerData : new DataManager({
        url: 'https://services.odata.org/V4/Northwind/Northwind.svcs/',
        adaptor: new ODataV4Adaptor,
        crossDomain: true
      }),
      query : new Query().from('Customers').select(['ContactName',
'CustomerID']).take(6),
      fields : { text: 'ContactName', value: 'CustomerID' },
      actionFailureTemplate : function(e) {
        return {
          template: footerVue
        }
      }
    }
  }
}
</script>
<style>
@import "../../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../../node_modules/@syncfusion/ej2-inputs/styles/material.css";
@import "../../node_modules/@syncfusion/ej2-vue-
dropdowns/styles/material.css";
</style>
```

{% previewsample "page.domainurl/code-snippet/combobox/templates/action-failure-cs1" %}

## See Also

- [How to achieve filtering](#)
- [How to group the data using header](#)
- [How to show the list items with icon](#)

## Virtualization in ComboBox Component

ComboBox virtualization is a technique used to efficiently render extensive lists of items while minimizing the impact on performance. This method is particularly advantageous when dealing with large datasets because it ensures that only a fixed number of DOM (Document Object Model) elements are created. When scrolling through the list, existing DOM elements are reused to display relevant data instead of generating new elements for each item. This recycling process is managed internally.

During virtual scrolling, the data retrieved from the data source depends on the popup height and the calculation of the list item height. Enabling the [enableVirtualization](#) option in a ComboBox activates this virtualization technique.

When fetching data from the data source, the [actionBegin](#) event is triggered before data retrieval begins. Then, the [actionComplete](#) event is triggered once the data is successfully fetched.

Furthermore, Incremental Search is supported with virtualization in the Combobox component. When a key is typed, the focus is moved to the respective element in the open popup state. In the closed popup state, the popup opens, and focus is moved to the respective element in the popup list based on the typed key. The Incremental Search functionality is well-suited for scenarios involving remote data binding.

When the `enableVirtualization` property is enabled, the `skip` and `take` properties provided by the user within the Query class at the initial state or during the `actionBegin` or `actionComplete` events will not be considered, since it is internally managed and calculated based on certain dimensions with respect to the popup height.

### Binding local data

The Combobox can generate its list items through an array of complex data. For this, the appropriate columns should be mapped to the [fields](#) property. When using virtual scrolling, the list updates based on the scroll offset value, triggering a request to fetch more data from the server. As the data is being fetched, the `actionBegin` event occurs before the data retrieval starts. Once the data retrieval is successful, the `actionComplete` event is triggered, indicating that the data fetch process is complete.

In the following example, `id` column and `text` column from complex data have been mapped to the `value` field and `text` field, respectively.

### APP.VUE

```
<template>
<div id="app">
  <div id="wrapper1">
    <ejs-combobox id='combobox' :dataSource='itemData'
placeholder='e.g Item 1' :fields='fields' :enableVirtualization='true'
:allowFiltering='false' popupHeight="200px"></ejs-combobox>
  </div>
</div>
</template>
</script>
```

```
import { ComboBoxComponent, VirtualScroll } from "@syncfusion/ej2-vue-
dropdowns";
let records = [];
function dataSource() {
    for (let i = 1; i <= 150; i++) {
        let item = {
            id: 'id' + i,
            text: "Item " + i,
        };
        records.push(item);
    }
}
dataSource();
//Component registration
export default {
    name: 'App',
    components: {
        "ejs-combobox": ComboBoxComponent
    },
    data () {
        return {
            itemData: records,
            fields: { value: 'id', text: 'text' },
            allowFiltering: true,
        }
    },
    provide: {
        combobox: [VirtualScroll]
    }
}
</script>
<style>
@import ".../node_modules/@syncfusion/ej2-base/styles/material.css";
@import ".../node_modules/@syncfusion/ej2-inputs/styles/material.css";
@import ".../node_modules/@syncfusion/ej2-vue-
dropdowns/styles/material.css";
@import ".../node_modules/@syncfusion/ej2-
notifications/styles/material.css";
#wrapper1{
    min-width: 250px;
    float: left;
    margin-left: 350px;
}
#wrapper2{
    min-width: 250px;
    float: right;
    margin-right: 100px;
}
</style>
```

{% previewsample "page.domainurl/code-snippet/combobox/virtual-scroll" %}

### Binding remote data

The Combobox supports retrieval of data from remote data services with the help of **DataManager** component. When using remote data, it initially fetches all the data from the server, triggering the



`actionBegin` and `actionComplete` events, and then stores the data locally. During virtual scrolling, additional data is retrieved from the locally stored data, triggering the `actionBegin` and `actionComplete` events at that time as well.

The following sample displays the OrderId from the `Orders` Data Service.

#### APP.VUE

```
<template>
<div id="app">
  <div id="wrapper1">
    <ejs-combobox id='combobox' :dataSource='itemData'
placeholder='e.g Item 1' :fields='fields' :enableVirtualization='true'
:allowFiltering='true' popupHeight="200px"></ejs-combobox>
  </div>
</div>
</template>
<script>
import { ComboBoxComponent, VirtualScroll } from "@syncfusion/ej2-vue-
dropdowns";
import { Query, DataManager, WebApiAdaptor } from '@syncfusion/ej2-data';
var remoteData = new DataManager({
  url: 'https://services.syncfusion.com/js/production/api/orders',
  adaptor: new WebApiAdaptor,
  crossDomain: true
});
//Component registration
export default {
  name: 'App',
  components: {
    "ejs-combobox": ComboBoxComponent
  },
  data () {
    return {
      itemData: remoteData,
      fields: { text: 'OrderID', value: 'OrderID' },
      allowFiltering: true,
    }
  },
  provide: {
    combobox: [VirtualScroll]
  }
}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-inputs/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-
dropdowns/styles/material.css";
@import "../node_modules/@syncfusion/ej2-
notifications/styles/material.css";
#wrapper1{
  min-width: 250px;
  float: left;
  margin-left: 350px;
}
#wrapper2{
```

```
min-width: 250px;
float: right;
margin-right: 100px;
}
</style>
```

{% previewsample "page.domainurl/code-snippet/combobox/virtual-scroll-remote" %}

## Grouping

The Combobox component supports grouping with Virtualization. It allows you to organize elements into groups based on different categories. Each item in the list can be classified using the [groupBy](#) field in the data table. After grouping, virtualization works similarly to local data binding, providing a seamless user experience. When the data source is bound to remote data, an initial request is made to retrieve all data for the purpose of grouping. Subsequently, the grouped data works in the same way as local data binding virtualization, enhancing performance and responsiveness.

The following sample shows the example for Grouping with Virtualization.

### APP.VUE

```
<template>
<div id="app">
  <div id="wrapper1">
    <ejs-combobox id='combobox' :dataSource='itemData'
placeholder='e.g Item 1' :fields='fields' :enableVirtualization='true'
:allowFiltering='true' popupHeight="200px"></ejs-combobox>
  </div>
</div>
</template>
<script>
import { ComboBoxComponent, VirtualScroll } from "@syncfusion/ej2-vue-
dropdowns";
let records = [];
function dataSource() {
  for (var i = 1; i <= 150; i++) {
    var item = {};
    item.id = 'id' + i;
    item.text = "Item ".concat(i);
    var randomGroup = Math.floor(Math.random() * 4) + 1;
    switch (randomGroup) {
      case 1:
        item.group = 'Group A';
        break;
      case 2:
        item.group = 'Group B';
        break;
      case 3:
        item.group = 'Group C';
        break;
      case 4:
        item.group = 'Group D';
        break;
      default:
        break;
    }
    records.push(item);
  }
}
```

```

    }
  }
  dataSource();
  //Component registration
  export default {
    name: 'App',
    components: {
      "ejs-combobox": ComboBoxComponent
    },
    data () {
      return {
        itemData: records,
        fields: { groupBy: 'group', value: 'id', text: 'text' },
        allowFiltering: true,
      }
    },
    provide: {
      combobox: [VirtualScroll]
    }
  }
</script>
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-inputs/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-
dropdowns/styles/material.css";
@import "../node_modules/@syncfusion/ej2-
notifications/styles/material.css";
#wrapper1{
  min-width: 250px;
  float: left;
  margin-left: 350px;
}
#wrapper2{
  min-width: 250px;
  float: right;
  margin-right: 100px;
}
</style>

```

{% previewsample "page.domainurl/code-snippet/combobox/virtual-scroll-group" %}

### Filtering with Virtualization

The ComboBox component supports Filtering with Virtualization. The ComboBox includes a built-in feature that enables data filtering when the [allowFiltering](#) option is enabled. In the context of Virtual Scrolling, the filtering process operates in response to the typed characters. Specifically, the DropDownList sends a request to the server, utilizing the full data source, to achieve filtering. Before initiating the request, an action event is triggered. Upon successful retrieval of data from the server, an action complete event is triggered. The initial data is loaded when the popup is opened. Whether the filter list has a selection or not, the popup closes.

The following sample shows the example for Filtering with Virtualization.

### APP.VUE

```
<template>
```

```

<div id="app">
  <div id="wrapper1">
    <ejs-combobox id='combobox' :dataSource='itemData'
placeholder='e.g Item 1' :fields='fields' :enableVirtualization='true'
:allowFiltering='true' popupHeight="200px"></ejs-combobox>
  </div>
</div>
</template>
<script>
import { ComboBoxComponent, VirtualScroll } from "@syncfusion/ej2-vue-
dropdowns";
let records = [];
function dataSource() {
  for (let i = 1; i <= 150; i++) {
    let item = {
      id: 'id' + i,
      text: "Item " + i,
    };
    records.push(item);
  }
}
dataSource();
//Component registration
export default {
  name: 'App',
  components: {
    "ejs-combobox": ComboBoxComponent
  },
  data () {
    return {
      itemData: records,
      fields: { value: 'id', text: 'text' },
      allowFiltering: true,
    }
  },
  provide: {
    combobox: [VirtualScroll]
  }
}
</script>
<style>
@import "../../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../../node_modules/@syncfusion/ej2-inputs/styles/material.css";
@import "../../node_modules/@syncfusion/ej2-vue-
dropdowns/styles/material.css";
@import "../../node_modules/@syncfusion/ej2-
notifications/styles/material.css";
#wrapper1{
  min-width: 250px;
  float: left;
  margin-left: 350px;
}
#wrapper2{
  min-width: 250px;
  float: right;
  margin-right: 100px;
}

```

```
</style>
```

```
{% previewsample "page.domainurl/code-snippet/combobox/virtual-scroll-filter" %}
```

## Grouping in Vue Combo box component

The ComboBox supports wrapping nested elements into a group based on different categories. The category of each list item can be mapped through the [groupByLink to the Video](#) field in the data table. The group header is displayed both as inline and fixed headers. The fixed group header content is updated dynamically on scrolling the popup list with its category value.

To group the Vue ComboBox items, you can check on this video:

In the following sample, the vegetables are grouped according on its category using `groupBy` field.

### APP.VUE

```
<template>
  <div id="app">
    <div id='container' style="margin:50px auto 0; width:250px;">
      <br>
      <ejs-combobox id='combobox' :dataSource='vegetableData'
:fields='fields' popupHeight="200px" placeholder="select a vegetable"></ejs-
combobox>
    </div>
  </div>
</template>
<script>
import Vue from 'vue';
import { ComboBoxPlugin } from "@syncfusion/ej2-vue-dropdowns";
Vue.use(ComboBoxPlugin);
export default {
  data () {
    return {
      vegetableData: [
        { Vegetable: 'Cabbage', Category: 'Leafy and Salad', Id: 'item1' },
        { Vegetable: 'Spinach', Category: 'Leafy and Salad', Id: 'item2' },
        { Vegetable: 'Wheat grass', Category: 'Leafy and Salad', Id: 'item3' },
        { Vegetable: 'Yarrow', Category: 'Leafy and Salad', Id: 'item4' },
        { Vegetable: 'Pumpkins', Category: 'Leafy and Salad', Id: 'item5' },
        { Vegetable: 'Chickpea', Category: 'Beans', Id: 'item6' },
        { Vegetable: 'Green bean', Category: 'Beans', Id: 'item7' },
        { Vegetable: 'Horse gram', Category: 'Beans', Id: 'item8' },
        { Vegetable: 'Garlic', Category: 'Bulb and Stem', Id: 'item9' },
        { Vegetable: 'Nopal', Category: 'Bulb and Stem', Id: 'item10' },
        { Vegetable: 'Onion', Category: 'Bulb and Stem', Id: 'item11' }
      ],
      fields : { groupBy: 'Category', text: 'Vegetable', value: 'Id' }
    }
  }
}</script>
```

```
<style>
@import
"https://ej2.syncfusion.com/vue/documentation/.../node_modules/@syncfusion/ej2-base/styles/material.css";
@import ".../node_modules/@syncfusion/ej2-inputs/styles/material.css";
@import ".../node_modules/@syncfusion/ej2-vue-
dropdowns/styles/material.css";
</style>
```

{% previewsample "page.domainurl/code-snippet/combobox/grouping-cs1" %}

### Customization

The grouping header is also provided with customization option. This allows custom designing using the [groupTemplate](#) property for both inline and fixed headers.

See Also

- [Group Template support to ComboBox.](#)

### Filtering in Vue Combo box component

The ComboBox has built-in support to filter data items when `allowFiltering` is enabled. The filter operation starts as soon as you start typing characters in the component.

To display filtered items in the popup, filter the required data and return it to the ComboBox via [updateData](#) method by using the [filteringLink to the Video](#) event.

To filter the Vue ComboBox items, you can check on this video:

The following sample illustrates how to query the data source and pass the data to the ComboBox through the `updateData` method in `filtering` event.

### APP.VUE

```
<template>
  <div id="app">
    <div id='container' style="margin:50px auto 0; width:250px;">
      <br>
      <ejs-combobox id='combobox' sortOrder='Ascending'
popupHeight="250px" :allowFiltering='allowFiltering' :filtering='filtering'
:dataSource='searchData' :fields='fields' placeholder="Select a
country"></ejs-combobox>
    </div>
  </div>
</template>
<script>
import Vue from 'vue';
import { ComboBoxPlugin } from "@syncfusion/ej2-vue-dropdowns";
Vue.use(ComboBoxPlugin);
import { Query } from '@syncfusion/ej2-data';
export default {
  data () {
    return {
      searchData : [
        { Index: "s1", Country: "California" }, { Index: "s2", Country:
"Florida" },
```

```

        { Index: "s3", Country: "Alaska" }, { Index: "s4", Country:
"Georgia" }
    ],
    fields : { text: "Country", value: "Index" },
    allowFiltering : true
  }
},
methods: {
  filtering: function(args) {
    var searchData = [
      { Index: "s1", Country: "California" }, { Index: "s2",
Country: "Florida" },
      { Index: "s3", Country: "Alaska" }, { Index: "s4", Country:
"Georgia" }
    ];
    var query = new Query();
    //frame the query based on search string with filter type.
    query = (args.text != "") ? query.where("Country", "startswith",
args.text, true) : query;
    //pass the filter data source, filter query to updateData
method.
    args.updateData(searchData, query);
  }
}
}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-inputs/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-
dropdowns/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/combobox/filtering/default-cs1" %}

### Limit the minimum filter character

When filtering the list items, you can set the limit for character count to raise remote request and fetch filtered data on the ComboBox. This can be done by manual validation within the filter event handler.

In the following example, the remote request does not fetch the search data until the search key contains three characters.

### APP.VUE

```

<template>
  <div id="app">
    <div id='container' style="margin:50px auto 0; width:250px;">
      <br>
      <ejs-combobox id='combobox' sortOrder='Ascending'
popupHeight="250px" :allowFiltering='allowFiltering' :filtering='filtering'
:dataSource='searchData' :fields='fields' placeholder="Select a
customer"></ejs-combobox>
    </div>
  </div>
</template>
<script>

```

```
import Vue from 'vue';
import { ComboBoxPlugin } from "@syncfusion/ej2-vue-dropdowns";
Vue.use(ComboBoxPlugin);
import { Query, DataManager, ODataV4Adaptor } from '@syncfusion/ej2-data';
export default {
  data () {
    return {
      searchData : new DataManager({
        url:
'https://services.odata.org/V4/Northwind/Northwind.svc/Customers',
        adaptor: new ODataV4Adaptor,
        crossDomain: true
      }),
      fields : { text: 'ContactName', value: 'CustomerID' },
      query : new Query().select(['ContactName', 'CustomerID']).take(6),
      allowFiltering : true
    }
  },
  methods: {
    filtering: function(e) {
      var searchData = new DataManager({
        url:
'https://services.odata.org/V4/Northwind/Northwind.svc/Customers',
        adaptor: new ODataV4Adaptor,
        crossDomain: true
      });
      // load overall data when search key empty.
      if (e.text == '') e.updateData(searchData);
      else {
        // restrict the remote request until search key contains 3
characters.
        if (e.text.length < 3) { return; }
        var query = new Query().select(['ContactName',
'CustomerID']);
        query = (e.text !== '') ? query.where('ContactName',
'startswith', e.text, true) : query;
        e.updateData(searchData, query);
      }
    }
  }
}
</script>
<style>
@import "../../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../../node_modules/@syncfusion/ej2-inputs/styles/material.css";
@import "../../node_modules/@syncfusion/ej2-vue-
dropdowns/styles/material.css";
</style>
```

{% previewsample "page.domainurl/code-snippet/combobox/filtering/min-char-cs1" %}

### Change the filter type

While filtering, you can change the filter type to `contains`, `startsWith`, or `endsWith` for string type within the filter event handler.

In the following examples, data filtering is done with `endsWith` type.



## APP.VUE

```
<template>
  <div id="app">
    <div id='container' style="margin:50px auto 0; width:250px;">
      <br>
      <ejs-combobox id='combobox' sortOrder='Ascending'
popupHeight="250px" :allowFiltering='allowFiltering' :filtering='filtering'
:dataSource='searchData' :query='query' :fields='fields' placeholder="Select
a customer"></ejs-combobox>
    </div>
  </div>
</template>
<script>
import Vue from 'vue';
import { ComboBoxPlugin } from "@syncfusion/ej2-vue-dropdowns";
Vue.use(ComboBoxPlugin);
import { Query, DataManager, ODataV4Adaptor } from '@syncfusion/ej2-data';
export default {
  data () {
    return {
      searchData : new DataManager({
        url:
'https://services.odata.org/V4/Northwind/Northwind.svc/Customers',
        adaptor: new ODataV4Adaptor,
        crossDomain: true
      }),
      fields : { text: 'ContactName', value: 'CustomerID' },
      query : new Query().select(['ContactName', 'CustomerID']).take(6),
      allowFiltering : true
    }
  },
  methods: {
    filtering: function(e) {
      var searchData = new DataManager({
        url:
'https://services.odata.org/V4/Northwind/Northwind.svc/Customers',
        adaptor: new ODataV4Adaptor,
        crossDomain: true
      });
      if (e.text == '') e.updateData(searchData);
      else {
        var query = new Query().select(['ContactName',
'CustomerID']);
        query = (e.text !== '') ? query.where('ContactName',
'startswith', e.text, true) : query;
        e.updateData(searchData, query);
      }
    }
  }
}
</script>
<style>
@import "../../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../../node_modules/@syncfusion/ej2-inputs/styles/material.css";
@import "../../node_modules/@syncfusion/ej2-vue-
dropdowns/styles/material.css";

```

```
</style>
```

```
{% previewsample "page.domainurl/code-snippet/combobox/filtering/type-cs1" %}
```

### Case sensitive filtering

Data items can be filtered either with or without case sensitivity using the DataManager. This can be done by passing the fourth optional parameter of the `where` clause.

The following example shows how to perform case-sensitive filter.

#### APP.VUE

```
<template>
  <div id="app">
    <div id='container' style="margin:50px auto 0; width:250px;">
      <br>
      <ejs-combobox id='combobox' sortOrder='Ascending'
popupHeight="250px" :allowFiltering='allowFiltering' :filtering='filtering'
:dataSource='searchData' :query='query' :fields='fields' placeholder="Select
a customer"></ejs-combobox>
    </div>
  </div>
</template>
<script>
import Vue from 'vue';
import { ComboBoxPlugin } from "@syncfusion/ej2-vue-dropdowns";
Vue.use(ComboBoxPlugin);
import { Query, DataManager, ODataV4Adaptor } from '@syncfusion/ej2-data';
export default {
  data () {
    return {
      searchData : new DataManager({
        url:
'https://services.odata.org/V4/Northwind/Northwind.svc/Customers',
        adaptor: new ODataV4Adaptor,
        crossDomain: true
      }),
      fields : { text: 'ContactName', value: 'CustomerID' },
      query : new Query().select(['ContactName', 'CustomerID']).take(6),
      allowFiltering : true
    }
  },
  methods: {
    filtering: function(e) {
      var searchData = new DataManager({
        url:
'https://services.odata.org/V4/Northwind/Northwind.svc/Customers',
        adaptor: new ODataV4Adaptor,
        crossDomain: true
      });
      if (e.text == '') e.updateData(searchData);
      else {
        var query = new Query().select(['ContactName',
'CustomerID']);
        query = (e.text !== '') ? query.where('ContactName',
'startswith', e.text, true) : query;
```

```

        e.updateData(searchData, query);
    }
}
}
}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-inputs/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-
dropdowns/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/combobox/filtering/case-cs1" %}

### Diacritics Filtering

ComboBox supports diacritics filtering which will ignore the [diacritics](#) and makes it easier to filter the results in international characters lists when the [ignoreAccent](#) is enabled.

In the following sample, data with diacritics are bound as dataSource for ComboBox.

#### APP.VUE

```

<template>
  <div id="app">
    <div id='container' style="margin:50px auto 0; width:250px;">
      <br>
      <ejs-combobox id='combobox' :dataSource='diacriticsData'
:ignoreAccent="ignoreAccent" :allowFiltering='allowFiltering'
placeholder="e.g: aero"></ejs-combobox>
    </div>
  </div>
</template>
<script>
import Vue from 'vue';
import { ComboBoxPlugin } from "@syncfusion/ej2-vue-dropdowns";
Vue.use(ComboBoxPlugin);
export default {
  data () {
    return {
      diacriticsData : [
        'Aeróbics',
        'Aeróbics en Agua',
        'Aerografía',
        'Aeromodelaje',
        'Águilas',
        'Ajedrez',
        'Ala Delta',
        'Álbumes de Música',
        'Alusivos',
        'Análisis de Escritura a Mano'
      ],
      ignoreAccent : true,
      allowFiltering : true
    }
  }
}

```

```

}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-inputs/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-
dropdowns/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/combobox/filtering/diacritics-cs1" %}

See Also

- [How to achieve autofill while filtering](#)
- [How to group the data using header](#)

## Localization in Vue Combo box component

The Localization library allows you to localize static text content of the [noRecordsTemplate](#) and [actionFailureTemplate](#) property according to the culture currently assigned to the ComboBox.

| Locale key | en-US (default) |

|-----|-----|

| noRecordsTemplate | No records found |

| actionFailureTemplate | The request failed |

## Loading translations

To load translation object to your application, use load function of the **L10n** class.

In the following sample, French culture is set to the ComboBox and no data is loaded. Hence, the **noRecordsTemplate** property displays its text in French culture initially, and if the sample is run offline, the **actionFailureTemplate** property displays its text appropriately.

## APP.VUE

```

<template>
  <div id="app">
    <div id='container' style="margin:50px auto 0; width:250px;">
      <br>
      <ejs-combobox id='combobox' locale='fr-BE' :fields='fields'
:dataSource='customerData' :query='query' placeholder="Sélectionnez un
client"></ejs-combobox>
    </div>
  </div>
</template>
<script>
import Vue from 'vue';
import { ComboBoxPlugin } from "@syncfusion/ej2-vue-dropdowns";
Vue.use(ComboBoxPlugin);
//import DataManager related classes
import { Query, DataManager, ODataV4Adaptor } from '@syncfusion/ej2-data';
// import L10n class for load function
import { L10n } from '@syncfusion/ej2-base';
L10n.load({

```

```

        'fr-BE': {
            'dropdowns': {
                'noRecordsTemplate': "Aucun enregistrement trouvé",
                'actionFailureTemplate': "Modèle d'échec d'action"
            }
        }
    });
export default {
    data () {
        return {
            customerData: new DataManager({
                url:
                'https://services.odata.org/V4/Northwind/Northwind.svc/Customers',
                adaptor: new ODataV4Adaptor,
                crossDomain: true
            }),
            fields : { text: 'ContactName', value: 'CustomerID' },
            query : new Query().select(['ContactName', 'CustomerID']).take(0),
        }
    }
}
</script>
<style>
@import "../../../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../../../node_modules/@syncfusion/ej2-inputs/styles/material.css";
@import "../../../node_modules/@syncfusion/ej2-vue-
dropdowns/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/combobox/locale-cs1" %}

See Also

- [Accessibility](#)
- [How to bind the data to the combobox](#)

### Style in Vue Combo box component

The following content provides the exact CSS structure that can be used to modify the control's appearance based on the user preference.

#### Customizing the appearance of wrapper element

Use the following CSS to customize the appearance of wrapper element.

`

```

.e-ddl.e-input-group.e-control-wrapper .e-input {
font-size: 20px;
font-family: emoji;
color: #ab3243;
background: #32a5ab;
}

```

,

### Customizing the dropdown icon's color

Use the following CSS to customize the dropdown icon's color.

,

```
.e-ddl.e-input-group .e-input-group-icon,.e-ddl.e-input-group.e-control-wrapper .e-input-group-
icon:hover {
color: #bb233d;
font-size: 13px;
}
```

,

### Customizing the focus color

Use the following CSS to customize the focusing color of input element.

,

```
.e-ddl.e-input-group.e-control-wrapper.e-input-focus::before, .e-ddl.e-input-group.e-control-wrapper.e-
input-focus::after {
background: #c000ff;
}
```

,

### Customizing the outline theme's focus color

Use the following CSS to customize the focusing color of outline theme.

,

```
.e-outline.e-input-group.e-input-focus:hover:not(.e-success):not(.e-warning):not(.e-error):not(.e-
disabled):not(.e-float-icon-left),.e-outline.e-input-group.e-input-focus.e-control-wrapper:hover:not(.e-
success):not(.e-warning):not(.e-error):not(.e-disabled):not(.e-float-icon-left),.e-outline.e-input-group.e-
input-focus:not(.e-success):not(.e-warning):not(.e-error):not(.e-disabled),.e-outline.e-input-group.e-
control-wrapper.e-input-focus:not(.e-success):not(.e-warning):not(.e-error):not(.e-disabled) {
border-color: #b1bd15;
box-shadow: inset 1px 1px #b1bd15, inset -1px 0 #b1bd15, inset 0 -1px #b1bd15;
}
```

,

### Customizing the disabled component's text color

Use the following CSS to customize the text color when the component is disabled.

,

```
.e-input-group.e-control-wrapper .e-input[disabled] {
-webkit-text-fill-color: #0d9133;
}
```

### Customizing the float label element's focusing color

Use the following CSS to customize the focusing color of float label element.

```
.e-float-input.e-input-group:not(.e-float-icon-left) .e-float-line::before,.e-float-input.e-control-
wrapper.e-input-group:not(.e-float-icon-left) .e-float-line::before,.e-float-input.e-input-group:not(.e-
float-icon-left) .e-float-line::after,.e-float-input.e-control-wrapper.e-input-group:not(.e-float-icon-left)
.e-float-line::after {
background-color: #2319b8;
}

.e-ddl.e-input-group.e-control-wrapper.e-float-input.e-input-focus .e-float-text.e-label-top, .e-float-
input.e-control-wrapper:not(.e-error).e-input-focus input ~ label.e-float-text {
color: #2319b8;
}
```

### Customizing the color of the placeholder text

Use the following CSS to customize the text color of placeholder.

```
.e-ddl.e-input-group input.e-input::placeholder {
color: red;
}
```

### Customizing the text selection color

Use the following CSS to customize the selection color of text and background.

```
.e-ddl.e-input-group input.e-input::selection {
color: red;
background: yellow;
}
```

### Customizing the background color of focus, hover, and active item's

Use the following CSS to customize the background color of focus, hover and active item's.

```
.e-dropdownbase .e-list-item.e-item-focus, .e-dropdownbase .e-list-item.e-active, .e-dropdownbase .e-
list-item.e-active.e-hover, .e-dropdownbase .e-list-item.e-hover {
background-color: #1f9c99;
```

```
color: #2319b8;
}
```

### Customizing the appearance of pop-up element

Use the following CSS to customize the appearance of popup element.

```
.e-dropdownbase .e-list-item, .e-dropdownbase .e-list-item.e-item-focus {
background-color: #29c2b8;
color: #207cd9;
font-family: emoji;
min-height: 29px;
}
```

### Adding mandatory asterisk to placeholder and float label

You can add a mandatory asterisk(\*) to placeholder and float label using `<b>.e-input-group.e-control-wrapper.e-float-input .e-float-text::after</b>` class.

#### APP.VUE

```
<template>
  <div id="app">
    <div id='container' style="margin:50px auto 0; width:250px;">
      <br>
      <ejs-combobox id='combobox' :dataSource='sportsData'
placeholder='Select a game' :floatLabelType= "auto"></ejs-combobox>
    </div>
  </div>
</template>
<script>
import Vue from 'vue';
import { ComboBoxPlugin } from "@syncfusion/ej2-vue-dropdowns";
Vue.use(ComboBoxPlugin);
export default {
  data () {
    return {
      sportsData: ['Badminton', 'Cricket', 'Football', 'Golf', 'Tennis']
    }
  }
}
</script>
<style>
@import "../../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../../node_modules/@syncfusion/ej2-inputs/styles/material.css";
@import "../../node_modules/@syncfusion/ej2-vue-
dropdowns/styles/material.css";
.e-input-group.e-control-wrapper.e-float-input .e-float-text::after {
  content: '*';
  color: red;
}
```



```
}
</style>
```

```
{% previewsample "page.domainurl/code-snippet/combobox/getting-started/getting-started-cs2" %}
```

## Accessibility in Vue Combo box component

The ComboBox component has been designed, keeping in mind the WAI-ARIA specifications, and applies the WAI-ARIA roles, states, and properties along with keyboard support. This component is characterized by complete keyboard interaction support and ARIA accessibility support that makes it easy for people who use assistive technologies (AT) or those who completely rely on keyboard navigation.

The ComboBox component followed the accessibility guidelines and standards, including [ADA](#), [Section 508](#), [WCAG 2.2](#) standards, and [WCAG roles](#) that are commonly used to evaluate accessibility.

The accessibility compliance for the ComboBox component is outlined below.

| Accessibility Criteria | Compatibility |

| -- | -- |

| [WCAG 2.2 Support](#) |  |

| [Section 508 Support](#) |  |

| [Screen Reader Support](#) |  |

| [Right-To-Left Support](#) |  |

| [Color Contrast](#) |  |

| [Mobile Device Support](#) |  |

| [Keyboard Navigation Support](#) |  |

| [Accessibility Checker Validation](#) |  |

| [Axe-core Accessibility Validation](#) |  |

<style>

```
.post .post-content img {
display: inline-block;
margin: 0.5em 0;
}
```

</style>

<div> - All features of the component meet the requirement.</div>

<div> - Some features of the component do not meet the requirement.</div>

<div> - The component does not meet the requirement.</div>

### WAI-ARIA attributes

The ComboBox component uses the **combobox** role, and each list item has an **option** role. The following ARIA attributes denote the ComboBox state.

#### | Properties | Functionalities |

| --- | --- |

| aria-haspopup | Indicates whether the ComboBox input element has a popup list or not. |

| aria-expanded | Indicates whether the popup list has expanded or not. |

| aria-selected | Indicates the selected option. |

| aria-readonly | Indicates the readonly state of the ComboBox element. |

| aria-disabled | Indicates whether the ComboBox component is in a disabled state or not. |

| aria-activedescendent | This attribute holds the ID of the active list item to focus its descendant child element. |

| aria-owns | This attribute contains the ID of the popup list to indicate popup as a child element. |

| aria-autocomplete | This attribute contains the 'both' to a list of options shows and the currently selected suggestion also shows inline. |

### Keyboard interaction

You can use the following key shortcuts to access the ComboBox without interruptions.

#### | Keyboard shortcuts | Actions |

| --- | --- |

| Arrow Down | Selects the first item in the ComboBox when no item selected. Otherwise, selects the item next to the currently selected item. |

| Arrow Up | Selects the item previous to the currently selected one. |

| Page Down | Scrolls down to the next page and selects the first item when popup list opens. |

| Page Up | Scrolls up to the previous page and selects the first item when popup list opens. |

| Enter | Selects the focused item and popup list closes when it is in open state. |

| Tab | Focuses on the next TabIndex element on the page when the popup is closed. Otherwise, closes the popup list and remains the focus of the component. |

| Shift + tab | Focuses on the previous TabIndex element on the page when the popup is closed. Otherwise, closes the popup list and remains the focus of the component. |

| Alt + Down | Open the popup list |

| Alt + Up | Close the popup list|

| Esc(Escape) | Closes the popup list when it is in an open state and the currently selected item remains the same. |

| Home | Cursor moves to before of first character in input |

| End | Cursor moves to next of last character in input |

In the below sample, alt+t keys are used to focus the ComboBox component.

#### APP.VUE

```
<template>
  <div id="app">
    <div id='container' style="margin:50px auto 0; width:250px;">
      <br>
      <ejs-combobox id='combobox' ref='comboboxInstance'
      :dataSource='gameList' :fields='fields' popupHeight='200px'
      placeholder='Select a game'></ejs-combobox>
    </div>
  </div>
</template>
<script>
import Vue from 'vue';
import { ComboBoxPlugin } from "@syncfusion/ej2-vue-dropdowns";
Vue.use(ComboBoxPlugin);
export default {
  mounted () {
    let comboboxObj = this.$refs.comboboxInstance;
    document.onkeyup = function (e) {
      if (e.altKey && e.keyCode === 84) {
        // press alt+t to focus the control.
        comboboxObj.$el.focus();
      }
    }
  },
  data () {
    return {
      gameList: [
        { Id: 'Game2', Game: 'Badminton' },
        { Id: 'Game3', Game: 'Basketball' },
        { Id: 'Game4', Game: 'Cricket' },
        { Id: 'Game5', Game: 'Football' },
        { Id: 'Game6', Game: 'Golf' },
        { Id: 'Game7', Game: 'Hockey' },
        { Id: 'Game8', Game: 'Rugby' },
        { Id: 'Game9', Game: 'Snooker' },
        { Id: 'Game10', Game: 'Tennis' }
      ],
      fields : { text: 'Game', value: 'Id' }
    }
  }
}
</script>
<style>
```

```
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-inputs/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-dropdowns/styles/material.css";
</style>
```

{% previewsample "page.domainurl/code-snippet/combobox/accessibility-cs1" %}

### Ensuring accessibility

The ComboBox component's accessibility levels are ensured through an [accessibility-checker](#) and [axe-core](#) software tools during automated testing.

The accessibility compliance of the ComboBox component is shown in the following sample. Open the [sample](#) in a new window to evaluate the accessibility of the ComboBox component with accessibility tools.

{% previewsample "https://ej2.syncfusion.com/accessibility/combo-box.html" %}

See also

- [Accessibility in Syncfusion Vue components](#)

### Two way binding in Vue Combo box component

Two-way binding can be achieved by using the **v-model** directive in Vue. In the following sample, when you change the value in one ComboBox component, v-model will automatically update the value in the other ComboBox.

The following example demonstrates how to set the **two-way-binding** in the ComboBox.

#### APP.VUE

```
<template>
<div id="app">
  <div id="wrapper1">
    <ejs-combobox id='first':dataSource='sportsData' :fields='fields'
:placeholder="waterMark" v-model="value" ></ejs-combobox>
  </div>
  <div id="wrapper2">
    <ejs-combobox id='second' :dataSource='sportsData'
:fields='fields' :placeholder="waterMark" v-model="value" ></ejs-combobox>
  </div>
</div>
</template>
<script>
import Vue from 'vue';
import { ComboBoxPlugin } from '@syncfusion/ej2-vue-dropdowns';
Vue.use(ComboBoxPlugin);
export default {
  name: 'app',
  data () {
    return {
      waterMark : 'Select a game',
      value: null,
      sportsData: [
        { Id: 'Game1', Game: 'Badminton' },
        { Id: 'Game2', Game: 'Basketball' },
      ]
    }
  }
}
```

```

    { Id: 'Game3', Game: 'Cricket' },
    { Id: 'Game4', Game: 'Football' },
    { Id: 'Game5', Game: 'Golf' },
    { Id: 'Game6', Game: 'Hockey' },
    { Id: 'Game7', Game: 'Rugby' },
    { Id: 'Game8', Game: 'Snooker' }
  ],
  fields: { value: 'Game' }
}
}
</script>
<style>
@import ".../node_modules/@syncfusion/ej2-base/styles/material.css";
@import ".../node_modules/@syncfusion/ej2-inputs/styles/material.css";
@import ".../node_modules/@syncfusion/ej2-vue-
dropdowns/styles/material.css";
#wrapper1{
  min-width: 250px;
  float: left;
  margin-left: 100px;
}
#wrapper2{
  min-width: 250px;
  float: right;
  margin-right: 100px;
}
</style>

```

{% previewsample "page.domainurl/code-snippet/combobox/two-way-cs1" %}

## How To

### Autofill in Vue Combo box component

The ComboBox supports the **autofill** behaviour with the help of [autofill](#) property. Whenever you change the input value, the ComboBox will autocomplete your data by matching the typed character. Suppose, if no matches found then, ComboBox doesn't suggest any item.

The following examples, showcase that how to work autofill with ComboBox.

### APP.VUE

```

<template>
  <div id="app">
    <div id='container' style="margin:50px auto 0; width:250px;">
      <br>
      <ejs-combobox id='combobox' :dataSource='sportsData'
      :autofill='autofill' placeholder='Select a game'></ejs-combobox>
    </div>
  </div>
</template>
<script>
import Vue from 'vue';
import { ComboBoxPlugin } from "@syncfusion/ej2-vue-dropdowns";
Vue.use(ComboBoxPlugin);
export default {
  data () {

```

```

    return {
      sportsData: ['Badminton', 'Cricket', 'Football', 'Golf', 'Tennis'],
      autofill : true,
    }
  }
}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-inputs/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-
dropdowns/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/combobox/autofill-cs1" %}

### Cascading in Vue Combo box component

The cascading ComboBox is a series of ComboBox, where the value of one ComboBox depends upon another's value. This can be configured by using the [change](#) event of the parent ComboBox. Within that change event handler, data has to be loaded to the child ComboBox based on the selected value of the parent ComboBox.

The following example, shows the cascade behavior of country, state, and city ComboBox. Here, the [dataBind](#) method is used to reflect the property changes immediately to the ComboBox.

### APP.VUE

```

<template>
  <div id="app">
    <div id='container' style="margin:50px auto 0; width:250px;">
      <br>
      <ejs-combobox id='countries' :dataSource='countryData'
      :fields='countryfields' :change='onCountryChange' placeholder='Select a
      country'></ejs-combobox>
      <div class="padding-top">
        <ejs-combobox id='states' :dataSource='stateData'
        :enabled='stateenabled' :fields='statefields' :change='onStateChange'
        placeholder='Select a state'></ejs-combobox>
      </div>
      <div class="padding-top">
        <ejs-combobox id='cities' :dataSource='cityData'
        :enabled='cityenabled' :fields='cityfields' placeholder='Select a
        city'></ejs-combobox>
      </div>
    </div>
  </div>
</template>
<script>
import Vue from 'vue';
import { ComboBoxPlugin } from "@syncfusion/ej2-vue-dropdowns";
Vue.use(ComboBoxPlugin);
import { Query } from '@syncfusion/ej2-data';
export default {
  data () {
    return {
      countryData: [

```

```

        { CountryName: 'Australia', CountryId: '2' },
        { CountryName: 'United States', CountryId: '1' }
    ],
    stateData: [
        { StateName: 'New York', CountryId: '1', StateId: '101' },
        { StateName: 'Virginia ', CountryId: '1', StateId: '102' },
        { StateName: 'Tasmania ', CountryId: '2', StateId: '105' }
    ],
    cityData : [
        { CityName: 'Albany', StateId: '101', CityId: 201 },
        { CityName: 'Beacon ', StateId: '101', CityId: 202 },
        { CityName: 'Emporia', StateId: '102', CityId: 206 },
        { CityName: 'Hampton ', StateId: '102', CityId: 205 },
        { CityName: 'Hobart', StateId: '105', CityId: 213 },
        { CityName: 'Launceston ', StateId: '105', CityId: 214 }
    ]
    countryfields : { value: 'CountryId', text: 'CountryName' }
    statefields : { value: 'StateId', text: 'StateName' },
    cityfields : { text: 'CityName', value: 'CityId' },
    stateenabled : false,
    cityenabled : false
    }
},
methods: {
    onCountryChange: function(e) {
        var countryObj =
document.getElementById('countries').ej2_instances[0];
        var stateObj = document.getElementById('states').ej2_instances[0];
        var cityObj = document.getElementById('cities').ej2_instances[0];
        //Query the data source based on country ComboBox selected value
        stateObj.query = new Query().where('CountryId', 'equal',
countryObj.value);
        // enable the state ComboBox
        stateObj.enabled = true;
        //clear the existing selection.
        stateObj.text = null;
        // bind the property changes to state ComboBox
        stateObj.dataBind();
        //clear the existing selection in city ComboBox
        cityObj.text = null;
        //disabe the city ComboBox
        cityObj.enabled = false;
        //bind the property cahnges to City ComboBox
        cityObj.dataBind();
    },
    onStateChange: function(e) {
        var stateObj = document.getElementById('states').ej2_instances[0];
        var cityObj = document.getElementById('cities').ej2_instances[0];
        cityObj.query = new Query().where('StateId', 'equal',
stateObj.value);
        // enable the city ComboBox
        cityObj.enabled = true;
        //clear the existing selection
        cityObj.text = null;
        // bind the property change to city ComboBox
        cityObj.dataBind();
    }
}

```

```

    }
  }
</script>
<style>
@import ".../node_modules/@syncfusion/ej2-base/styles/material.css";
@import ".../node_modules/@syncfusion/ej2-inputs/styles/material.css";
@import ".../node_modules/@syncfusion/ej2-vue-
dropdowns/styles/material.css";
  #container .padding-top {
    padding-top: 35px;
  }
</style>

```

{% previewsample "page.domainurl/code-snippet/combobox/cascade-cs1" %}

### Icons support in Vue Combo box component

You can render **icons** to the list items by mapping the the [iconCss](#) &#160;fields. This **iconCss** field create a span in the list item with mapped class name to allow styling as per your need.

In the following sample, icon classes are mapped with **iconCss** field.

#### APP.VUE

```

<template>
  <div id="app">
    <div id='container' style="margin:50px auto 0; width:250px;">
      <br>
      <ejs-combobox id='combobox' :dataSource='sortFormatData'
:fields='fields' placeholder='Select a format'></ejs-combobox>
    </div>
  </div>
</template>
<script>
import Vue from 'vue';
import { ComboBoxPlugin } from "@syncfusion/ej2-vue-dropdowns";
Vue.use(ComboBoxPlugin);
export default {
  data () {
    return {
      sortFormatData: [
        { Class: 'sort', Type: 'Sort A to Z', Id: '1' },
        { Class: 'filter', Type: 'Filter', Id: '2' },
        { Class: 'clear', Type: 'Clear', Id: '3' }
      ],
      fields : { text: 'Type', iconCss: 'Class', value: 'Id' }
    }
  }
}
</script>
<style>
@import ".../node_modules/@syncfusion/ej2-base/styles/material.css";
@import ".../node_modules/@syncfusion/ej2-inputs/styles/material.css";
@import ".../node_modules/@syncfusion/ej2-vue-
dropdowns/styles/material.css";
.e-list-icon{
  line-height: 1.3;

```



```
padding-right: 10px;
text-indent: 5px;
}
.sort:before {
  content: '\e890';
  font-family: 'e-icons';
  font-size: 15px;
}
.filter:before {
  content: '\e7ee';
  font-family: 'e-icons';
  font-size: 15px;
  opacity: 0.78;
}
.clear:before {
  content: '\e7fc';
  font-family: 'e-icons';
  font-size: 15px;
}
</style>
```

{% previewsample "page.domainurl/code-snippet/combobox/icons-cs1" %}

## Context Menu

### Getting Started with the Vue Context menu Component in Vue 2

This article provides a step-by-step guide for setting up a Vue 2 project using [Vue-CLI](#) and integrating the Syncfusion Vue Context menu component using the [Composition API](#) / [Options API](#).

#### Prerequisites

[System requirements for Syncfusion Vue UI components](#)

#### Dependencies

The following list of dependencies are required to use the ContextMenu component in your application.

```
`js
|-- @syncfusion/ej2-vue-navigations
|-- @syncfusion/ej2-vue-base
|-- @syncfusion/ej2-navigations
|-- @syncfusion/ej2-base
|-- @syncfusion/ej2-data
|-- @syncfusion/ej2-lists
|-- @syncfusion/ej2-inputs
|-- @syncfusion/ej2-popups
|-- @syncfusion/ej2-buttons
\`
```

### Setting up the Vue 2 project

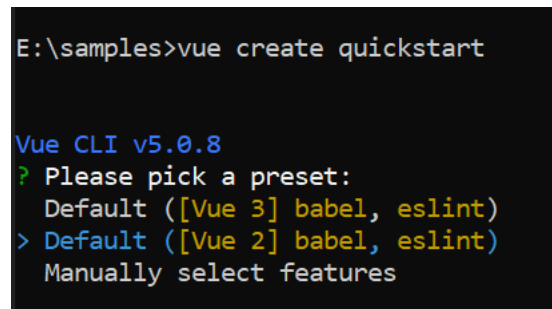
To generate a Vue 2 project using Vue-CLI, use the [vue create](#) command. Follow these steps to install Vue CLI and create a new project:

```
`bash
npm install -g @vue/cli
vue create quickstart
cd quickstart
npm run serve
`
```

or

```
`bash
yarn global add @vue/cli
vue create quickstart
cd quickstart
yarn run serve
`
```

When creating a new project, choose the option `Default ([Vue 2] babel, eslint)` from the menu.



```
E:\samples>vue create quickstart

Vue CLI v5.0.8
? Please pick a preset:
  Default ([Vue 3] babel, eslint)
> Default ([Vue 2] babel, eslint)
  Manually select features
```

Once the `quickstart` project is set up with default settings, proceed to add Syncfusion components to the project.

### Add Syncfusion Vue packages

Syncfusion packages are available at [npmjs.com](https://www.npmjs.com). To use Vue components, install the required npm package.

This article uses the [Vue Context menu component](#) as an example. Install the `@syncfusion/ej2-vue-navigations` package by running the following command:

```
`bash
npm install @syncfusion/ej2-vue-navigations --save
`

or
```

```
`bash
```

```
yarn add @syncfusion/ej2-vue-navigations
```

```
,
```

### Import Syncfusion CSS styles

You can import themes for the Syncfusion Vue component in various ways, such as using CSS or SASS styles from npm packages, CDN, [CRG](#) and [Theme Studio](#). Refer to [themes topic](#) to know more about built-in themes and different ways to refer to themes in a Vue project.

In this article, the **Material** theme is applied using CSS styles, which are available in installed packages. The necessary **Material** CSS styles for the Context menu component and its dependents were imported into the `<style>` section of **src/App.vue** file.

#### ~/SRC/APP.VUE

```
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-buttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-inputs/styles/material.css";
@import "../node_modules/@syncfusion/ej2-popups/styles/material.css";
@import "../node_modules/@syncfusion/ej2-navigations/styles/material.css";
</style>
```

### Add Syncfusion Vue component

Follow the below steps to add the Vue Context menu component using **Composition API** or **Options API**:

1\ First, import and register the Context menu component in the **script** section of the **src/App.vue** file. If you are using the **Composition API**, you should add the **setup** attribute to the **script** tag to indicate that Vue will be using the **Composition API**.

#### COMPOSITION API (~/SRC/APP.VUE)

```
<script setup>
import { ContextMenuComponent as EjsContextmenu } from "@syncfusion/ej2-vue-navigations";
</script>
```

#### OPTIONS API (~/SRC/APP.VUE)

```
<script>
import { ContextMenuComponent } from "@syncfusion/ej2-vue-navigations";
export default {
  components: {
    'ejs-contextmenu': ContextMenuComponent
  }
}
</script>
```

2\ In the **template** section, define the Context menu component with the [items](#) property.

#### ~/SRC/APP.VUE

```
<template>
<div>
<div id="target">Right click / Touch hold to open the ContextMenu</div>
<ejs-contextmenu target='#target' :items='menuItems'></ejs-contextmenu>
</div>
</template>
```

3\ Declare the value for the `itemd` property in the `script` section.

#### **COMPOSITION API (~SRC/APP.VUE)**

```
<script setup>
const menuItems = [
{
text: 'Cut'
},
{
text: 'Copy'
},
{
text: 'Paste'
}];
</script>
```

#### **OPTIONS API (~SRC/APP.VUE)**

```
<script>
data () {
return {
menuItems:[
{
text: 'Cut'
},
{
text: 'Copy'
},
{
text: 'Paste'
}
}];
};
}
</script>
```

Here is the summarized code for the above steps in the `src/App.vue` file:

#### **COMPOSITION API (~SRC/APP.VUE)**

```
<template>
<div>
<div id="target">Right click / Touch hold to open the ContextMenu</div>
<ejs-contextmenu target='#target' :items='menuItems'></ejs-contextmenu>
</div>
</template>
<script setup>
import { ContextMenuComponent as EjsContextmenu } from "@syncfusion/ej2-vue-
navigations";
```

```

import { enableRipple } from '@syncfusion/ej2-base';
enableRipple(true);
const menuItems = [
    {
        text: 'Cut'
    },
    {
        text: 'Copy'
    },
    {
        text: 'Paste'
    }
];
</script>
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-buttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-inputs/styles/material.css";
@import "../node_modules/@syncfusion/ej2-popups/styles/material.css";
@import "../node_modules/@syncfusion/ej2-navigations/styles/material.css";
#target {
    border: 1px dashed;
    height: 150px;
    padding: 10px;
    position: relative;
    text-align: justify;
    color: gray;
    user-select: none;
}
</style>

```

### OPTIONS API (~SRC/APP.VUE)

```

<template>
<div>
<div id="target">Right click / Touch hold to open the ContextMenu</div>
<ejs-contextmenu target='#target' :items='menuItems'></ejs-contextmenu>
</div>
</template>
<script>
import { ContextMenuComponent } from "@syncfusion/ej2-vue-navigations";
import { enableRipple } from '@syncfusion/ej2-base';
enableRipple(true);
export default {
    components: {
        'ejs-contextmenu': ContextMenuComponent
    },
    data () {
        return {
            menuItems:[
                {
                    text: 'Cut'
                },
                {
                    text: 'Copy'
                },
                {

```

```

        text: 'Paste'
      }
    ]
  };
}
}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-buttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-inputs/styles/material.css";
@import "../node_modules/@syncfusion/ej2-popups/styles/material.css";
@import "../node_modules/@syncfusion/ej2-navigations/styles/material.css";
#target {
  border: 1px dashed;
  height: 150px;
  padding: 10px;
  position: relative;
  text-align: justify;
  color: gray;
  user-select: none;
}
</style>

```

### Run the project

To run the project, use the following command:

```
`bash
```

```
npm run serve
```

```
,
```

```
or
```

```
`bash
```

```
yarn run serve
```

```
,
```

```
{% previewsample "page.domainurl/code-snippet/context-menu/default-cs2" %}
```

### Rendering items with Separator

The Separators are the horizontal lines that are used to separate the menu items. You cannot select the separators. You can enable separators to group the menu items using the [separator](#) property. Cut, Copy, and Paste menu items are grouped using the [separator](#) property in the following sample.

### COMPOSITION API (~SRC/APP.VUE)

```

<template>
<div>
<div id="target">Right click / Touch hold to open the ContextMenu</div>
<ejs-contextmenu target='#target' :items='menuItems'></ejs-contextmenu>
</div>
</template>
<script setup>
import { ContextMenuComponent as EjsContextmenu } from "@syncfusion/ej2-vue-
navigations";

```

```

import { enableRipple } from '@syncfusion/ej2-base';
enableRipple(true);
const menuItems = [
    {
        text: 'Cut'
    },
    {
        text: 'Copy'
    },
    {
        text: 'Paste'
    },
    {
        separator: true
    },
    {
        text: 'Font'
    },
    {
        text: 'Paragraph'
    }
];
</script>
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-buttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-inputs/styles/material.css";
@import "../node_modules/@syncfusion/ej2-popups/styles/material.css";
@import "../node_modules/@syncfusion/ej2-navigations/styles/material.css";
#target {
    border: 1px dashed;
    height: 150px;
    padding: 10px;
    position: relative;
    text-align: justify;
    color: gray;
    user-select: none;
}
</style>

```

### OPTIONS API (~SRC/APP.VUE)

```

<template>
<div>
<div id="target">Right click / Touch hold to open the ContextMenu</div>
<ejs-contextmenu target='#target' :items='menuItems'></ejs-contextmenu>
</div>
</template>
<script>
import { ContextMenuComponent } from "@syncfusion/ej2-vue-navigations";
import { enableRipple } from '@syncfusion/ej2-base';
enableRipple(true);
export default {
    components: {
        'ejs-contextmenu': ContextMenuComponent
    },
    data () {

```

```
    return {
      menuItems: [
        {
          text: 'Cut'
        },
        {
          text: 'Copy'
        },
        {
          text: 'Paste'
        },
        {
          separator: true
        },
        {
          text: 'Font'
        },
        {
          text: 'Paragraph'
        }
      ]
    };
  }
}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-buttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-inputs/styles/material.css";
@import "../node_modules/@syncfusion/ej2-popups/styles/material.css";
@import "../node_modules/@syncfusion/ej2-navigations/styles/material.css";
#target {
  border: 1px dashed;
  height: 150px;
  padding: 10px;
  position: relative;
  text-align: justify;
  color: gray;
  user-select: none;
}
</style>
```

{% previewsample "page.domainurl/code-snippet/context-menu/default-cs3" %}

The [separator](#) property **should not** be given along with the other fields in the [MenuItem](#).

See Also

- [ContextMenu with icons](#)
- [Multi-level nesting](#)

### Getting Started with the Vue ContextMenu Component in Vue 3

This article provides a step-by-step guide for setting up a [Vite](#) project with a JavaScript environment and integrating the Syncfusion Vue ContextMenu component using the [Composition API](#) / [Options API](#).



The **Composition API** is a new feature introduced in Vue.js 3 that provides an alternative way to organize and reuse component logic. It allows developers to write components as functions that use smaller, reusable functions called composition functions to manage their properties and behavior.

The **Options API** is the traditional way of writing Vue.js components, where the component logic is organized into a series of options that define the component's properties and behavior. These options include data, methods, computed properties, watchers, lifecycle hooks, and more.

#### Prerequisites

##### [System requirements for Syncfusion Vue UI components](#)

#### Set up the Vite project

A recommended approach for beginning with Vue is to scaffold a project using [Vite](#). To create a new Vite project, use one of the commands that are specific to either NPM or Yarn.

```
`bash
```

```
npm create vite@latest
```

```
,
```

or

```
`bash
```

```
yarn create vite
```

```
,
```

Using one of the above commands will lead you to set up additional configurations for the project as below:

1. Define the project name: We can specify the name of the project directly. Let's specify the name of the project as **my-project** for this article.

```
`bash
```

```
? Project name: » my-project
```

```
,
```

2. Select **Vue** as the framework. It will create a Vue 3 project.

```
`bash
```

```
? Select a framework: » - Use arrow-keys. Return to submit.
```

```
Vanilla
```

```
Vue
```

```
React
```

```
Preact
```

```
Lit
```

```
Svelte
```

```
Others
```

```
,
```

3. Choose `JavaScript` as the framework variant to build this Vite project using JavaScript and Vue.

```
`bash
```

? Select a variant: » - Use arrow-keys. Return to submit.

JavaScript

TypeScript

Customize with create-vue ↗

Nuxt ↗

,

4. Upon completing the aforementioned steps to create the `my-project`, run the following command to install its dependencies:

```
`bash
```

```
cd my-project
```

```
npm install
```

,

or

```
`bash
```

```
cd my-project
```

```
yarn install
```

,

Now that `my-project` is ready to run with default settings, let's add Syncfusion components to the project.

#### Add Syncfusion Vue packages

Syncfusion Vue component packages are available at [npmjs.com](https://www.npmjs.com). To use Syncfusion Vue components in the project, install the corresponding npm package.

This article uses the [Vue ContextMenu component](#) as an example. To use the Vue ContextMenu component in the project, the `@syncfusion/ej2-vue-navigations` package needs to be installed using the following command:

```
`bash
```

```
npm install @syncfusion/ej2-vue-navigations --save
```

,

or

```
`bash
```

```
yarn add @syncfusion/ej2-vue-navigations
```

,

### Import Syncfusion CSS styles

You can import themes for the Syncfusion Vue component in various ways, such as using CSS or SASS styles from npm packages, CDN, [CRG](#) and [Theme Studio](#). Refer to [themes topic](#) to know more about built-in themes and different ways to refer to themes in a Vue project.

In this article, **Material** theme is applied using CSS styles, which are available in installed packages. The necessary **Material** CSS styles for the ContextMenu component and its dependents were imported into the `<style>` section of **src/App.vue** file.

#### ~/SRC/APP.VUE

```
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-buttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-inputs/styles/material.css";
@import "../node_modules/@syncfusion/ej2-popups/styles/material.css";
@import "../node_modules/@syncfusion/ej2-navigations/styles/material.css";
</style>
```

The order of importing CSS styles should be in line with its dependency graph.

### Add Syncfusion Vue component

Follow the below steps to add the Vue ContextMenu component using **Composition API** or **Options API**:

1.First, import and register the ContextMenu component and its child directives in the **script** section of the **src/App.vue** file. If you are using the **Composition API**, you should add the **setup** attribute to the **script** tag to indicate that Vue will be using the **Composition API**.

#### COMPOSITION API (~/SRC/APP.VUE)

```
<script setup>
import { ContextMenuComponent as EjsContextmenu } from "@syncfusion/ej2-vue-navigations";
</script>
```

#### OPTIONS API (~/SRC/APP.VUE)

```
<script>
import { ContextMenuComponent } from "@syncfusion/ej2-vue-navigations";
//Component registration
export default {
  name: "App",
  components: {
    "ejs-contextmenu": ContextMenuComponent
  }
}
</script>
```

2.In the **template** section, define the ContextMenu component with the [items](#) property.

#### ~/SRC/APP.VUE

```
<template>
```

```
<div>
<div id="target">Right click / Touch hold to open the ContextMenu</div>
<ejs-contextmenu target='#target' :items='menuItems'></ejs-contextmenu>
</div>
</template>
```

3. Declare the values for the `items` property in the `script` section.

#### **COMPOSITION API (~SRC/APP.VUE)**

```
<script setup>
const menuItems = [
  {
    text: 'Cut'
  },
  {
    text: 'Copy'
  },
  {
    text: 'Paste'
  },
  {
    separator: true
  },
  {
    text: 'Font'
  },
  {
    text: 'Paragraph'
  }
];
</script>
```

#### **OPTIONS API (~SRC/APP.VUE)**

```
<script>
data() {
  return {
    menuItems:[
      {
        text: 'Cut'
      },
      {
        text: 'Copy'
      },
      {
        text: 'Paste'
      },
      {
        separator: true
      },
      {
        text: 'Font'
      },
      {
        text: 'Paragraph'
      }
    ]
  }
}
</script>
```

```
  }]  
  };  
}  
</script>
```

Here is the summarized code for the above steps in the **src/App.vue** file:

#### **COMPOSITION API (~SRC/APP.VUE)**

```
<template>  
<div>  
<div id="target">Right click / Touch hold to open the ContextMenu</div>  
<ejs-contextmenu target='#target' :items='menuItems'></ejs-contextmenu>  
</div>  
</template>  
<script setup>  
import { ContextMenuComponent as EjsContextmenu } from "@syncfusion/ej2-vue-  
navigations";  
const menuItems = [  
  {  
    text: 'Cut'  
  },  
  {  
    text: 'Copy'  
  },  
  {  
    text: 'Paste'  
  },  
  {  
    separator: true  
  },  
  {  
    text: 'Font'  
  },  
  {  
    text: 'Paragraph'  
  }  
];  
</script>  
<style>  
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";  
@import "../node_modules/@syncfusion/ej2-buttons/styles/material.css";  
@import "../node_modules/@syncfusion/ej2-inputs/styles/material.css";  
@import "../node_modules/@syncfusion/ej2-popups/styles/material.css";  
@import "../node_modules/@syncfusion/ej2-navigations/styles/material.css";  
#target {  
  border: 1px dashed;  
  height: 150px;  
  padding: 10px;  
  position: relative;  
  text-align: justify;  
  color: gray;  
  user-select: none;  
}  
</style>
```

**OPTIONS API (~SRC/APP.VUE)**

```
<template>
<div>
<div id="target">Right click / Touch hold to open the ContextMenu</div>
<ejs-contextmenu target='#target' :items='menuItems'></ejs-contextmenu>
</div>
</template>
<script>
import { ContextMenuComponent } from "@syncfusion/ej2-vue-navigations";
// Component registration
export default {
name: "App",
// Declaring component and its directives
components: {
"ejs-contextmenu": ContextMenuComponent
},
// Bound properties declarations
data() {
return {
menuItems:[
{
text: 'Cut'
},
{
text: 'Copy'
},
{
text: 'Paste'
},
{
separator: true
},
{
text: 'Font'
},
{
text: 'Paragraph'
}],
};
};
</script>
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-buttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-inputs/styles/material.css";
@import "../node_modules/@syncfusion/ej2-popups/styles/material.css";
@import "../node_modules/@syncfusion/ej2-navigations/styles/material.css";
#target {
border: 1px dashed;
height: 150px;
padding: 10px;
position: relative;
text-align: justify;
color: gray;
user-select: none;
}
```

```
}  
</style>
```

### Run the project

To run the project, use the following command:

```
`bash
```

```
npm run dev
```

```
,
```

or

```
`bash
```

```
yarn run dev
```

```
,
```

The output will appear as follows:



For migrating from Vue 2 to Vue 3, refer to the [migration](#) documentation.

See also

- [Getting Started with Vue UI Components using Composition API and TypeScript](#)
- [Getting Started with Vue UI Components using Options API and TypeScript](#)

### Icons and navigation in Vue Context menu component

#### Icons

The ContextMenu item have an icon/image in it to provide visual representation of the action. To place the icon on a menu item, set the [iconCss](#) property to e-icons with the required icon CSS. By default, the icon is positioned to the left side of the menu item. In the following sample, the icons for Cut, Copy and Paste menu items are

added using the `iconCss` property.

**APP.VUE**

```

<template>
<div>
<div id="target">Right click / Touch hold to open the ContextMenu</div>
<ejs-contextmenu target='#target' :items='menuItems'></ejs-contextmenu>
</div>
</template>
<script>
import Vue from 'vue';
import { ContextMenuPlugin } from "@syncfusion/ej2-vue-navigations";
import { enableRipple } from '@syncfusion/ej2-base';
enableRipple(true);
Vue.use(ContextMenuPlugin);
export default {
  data () {
    return {
      menuItems:[
        {
          text: 'Cut',
          iconCss: 'e-cm-icons e-cut'
        },
        {
          text: 'Copy',
          iconCss: 'e-icons e-copy'
        },
        {
          text: 'Paste',
          iconCss: 'e-cm-icons e-paste',
        }
      ]
    };
  }
}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-buttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-inputs/styles/material.css";
@import "../node_modules/@syncfusion/ej2-popups/styles/material.css";
@import "../node_modules/@syncfusion/ej2-navigations/styles/material.css";
#target {
  border: 1px dashed;
  height: 150px;
  padding: 10px;
  position: relative;
  text-align: justify;
  color: gray;
  user-select: none;
}
@font-face {
  font-family: 'ddb-icons';
  src:
    url(data:application/x-font-ttf;charset=utf-
8;base64,AAEAAAAKAIAAAwAgTlMvMj0gSRkAAAEoAAAAVmNtYXDNdE+dkAAABlAAAADxnbHlmlh3
3NQAAAdwAAAjMAgVhZBKOK9sAAADQAAAAANmhoZWEHeANwAAAArAAAACRobXR4E6AAAAAAAYAAAAA
UbG9jYQGOAegAAAHQAAADG1heHABEwBlAAABCAAAACBuYW11lLBM9QAABCgAAAI9cG9zdMJntbU
AAAZoAAAAUAABAAADUv9qAFoEAAAAAADygABAAAAAAAAAAAAAAAAABQABAAAAAQAAojXaQl8

```



```

PPPUACwPoAAAAANfSc4gAAAAA19JziAAA//oDyGPsAAAAACAACAAAAAAAAAAAAEAAAFkABAAAAAA
AAgAAAAoACgAAAP8AAAAAAAAAAQPtAZAABQAAAnoCvAAAAIwCegK8AAAB4AAxAQIAAAIABQMAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAUGZFZABA5wDnAwNS/2oAWgPsAJYAAAABAAAAAAAAABAAAAAPoAAA
D6AAAA+gAAAPoAAAAAAACAAAAAwAAABQAAwABAAAAFAAEACgAAAAEAAQAAQAA5wP//wAA5wD//wA
AAAEABAAAAAEAAgADAAQAAAAAAI4AwgEAASYAAwAA//oDNQPsAA4AHQBYAAALHgEOAScmJy4BNz4
BMzIFFgYHBgcGLgE2NzYzMhYBHgEXDgEHDgEHDgIWFxYXFjY3NjQ3PgE3HgEXFhQXHgE3PgE3PgE
uAScuAScuASc+ATc+AQcLASYWAVEfFxo6IBkNCQIHCy8bCQG9BwIJDRkgOhoXHwoKgi/+TRlRDyE
OIxo+ExckFAQMfIkWvhcMBwYlFRYkBWcMf1YwFCALDAQUIxcUPhojDiAOUR4cAQvEwswB6gtDTyc
JCBsSKxYhJ0gWKxIaCQknUEILAYcCf2TPI0w2HBUMdg0sOzsakQ4ONzcniiYXNBgYNBcmiiyc3OA8
GHRQaOzssDQ4mFRw2TiLOZGdBA/5vAZEDQQAEEAAAAAQA+kABQANABcAHwAAARUzFSErAYERIZU
jNSEBIREhESMVITUjMyMVITUjNSMC733+iT8B9D4+/oj+igE4AXc//c4++j8BOT+7AbZ8+gF2/ks
Bdz4//ksB9AF2fHw+Pj8AAAIAAAAAA7cD6QACACQAAAEhEwMOAQcVITUmJyY1ND8BIRcWFxYVFAC
GKwEVITUmJyYnASMCKP8AguQrOy0BGkIRHREkASstEgEEDhQxEQGaJxUcLP7PDAFNAVL+PHBHCBS
bBgsUKR8wX3owBg4NFgsQGxsDFx1zAyMAAAACAAAAAAPKA+oAAgATAAABFxEbDgEHHgEXETMRMxE
zETM1IQL+zPlabpADA5t0f2F+XP41AfBMAZgBJwmYCHSbA/48A2r8lgNqfgAAAAASAN4AAQAAAAA
AAAABAAAAQAAAAAAQAJAEEAAQAAAAAAAgAHAAoAAQAAAAAAAwAJABEAAQAAAAABAAJABoAAQA
AAAAABQALACMAAQAAAAABgAJAC4AAQAAAAAAACgAsADcAAQAAAAAAACwASAGMAAwABBAkAAAAACAHU
AAwABBAkAAQASAHcAAwABBAkAAgAOAIkAAwABBAkAAwASAJcAAwABBAkABAASAKkAAwABBAkABQA
WALSAAwABBAkABgASANEAAwABBAkACgBYAOMAawABBAkACwAkATsgZGRiLWljB25zUmVndWxhcmlR
kYilpY29uc2RkYilpY29uc1ZlcnpB24gMS4wZGRiLWljB25zRm9udCBnZW5lcmF0ZWQgdXNpbmc
gU3luY2Zlc2lubiBNZXRYbyBTdHVkaW93d3cuc3luY2Zlc2lubi5jb20AIABkAGQAYgAtAGkAYwB
vAG4AcwBSAGUAZwB1AGwAYQByAGQAZABiAC0AaQBJAG8AbgBzAGQAZABiAC0AaQBJAG8AbgBzAFY
AZQByAHMAaQBvAG4AIAAxAAC4AMABkAGQAYgAtAGkAYwBvAG4AcwBGAG8AbgB0ACAAZwB1AG4AZQB
yAGEAdABlAGQAIAB1AHMAaQBvAG4CAIABTAHkAbgBjAGYAdQBzAGkAbwBuACAATQBlAHQAcgBvACA
AUwB0AHUAZABpAG8AdwB3AHcALgBzAHkAbgBjAGYAdQBzAGkAbwBuAC4AYwBvAG0AAAAAAgAAAAA
AAAAKAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAFAQIBAwEEAQUBBgADY3V0CHBhc3RlXzAxZGZvbG9u
OcGFyYS1tYXJrLS0tMDMAAA==) format('trueType');
    font-weight: normal;
    font-style: normal;
}
.e-cm-icons {
    font-family: 'ddb-icons' !important;
    speak: none;
    font-size: 55px;
    font-style: normal;
    font-weight: normal;
    font-variant: normal;
    text-transform: none;
    line-height: 1;
    -webkit-font-smoothing: antialiased;
    -moz-osx-font-smoothing: grayscale;
}
.e-cut::before {
    content: '\e700';
}
.e-copy::before {
    content: '\e70a';
}
.e-paste::before {
    content: '\e701';
}
}
</style>

```

{% previewsample "page.domainurl/code-snippet/context-menu/default-cs11" %}

## Navigation

Navigation in ContextMenu is usage to navigate to the other web page when menu item is clicked. This can be achieved by providing link to the menu item using the [url](#) property. In the following sample, Navigation URL for Flipkart, Amazon, and Snapdeal menu items are added using the [url](#) property.

### APP.VUE

```
<template>
<div>
<div id="target">Right click / Touch hold to open the ContextMenu</div>
<ejs-contextmenu target='#target' :items='menuItems'
:beforeItemRender='onBeforeItemRender'></ejs-contextmenu>
</div>
</template>
<script>
import Vue from 'vue';
import { ContextMenuPlugin } from "@syncfusion/ej2-vue-navigations";
import { enableRipple } from '@syncfusion/ej2-base';
enableRipple(true);
Vue.use(ContextMenuPlugin);
export default {
  data () {
    return {
      menuItems:[
        {
          text: 'Flipkart',
          iconCss: 'e-cart-icon e-link',
          url: 'https://www.google.co.in/search?q=flipkart'
        },
        {
          text: 'Amazon',
          iconCss: 'e-cart-icon e-link',
          url: 'https://www.google.co.in/search?q=amazon'
        },
        {
          text: 'Snapdeal',
          iconCss: 'e-cart-icon e-link',
          url: 'https://www.google.co.in/search?q=snapdeal'
        }
      ]
    };
  },
  methods: {
    onBeforeItemRender: function(args) {
      args.element.getElementsByTagName('a')[0].setAttribute('target',
'_blank');
    }
  }
}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-buttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-inputs/styles/material.css";
@import "../node_modules/@syncfusion/ej2-popups/styles/material.css";
@import "../node_modules/@syncfusion/ej2-navigations/styles/material.css";
#target {
```

```
border: 1px dashed;
height: 150px;
padding: 10px;
position: relative;
text-align: justify;
color: gray;
user-select: none;
}
@font-face {
font-family: 'cart';
src:
url(data:application/x-font-ttf;charset=utf-
8;base64,AAEAAAAKAIAAAwAgTlMvMj0gSQ4AAAEoAAAVmNtYXdDnEODVAAABiAAAADZnbHlmgat
ngwAAACgAAADYaGVhZBktP4wAAADQAAAAANmhoZWEHmQNpAAAArAAAACRobXR4B+j//gAAAYAAAA
IbG9jYQBsAAAAAAHAAAAABmIheHABDwBQAAABCAAAACBuYWllfiV2lQAAAQAAAAIBcG9zdIZzcJA
AAASkAAAAOgABAAADUv9qAfOEAP/+//wD7AABAAAAAAAAAAAAAAAAAAAAAAAAAgABAAAAAQAA2UwSaF8
PPPUACwPoAAAAANfSfWUAAAAA19J9Zf/+AAAD7APdAAAAACAACAAAAAAAAAAAAEAAAAACAEQAAwAAAAA
AAgAAAAoACgAAAP8AAAAAAAAAAAAAQp0AZAABQAAANoCvAAAAIwCegK8AAAB4AAxAQIAAAIABQMAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAUGZFZABA5wDnAANS/2oAWGpDAJYAAAABAAAAAAAAABAAAAAPo//4
AAAAACAAAAAwAAABQAAwABAAAAFAAEACIAAAAEAAQAAQAA5wD//wAA5wD//wAAAAEABAAAAAEAAAA
AAAAAbAAAAAP//gAAA+wD3QAABIAQwAAAJR4BMjY0JicOAQUeATI2NCYiBgEOAwclIgyYXEx4BMwU
yFgcOAQclIgyYXBhYXBT4BPwE2PwI2NxM+AycmIyIGAEABJzonJx0gJ/6jASc6Jyc6JwMXIDgZPyX
9giQkBgkII0yQBACQgCQo/JP7RIxcBARcjAVgkQg0QDgkICgkNkw4xMBwCBScJE1UdJyc6JwEBJx0
dJyc6JycDZQg0PigBAy0k/uAkMAQnHRsMAQMTDA8QAQMBlCilIhYWFxYhAYciNg4YDBMCAAAAEgD
eAAEAAAAAAAAAAAAEAAAAAAAAEABAAABAAEAAAAAAAAIABwAFAAEAAAAAAAAAMABAAMAAEAAAAAAAAQ
ABAAQAAEAAAAAAAAUACwAUAAEAAAAAAAAAYABAAfAAEAAAAAAAAoALAAjAAEAAAAAAAAsAEgBPAAMAAQ
JAAAAAgBhAAMAAQQJAAAEACABjAAMAAQQJAAIADgBrAAMAAQQJAAAMACAB5AAMAAQQJAAQACACBAAM
AAQQJAAUAFgCJAAMAAQQJAAAYACACfAAMAAQQJAAoAWACnAAMAAQQJAAAsAJAD/IGNhcnRSZWdlbGF
yY2FydGNhcnRWXJzaW9uIDEuMGNhcnRlb250IGdlbmVYXRlZCB1c2luZyBTeW5jZnVzaW9uIE1
ldHJvIFN0dWRpb3d3dy5zeW5jZnVzaW9uLmNvbQAgAGMAYQByAHQAUGBlAGcAdQBsaGEAcgBjAGE
AcgB0AGMAYQByAHQAUGBlAHIAcwbPAG8AbgAgADEALgAwAGMAYQByAHQARgBvAG4AdAAgAGcAZQB
uAGUAcgBhAHQAQZQBkACAAQZQBzAGkAbgBnACAAUwB5AG4AYwBmAHUAcwBpAG8AbgAgAE0AZQB0AHI
AbwAgAFZMAcABIAQQAaQbVbAHcAdwB3AC4AcwB5AG4AYwBmAHUAcwBpAG8AbgAgAGMAbwBtAAAAAAI
AAAAAAAAACgAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAgECAQMAEHNob3BwaW5nLWNhcnQtMDUAAAAA
A) format('trueType');
font-weight: normal;
font-style: normal;
}
.e-cart-icon {
font-family: 'cart' !important;
speak: none;
font-size: 55px;
font-style: normal;
font-weight: normal;
font-variant: normal;
text-transform: none;
line-height: 1;
-webkit-font-smoothing: antialiased;
-moz-osx-font-smoothing: grayscale;
}
.e-link::before {
content: '\e700';
}
</style>
```

```
{% previewsample "page.domainurl/code-snippet/context-menu/default-cs12" %}
```

To open the links in new tab, set `target` attribute with the value `_blank` in the [beforeItemRender](#) event.

See Also

- [How to change menu items dynamically](#)

## Template and multilevel nesting in Vue Context menu component

### Template

The ContextMenu items can be customized by using the [Render](#) event. The item render event triggers while rendering each menu item. The event argument will be used to identify the menu item and customize it based on the requirement. In the following sample, the menu item is rendered with keycode for specified action in ContextMenu using the template. Here, the keycode is specified for Save as, View page source, and Inspect in the right side corner of the menu items by adding span element in the [beforeItemRender](#) event.

### APP.VUE

```
<template>
<div>
<div id="target">Right click / Touch hold to open the ContextMenu</div>
<ejs-contextmenu target='#target' :items='menuItems'
:beforeItemRender="onBeforeItemRender"></ejs-contextmenu>
</div>
</template>
<script>
import Vue from 'vue';
import { ContextMenuPlugin } from "@syncfusion/ej2-vue-navigations";
import { enableRipple, createElement } from '@syncfusion/ej2-base';
enableRipple(true);
Vue.use(ContextMenuPlugin);
export default {
  data () {
    return {
      menuItems:[
        {
          text: 'Save as...'
        },
        {
          text: 'View page source'
        },
        {
          text: 'Inspect'
        }
      ]
    };
  },
  methods: {
    onBeforeItemRender: function(args) {
      var shortCutSpan = createElement('span');
      var text = args.item.text;
      var shortCutText = text === 'Save as...' ? 'Ctrl + S' : (text
      === 'View page source' ?
      'Ctrl + U' : 'Ctrl + Shift + I');
      shortCutSpan.textContent = shortCutText;
      args.element.appendChild(shortCutSpan);
      shortCutSpan.setAttribute('class','shortcut');
```

```

    }
  }
}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-buttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-inputs/styles/material.css";
@import "../node_modules/@syncfusion/ej2-popups/styles/material.css";
@import "../node_modules/@syncfusion/ej2-navigations/styles/material.css";
#target {
  border: 1px dashed;
  height: 150px;
  padding: 10px;
  position: relative;
  text-align: justify;
  color: gray;
  user-select: none;
}
.shortcut {
  float: right;
  font-size: 10px;
  opacity: 0.5;
  padding-left: 50px;
}
</style>

```

{% previewsample "page.domainurl/code-snippet/context-menu/default-cs13" %}

To create span element, `createElement` utility function used from `ej2-base`.

### Multilevel nesting

The Multiple level nesting supports in ContextMenu. It can be achieved by mapping the `items` property inside the parent `menuItems`. In the below sample, three level nesting of ContextMenu is provided.

### APP.VUE

```

<template>
<div>
<div id="target">Right click / Touch hold to open the ContextMenu</div>
<ejs-contextmenu target='#target' :items='menuItems'></ejs-contextmenu>
</div>
</template>
<script>
import Vue from 'vue';
import { ContextMenuPlugin } from "@syncfusion/ej2-vue-navigations";
import { enableRipple } from '@syncfusion/ej2-base';
enableRipple(true);
Vue.use(ContextMenuPlugin);
export default {
  data () {
    return {
      menuItems: [
        {
          text: 'Show All Bookmarks',

```

```

        {
            text: 'Bookmarks Toolbar',
            items: [
                {
                    text: 'Most Visited',
                    items: [
                        {
                            text: 'Google',
                        },
                        {
                            text: 'Gmail'
                        }
                    ]
                },
                {
                    text: 'Recently Added'
                }
            ]
        }
    ];
}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-buttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-inputs/styles/material.css";
@import "../node_modules/@syncfusion/ej2-popups/styles/material.css";
@import "../node_modules/@syncfusion/ej2-navigations/styles/material.css";
#target {
    border: 1px dashed;
    height: 150px;
    padding: 10px;
    position: relative;
    text-align: justify;
    color: gray;
    user-select: none;
}
</style>

```

{% previewsample "page.domainurl/code-snippet/context-menu/default-cs14" %}

### Accessibility in Vue Context menu component

The Context menu component followed the accessibility guidelines and standards, including [ADA](#), [Section 508](#), [WCAG 2.2](#) standards, and [WCAG roles](#) that are commonly used to evaluate accessibility.

The accessibility compliance for the Context menu component is outlined below.

| Accessibility Criteria | Compatibility |

| -- | -- |

| [WCAG 2.2](#) Support |  |

| [Section 508](#) Support |  |

| Screen Reader Support | ` |`

| Right-To-Left Support | ` |`

| Color Contrast | ` |`

| Mobile Device Support | ` |`

| Keyboard Navigation Support | ` |`

| [Accessibility Checker](#) Validation | ` |`

| [Axe-core](#) Accessibility Validation | ` |`

```
<style>
.post .post-content img {
display: inline-block;
margin: 0.5em 0;
}
</style>

<div> - All
features of the component meet the requirement.</div>

<div> - Some features of the component do not meet the requirement.</div>

<div> - The
component does not meet the requirement.</div>
```

### WAI-ARIA attributes

The Context menu component followed the [WAI-ARIA](#) patterns to meet the accessibility. The following ARIA attributes are used in the Context menu component:

Attributes	Purpose
role	Indicates Context menu component popup as <code>menu</code> , and the popup items as <code>menuitem</code> .
aria-haspopup	Indicates the availability and type of interactive popup element.
aria-expanded	Indicates whether the subtree can be expanded or collapsed, as well as indicates whether its current state is expanded or collapsed.
aria-label	Indicates the menu item text.

### Keyboard interaction

The Context menu component followed the [keyboard interaction](#) guideline, making it easy for people who use assistive technologies (AT) and those who completely rely on keyboard navigation. The following keyboard shortcuts are supported by the Context menu component.

| **Press** | **To do this** |

| --- | --- |

| **Esc** | **Closes the opened sub menu.** |

| **Enter** | **Selects the focused item.** |

| **Up** | **Navigates up or to the previous menu item.** |

| **Down** | **Navigates down or to the next menu item.** |

| **Left** | **Close the current sub menu and navigates to the parent menu.** |

| **Right** | **Navigates and open the next sub menu.** |

### Ensuring accessibility

The Context menu component's accessibility levels are ensured through an [accessibility-checker](#) and [axe-core](#) software tools during automated testing.

The accessibility compliance of the Context menu component is shown in the following sample. Open the [sample](#) in a new window to evaluate the accessibility of the Context menu component with accessibility tools.

```
{% previewsample "page.domainurl/code-snippet/context-menu/default-cs1" %}
```

See also

- [Accessibility in Syncfusion Vue components](#)

### Style and appearance in Vue Context menu component

To modify the ContextMenu appearance, you need to override the default CSS of ContextMenu component. Please find the list of CSS classes and its corresponding section in ContextMenu component. Also, you have an option to create your own custom theme for the controls using our [Theme Studio](#).

CSS Class | Purpose of Class

|.e-contextmenu-wrapper |To customize the context menu wrapper

|.e-contextmenu-wrapper .e-menu-parent|To customize the context menu items

|.e-contextmenu-wrapper ul .e-menu-item.e-selected .e-caret::before|To customize the context menu caret icon

|.e-contextmenu-wrapper ul .e-menu-item .e-menu-icon::before|To customize the icons of the context menu

### How To

#### Open and close contextmenu in Vue Context menu component

ContextMenu can be opened and closed programmatically whenever required by using the open and close methods.



Install Syncfusion **Button** packages using below command.

```
`bash
```

```
npm install @syncfusion/ej2-vue-buttons --save
```

```
`
```

In the following example, the ContextMenu is opened using the [open](#) method at the specified position using **top** and **left**. Also, ContextMenu is closed using [close](#) method on ContextMenu item click or document click.

### APP.VUE

```
<template>
<div>
<ejs-contextmenu id='cmenu' :items='menuItems'></ejs-contextmenu>
<ejs-button v-on:click.native='btnClick'>Open ContextMenu</ejs-button>
</div>
</template>
<script>
import Vue from 'vue';
import { ContextMenuPlugin } from "@syncfusion/ej2-vue-navigations";
import { ButtonPlugin } from "@syncfusion/ej2-vue-buttons";
import { enableRipple } from '@syncfusion/ej2-base';
enableRipple(true);
Vue.use(ContextMenuPlugin);
Vue.use(ButtonPlugin);
export default {
  data () {
    return {
      menuItems:[
        {
          text: 'Cut'
        },
        {
          text: 'Copy'
        },
        {
          text: 'Paste'
        }
      ]
    };
  },
  methods: {
    btnClick: function(event) {
      document.getElementById('cmenu').ej2_instances[0].open(40, 20);
    }
  }
}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-buttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-inputs/styles/material.css";
@import "../node_modules/@syncfusion/ej2-popups/styles/material.css";
@import "../node_modules/@syncfusion/ej2-navigations/styles/material.css";
#target {
```

```
border: 1px dashed;
height: 150px;
padding: 10px;
position: relative;
text-align: justify;
color: gray;
user-select: none;
}
</style>
```

```
{% previewsample "page.domainurl/code-snippet/context-menu/default-cs7" %}
```

### Change menu items dynamically in Vue Context menu component

The items visible in the ContextMenu can be changed dynamically based on the target in which you open the ContextMenu. To achieve this behavior, initialize ContextMenu with all items using [items](#) property and then based on the context you open hide/show required items using [hideItems](#)/[showItems](#) method in [beforeOpen](#) event.

In the following example, the datasource for Clipboard div is **Cut**, **Copy**, **Paste** and for the Editor div is **Add**, **Edit**, **Delete** is changed on [beforeOpen](#) event using [hideItems](#) and [showItems](#) method.

### APP.VUE

```
<template>
<div>
  <div id="target">
    <div id='left' class='e-div'>Clipboard</div>
    <div id='right' class='e-div'>Editor</div>
  </div>
  <ejs-contextmenu id='cmenu' target='#target' :items='menuItems'
:beforeOpen="onBeforeOpen"></ejs-contextmenu>
</div>
</template>
<script>
import Vue from 'vue';
import { ContextMenuPlugin } from "@syncfusion/ej2-vue-navigations";
import { enableRipple } from '@syncfusion/ej2-base';
enableRipple(true);
Vue.use(ContextMenuPlugin);
export default {
  data () {
    return {
      menuItems: [
        {
          text: 'Cut'
        },
        {
          text: 'Copy'
        },
        {
          text: 'Paste'
        },
        {
          text: 'Add'
        },
        {

```

```

        text: 'Edit'
      },
      {
        text: 'Delete'
      }
    ]
  };
},
methods: {
  onBeforeOpen: function(args) {
    var menuObj = document.getElementById("cmenu").ej2_instances[0];
    // To hide/show items on right click.
    if ((args.event.target).id === 'right') {
      menuObj.hideItems(['Cut', 'Copy', 'Paste']);
      menuObj.showItems(['Add', 'Edit', 'Delete']);
    } else if (args.event.target.id === 'left') {
      menuObj.showItems(['Cut', 'Copy', 'Paste']);
      menuObj.hideItems(['Add', 'Edit', 'Delete']);
    }
  }
}
}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-buttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-inputs/styles/material.css";
@import "../node_modules/@syncfusion/ej2-popups/styles/material.css";
@import "../node_modules/@syncfusion/ej2-navigations/styles/material.css";
#target {
  user-select: none;
}
.e-div {
  width: 40%;
  border: 1px dashed;
  height: 150px;
  margin: 10px;
  text-align: center;
  line-height: 150px;
  display: inline-block;
}
</style>

```

{% previewsample "page.domainurl/code-snippet/context-menu/default-cs5" %}

Template in Vue Context menu component

[Show table in sub ContextMenu](#)

Menu items of the ContextMenu can be customized according to the requirement. The section explains about how to customize table template

in sub menu item.

This can be achieved by appending table layout while `li` rendering by using [beforeItemRender](#) event.

## **APP.VUE**

```

<template>
<div>
<div id="target">Right click / Touch hold to open the ContextMenu</div>
<ejs-contextmenu target='#target' :items='menuItems'
:beforeItemRender="onBeforeItemRender"></ejs-contextmenu>
</div>
</template>
<script>
import Vue from 'vue';
import { ContextMenuPlugin } from "@syncfusion/ej2-vue-navigations";
import { enableRipple } from '@syncfusion/ej2-base';
enableRipple(true);
Vue.use(ContextMenuPlugin);
export default {
  data () {
    return {
      menuItems:[
        {
          text: 'Cut',
          iconCss: 'e-cm-icons e-cut'
        },
        {
          text: 'Copy',
          iconCss: 'e-icons e-copy'
        },
        {
          text: 'Paste',
          iconCss: 'e-cm-icons e-paste'
        },
        {
          separator: true
        },
        {
          text: 'Link',
          iconCss: 'e-icons e-link'
        },
        {
          text: 'Table',
          iconCss: 'e-icons e-table',
          items: [
            {
              id: 'table'
            }
          ]
        }
      ]
    };
  },
  methods: {
    onBeforeItemRender: function(args) {
      // To create header element.
      var header = document.createElement('h4');
      header.textContent = 'Insert Table';
      // To create table with five rows and six columns.
      var table = document.createElement('table');
      for (var i = 0; i < 5; i++) {
        var row = document.createElement('tr');
        table.appendChild(row);
      }
    }
  }
}

```

```

        for (var j = 0; j < 6; j++) {
            var col = document.createElement('td');
            row.appendChild(col);
            col.setAttribute('class', 'e-border');
        }
    }
    // To append table on `li` rendering.
    if (args.item.id === 'table') {
        args.element.classList.add('bg-transparent');
        args.element.appendChild(header);
        args.element.appendChild(table);
    }
}
}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-buttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-inputs/styles/material.css";
@import "../node_modules/@syncfusion/ej2-popups/styles/material.css";
@import "../node_modules/@syncfusion/ej2-navigations/styles/material.css";
.list {
    display: none;
}
.e-border {
    border: 1px solid rgba(0, 0, 0, 0.87);
    padding: 8px;
}
#target {
    border: 1px dashed;
    height: 150px;
    padding: 10px;
    position: relative;
    text-align: justify;
    color: gray;
    user-select: none;
}
@font-face {
    font-family: 'ddb-icons';
    src:
        url(data:application/x-font-ttf;charset=utf-
8;base64,AAEAAAAKAIAAAwAgTlMvMj0gSRkAAAEoAAAAVmNtYXDnE+dkAAABlAAADxnbHlmlh3
3NQAAAdwAAAjMAgVhZBKOK9sAAADQAAAAANmhoZWEHeANwAAAArAAAACRobXR4E6AAAAAAAYAAAA
UbG9jYQGOAegAAAHQAAAADG1heHABEwBlAAABCAAAACBuYW1l1LBM9QAABCgAAAI9cG9zdMJntbU
AAAZoAAAAUAABAAADUv9qAFoEAAAAAADygABAAAAAAAAAAAAAAAAAAAAABQABAAAAAQAAojXaQl8
PPPUACwPoAAAAANfSc4gAAAAA19JziAAA//oDygPsAAAACAACAAAAAAAAAAAAEAAAFkABAAAAAA
AAgAAAAoACgAAAP8AAAAAAAAAAAAAQPtAZAABQAAANoCvAAAAIwCegK8AAAB4AAxAQIAAAIABQMAAAA
AAAAAAAAAAAAAAAAAAAAAUGZFZABA5wDnAwNS/2oAWgPsAJYAAAABAAAAAAAAAAAAAABAAAAAPoAAA
D6AAAA+gAAAPoAAAAAAACAAAAAwAAABQAAAwABAAAAFAAEACgAAAAEAAQAA5wP//wAA5wD//wA
AAAEABAAAAAEAAgADAAQAAAAAAI4AwgEAAASYAAwAA//oDNQPsAA4AHQBYAAAlHgEOAScmJy4BNz4
BMzIFFgYHBgcGLgE2NzYzYmYBhgEXDgEHDgEHDgIWFxYXFjY3NjQ3PgE3HgEXFhQXHgE3PgE3PgE
uAScuAScuASc+ATc+AQcLASYWAVEfFx06IBkNCQIHCy8bCQG9BwIJDRkgOhoXHwoKgi/+TR1RDyE
OIxo+ExckFAQMfikwVhCMBwYlFRYkBWCMF1YwFCALDAQUIxcUPhojDiAOUR4cAQvEwWSB6gtDTyc
JCBsSKxYhJ0gWKxIaCQknUEILAYcCf2TPI0w2HBUMDg0sOZsaKQ4ONZcniyYXNBgYNBcmiyc3OA8
GHRQaOzssDQ4mFRw2TiLOZGdBA/5vAZEDQQAEEAAAAA0qA+kABQANABcAHwAAARUzFSErAYERIZU
jNSEBIREhESMVITUjMyMVITUjNSMC733+iT8B9D4+/oj+igE4AXc//c4++j8BOT+7AbZ8+gF2/ks

```

```

Bdz4//ksB9AF2fHw+Pj8AAAIAAAAAA7cD6QACACQAAAEhEwMOAQcVITUmJyY1ND8BIRcWFxYVFAC
GKwEVITUmJyYnASMCKP8AguQrOy0BGkIRHREkASstEgEEDhQxEQGaJxUcLP7PDAFNAVL+PHBHCBS
bBgsUKR8wX3owBg4NFgsQGxsDFx1zAyMAAAACAAAAAPKA+oAAgATAAABFxEBDgEHHgEXETMRMxE
zETM1IQL+zP1abpADA5t0f2F+XP41AfbMAZgBJwmYcHSbA/48A2r8lgNqfgAAAAASAN4AAQAAAAA
AAAAABAAAAQAAAAAAQAJAAEAAQAAAAAAAgAHAAoAAQAAAAAAAwAJABEAAQAAAAABAAJABoAAQA
AAAAABQALACMAAQAAAAABgAJAC4AAQAAAAAACgAsADcAAQAAAAAACwASAGMAAwABBAkAAAAACAHU
AAwABBAkAAQASAHcAAwABBAkAAgAOAIkAAwABBAkAAwASAJcAAwABBAkABAASAKkAAwABBAkABQA
WALsAAwABBAkABgASANEAAwABBAkACgBYAOMAAwABBAkACwAkATsgZGRiLWlj25zUmVndWxhcmR
kYilpY29uc2RkYilpY29uc1ZlcnNpb24gMS4wZGRiLWlj25zRm9udCBnZW5lcmF0ZWQgdXNpbmc
gU3luY2Zlc2lubiBNZXRYbyBTdHVkaW93d3cuc3luY2Zlc2lubi5jb20AIABkAGQAYgAtAGkAYwB
vAG4AcwBSAGUAZwB1AGwAYQByAGQAZABiAC0AaQBjAG8AbgBzAGQAZABiAC0AaQBjAG8AbgBzAFY
AZQByAHMAaQBvAG4AIAAxAAC4AMABkAGQAYgAtAGkAYwBvAG4AcwBGAG8AbgB0ACAAZwB1AG4AZQB
yAGEAdABLAGQAIAB1AHMAaQBvAGcAIABTAHkAbgBjAGYAdQBzAGkAbwBuACAATQB1AHQAcgBvACA
AUwB0AHUAZABpAG8AdwB3AHcALgBzAHkAbgBjAGYAdQBzAGkAbwBuAC4AYwBvAG0AAAAAAGAAAA
AAAAKAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAFAQIBAwEEAQUBBgADY3V0CHBhc3RlXzAxBGZvbncQ
OcGFyYS1tYXJrLS0tMDMAAA==) format('true');
    font-weight: normal;
    font-style: normal;
}
.e-cm-icons {
    font-family: 'ddb-icons' !important;
    speak: none;
    font-size: 55px;
    font-style: normal;
    font-weight: normal;
    font-variant: normal;
    text-transform: none;
    line-height: 1;
    -webkit-font-smoothing: antialiased;
    -moz-osx-font-smoothing: grayscale;
}
.e-cut::before {
    content: '\e700';
}
.e-copy::before {
    content: '\e70a';
}
.e-paste::before {
    content: '\e701';
}
.e-link::before {
    content: '\e290';
}
.e-table::before {
    content: '\e705';
}
.e-contextmenu-wrapper ul .e-menu-item.bg-transparent {
    background-color: transparent;
    line-height: normal;
    height: auto;
}
h4 {
    text-align: center;
    margin-top: 5px;
    margin-bottom: 5px;
}
</style>

```

```
{% previewsample "page.domainurl/code-snippet/context-menu/default-cs8" %}
```

*Show UI components in ContextMenu*

UI components can also be placed inside the each `li` element of ContextMenu.

In the following example, CheckBox component is placed inside each `li` element and this can be achieved by creating CheckBox component in [beforeItemRender](#) event and appending it into the `li` element.

### APP.VUE

```
<template>
<div>
<div id="target">Right click / Touch hold to open the ContextMenu</div>
<ejs-contextmenu target='#target' :items='menuItems'
:beforeItemRender='itemRender' :beforeClose='beforeClose'></ejs-contextmenu>
</div>
</template>
<script>
import Vue from 'vue';
import { ContextMenuPlugin } from "@syncfusion/ej2-vue-navigations";
import { enableRipple, closest, createElement } from '@syncfusion/ej2-base';
import { createCheckBox } from '@syncfusion/ej2-buttons';
enableRipple(true);
Vue.use(ContextMenuPlugin);
export default {
  data () {
    return {
      menuItems:[
        { text: 'Option 1' },
        { text: 'Option 2' },
        { text: 'Option 3' }
      ]
    };
  },
  methods: {
    itemRender: function(args) {
      var check = createCheckBox(createElement, false, {
        label: args.item.text,
        checked: (args.item.text == 'Option 2') ? true : false
      });
      args.element.innerHTML = '';
      args.element.appendChild(check);
    },
    beforeClose: function(args) {
      if (args.event.target.closest('.e-menu-item')) {
        args.cancel = true;
        var selectedElem, i, checkbox, ele, frame;
        selectedElem = args.element.querySelectorAll('.e-selected');
        for(i = 0; i < selectedElem.length; i++){
          ele = selectedElem[i];
          ele.classList.remove('e-selected');
        }
        checkbox = closest(args.event.target, '.e-checkbox-wrapper');
        frame = checkbox && checkbox.querySelector('.e-frame');
```

```

        if (checkbox && frame.classList.contains('e-check')) {
            frame.classList.remove('e-check');
        } else if (checkbox) {
            frame.classList.add('e-check');
        }
    }
}
}
}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-buttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-inputs/styles/material.css";
@import "../node_modules/@syncfusion/ej2-popups/styles/material.css";
@import "../node_modules/@syncfusion/ej2-navigations/styles/material.css";
.list {
    display: none;
}
#target {
    border: 1px dashed;
    height: 150px;
    padding: 10px;
    position: relative;
    text-align: justify;
    color: gray;
    user-select: none;
}
</style>

```

{% previewsample "page.domainurl/code-snippet/context-menu/default-cs9" %}

Underline a character in the item text in Vue Context menu component

Underline a particular character in a text can be handled in [beforeItemRender](#) event by adding `<u>` tag in between the text and given as innerHTML in `li` rendering.

#### APP.VUE

```

<template>
<div>
<div id="target">Right click / Touch hold to open the ContextMenu</div>
<ejs-contextmenu target='#target' :items='menuItems'
:beforeItemRender='itemRender'></ejs-contextmenu>
</div>
</template>
<script>
import Vue from 'vue';
import { ContextMenuPlugin } from "@syncfusion/ej2-vue-navigations";
import { enableRipple } from '@syncfusion/ej2-base';
enableRipple(true);
Vue.use(ContextMenuPlugin);
export default {
    data () {
        return {
            menuItems: [
                {

```



```

        text: 'Cut'
      },
      {
        text: 'Copy'
      },
      {
        text: 'Paste'
      }
    ]
  };
},
methods: {
  itemRender: function(args ) {
    if (args.item.text === 'Copy') {
      // To underline a particular character.
      args.element.innerHTML = '<u>C</u>opy';
    }
  }
}
}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-buttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-inputs/styles/material.css";
@import "../node_modules/@syncfusion/ej2-popups/styles/material.css";
@import "../node_modules/@syncfusion/ej2-navigations/styles/material.css";
.list {
  display: none;
}
#target {
  border: 1px dashed;
  height: 150px;
  padding: 10px;
  position: relative;
  text-align: justify;
  color: gray;
  user-select: none;
}
</style>

```

{% previewsample "page.domainurl/code-snippet/context-menu/default-cs10" %}

Open a dialog on contextmenu item click in Vue Context menu component

This section explains about how to open a dialog on ContextMenu item click. This can be achieved by handling dialog open in [select](#) event of the ContextMenu.

Install Syncfusion **Popups** packages using below command.

```
`bash
```

```
npm install @syncfusion/ej2-vue-popups --save
```

```
,
```

In the following sample, Dialog will open while clicking **Save As...** item.

**APP.VUE**

```

<template>
<div>
<div id="target">Right click / Touch hold to open the ContextMenu</div>
<ejs-dialog id="dialog" content="This file can be saved as PDF"
:buttons='buttons' width='200px' height='110px' :visible='false'></ejs-
dialog>
<ejs-contextmenu target='#target' :items='menuItems'
:select='onSelect'></ejs-contextmenu>
</div>
</template>
<script>
import Vue from 'vue';
import { ContextMenuPlugin } from "@syncfusion/ej2-vue-navigations";
import { DialogPlugin } from "@syncfusion/ej2-vue-popups";
import { enableRipple } from '@syncfusion/ej2-base';
enableRipple(true);
Vue.use(ContextMenuPlugin);
Vue.use(DialogPlugin);
export default {
  data () {
    return {
      menuItems:[
        {
          text: 'Back'
        },
        {
          text: 'Forward'
        },
        {
          text: 'Reload'
        },
        {
          separator: true
        },
        {
          text: 'Save As...'
        },
        {
          text: 'Print'
        },
        {
          text: 'Cast'
        }
      ],
      buttons: [{
        buttonModel: {
          isPrimary: true,
          content: 'Submit',
          cssClass: 'e-flat',
        },
        click: function () {
          this.hide();
        }
      }],
    };
  },
  methods: {
    onSelect: function(args) {

```

```

        if(args.item.text === 'Save As...') {
            if (document.getElementById('dialog_dialog-content')) {

document.getElementById("dialog").ej2_instances[0].show();
            }
            else {

document.getElementById("dialog").ej2_instances[0].appendTo('#dialog');
            }
        }
    }
}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-buttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-inputs/styles/material.css";
@import "../node_modules/@syncfusion/ej2-popups/styles/material.css";
@import "../node_modules/@syncfusion/ej2-navigations/styles/material.css";
.list {
    display: none;
}
#target {
    border: 1px dashed;
    height: 150px;
    padding: 10px;
    position: relative;
    text-align: justify;
    color: gray;
    user-select: none;
}
</style>

```

{% previewsample "page.domainurl/code-snippet/context-menu/default-cs6" %}

### Change animation settings in Vue Context menu component

To change the animation of the ContextMenu, [animationSettings](#) property

is used. The supported effects for ContextMenu are,

| Effect | Functionality |

| ----- | ----- |

| None | Specifies the sub menu transform with no animation effect. |

| SlideDown | Specifies the sub menu transform with slide down effect. |

| ZoomIn | Specifies the sub menu transform with zoom in effect. |

| FadeIn | Specifies the sub menu transform with fade in effect. |

The following sample illustrates how to open ContextMenu with **FadeIn** effect with the **duration** of **800ms**.

### APP.VUE

```

<template>
<div>
<div id="target">Right click / Touch hold to open the ContextMenu</div>
<ejs-contextmenu target='#target' :items='menuItems'
:animationSettings='animationSettings'></ejs-contextmenu>
</div>
</template>
<script>
import Vue from 'vue';
import { ContextMenuPlugin } from "@syncfusion/ej2-vue-navigations";
import { MenuEventArgs } from '@syncfusion/ej2-navigations';
import { enableRipple } from '@syncfusion/ej2-base';
enableRipple(true);
Vue.use(ContextMenuPlugin);
export default {
  data () {
    return {
      menuItems:[
        {
          text: 'Show All Bookmarks'
        },
        {
          text: 'Bookmarks Toolbar',
          items: [
            {
              text: 'Most Visited',
              items:[
                {
                  text: 'Gmail'
                },
                {
                  text: 'Google'
                }
              ]
            }
          ],
        },
        {
          text: 'Recently Added'
        }
      ],
      animationSettings: { effect: 'FadeIn', duration: 800 }
    }
  },
};
</script>
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-buttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-inputs/styles/material.css";
@import "../node_modules/@syncfusion/ej2-popups/styles/material.css";
@import "../node_modules/@syncfusion/ej2-navigations/styles/material.css";
#target {
  border: 1px dashed;
  height: 150px;
  padding: 10px;
  position: relative;
  text-align: justify;

```

```
color: gray;
user-select: none;
}
</style>
```

{% previewsample "page.domainurl/code-snippet/context-menu/default-cs4" %}

## Dashboard layout

### Getting Started with the Vue Dashboard layout Component in Vue 2

This article provides a step-by-step guide for setting up a Vue 2 project using [Vue-CLI](#)[Link to the Video](#) and integrating the Syncfusion Vue Dashboard layout component.

To get start quickly with Vue Dashboard layout, you can check on this video:

Prerequisites

[System requirements for Syncfusion Vue UI components](#)

Dependencies

The following list of dependencies is required to use the DashboardLayout component in your application.

```
`javascript
|-- @syncfusion/ej2-vue-layouts
|-- @syncfusion/ej2-vue-base
|-- @syncfusion/ej2-base
|-- @syncfusion/ej2-layouts
`
```

### Setting up the Vue 2 project

To generate a Vue 2 project using Vue-CLI, use the [vue create](#) command. Follow these steps to install Vue CLI and create a new project:

```
`bash
npm install -g @vue/cli
vue create quickstart
cd quickstart
npm run serve
`
```

or

```
`bash
yarn global add @vue/cli
vue create quickstart
cd quickstart
```

```
yarn run serve
```

When creating a new project, choose the option `Default ([Vue 2] babel, eslint)` from the menu.

```
E:\samples>vue create quickstart

Vue CLI v5.0.8
? Please pick a preset:
  Default ([Vue 3] babel, eslint)
> Default ([Vue 2] babel, eslint)
  Manually select features
```

Once the `quickstart` project is set up with default settings, proceed to add Syncfusion components to the project.

### Add Syncfusion Vue packages

Syncfusion packages are available at [npmjs.com](https://www.npmjs.com). To use Vue components, install the required npm package.

This article uses the [Vue Layouts component](#) as an example. Install the `@syncfusion/ej2-vue-layouts` package by running the following command:

```
`bash
```

```
npm install @syncfusion/ej2-vue-layouts --save
```

or

```
`bash
```

```
yarn add @syncfusion/ej2-vue-layouts
```

### Import Syncfusion CSS styles

You can import themes for the Syncfusion Vue component in various ways, such as using CSS or SASS styles from npm packages, CDN, [CRG](#) and [Theme Studio](#). Refer to [themes topic](#) to know more about built-in themes and different ways to refer to themes in a Vue project.

In this article, the `Material` theme is applied using CSS styles, which are available in installed packages. The necessary Material CSS styles for the Dashboard layout component and its dependents were imported into the `<style>` section of `src/App.vue` file.

#### ~/SRC/APP.VUE

```
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-layouts/styles/material.css";
</style>
```

Note: If you want to refer the combined component styles, please make use of our [CRG](#) (Custom Resource Generator) in your application.

### Add Syncfusion Vue component

Follow the below steps to add the Vue Dashboard layout component:

1\ First, import and register the Dashboard layout component in the **script** section of the **src/App.vue** file.

#### ~/SRC/APP.VUE

```
<script>
// Import syncfusion dashboardlayout component from layouts package
import { DashboardLayoutComponent } from "@syncfusion/ej2-vue-layouts";
export default {
  components: {
    'ejs-dashboardlayout': DashboardLayoutComponent
  }
}
</script>
```

2\ In the **template** section, define the Dashboard layout component with [cellSpacing](#) property.

#### ~/SRC/APP.VUE

```
<template>
<div>
<div class="control-section">
<!-- DashboardLayout element declaration -->
<ejs-dashboardlayout id='defaultLayout' :cellSpacing="spacing" :columns="6">
<div id="one" class="e-panel" data-row="0" data-col="0" data-sizeX="1" data-sizeY="1">
<div class="e-panel-container">
<div class="text-align">
<div>0</div>
</div>
</div>
</div>
<div id="two" class="e-panel" data-row="1" data-col="0" data-sizeX="1" data-sizeY="2">
<div class="e-panel-container">
<div class="text-align">
<div>1</div>
</div>
</div>
</div>
<div id="three" class="e-panel" data-row="0" data-col="1" data-sizeX="2" data-sizeY="2">
<div class="e-panel-container">
<div class="text-align">
<div>2</div>
</div>
</div>
</div>
<div id="four" class="e-panel" data-row="2" data-col="1" data-sizeX="1" data-sizeY="1">
<div class="e-panel-container">
```

```
<div class="text-align">
<div>3</div>
</div>
</div>
</div>
<div id="five" class="e-panel" data-row="2" data-col="2" data-sizeX="2"
data-sizeY="1">
<div class="e-panel-container">
<div class="text-align">
<div>4</div>
</div>
</div>
</div>
<div id="six" class="e-panel" data-row="0" data-col="3" data-sizeX="1" data-
sizeY="1">
<div class="e-panel-container">
<div class="text-align">
<div>5</div>
</div>
</div>
</div>
<div id="seven" class="e-panel" data-row="1" data-col="3" data-sizeX="1"
data-sizeY="1">
<div class="e-panel-container">
<div class="text-align">
<div>6</div>
</div>
</div>
</div>
<div id="eight" class="e-panel" data-row="0" data-col="4" data-sizeX="1"
data-sizeY="3">
<div class="e-panel-container">
<div class="text-align">
<div>7</div>
</div>
</div>
</div>
</ejs-dashboardlayout>
<!-- end of dashboardlayout element -->
</div>
</div>
</template>
```

3\ Declare the value for the `cellSpacing` property in the `script` section.

~/SRC/APP.VUE

```
<script>
data: function() {
  return {
    count: 8,
    spacing: [10,10]
  };
}
</script>
```



Here is the summarized code for the above steps in the **src/App.vue** file:

**~/SRC/APP.VUE**

```
<template>
  <div>
    <div class="control-section">
      <!-- DashboardLayout element declaration -->
      <ejs-dashboardlayout id='defaultLayout' :cellSpacing="spacing"
:columns="6">
        <div id="one" class="e-panel" data-row="0" data-col="0"
data-sizeX="1" data-sizeY="1">
          <div class="e-panel-container">
            <div class="text-align">
              <div>0</div>
            </div>
          </div>
        </div>
        <div id="two" class="e-panel" data-row="1" data-col="0"
data-sizeX="1" data-sizeY="2">
          <div class="e-panel-container">
            <div class="text-align">
              <div>1</div>
            </div>
          </div>
        </div>
        <div id="three" class="e-panel" data-row="0" data-col="1"
data-sizeX="2" data-sizeY="2">
          <div class="e-panel-container">
            <div class="text-align">
              <div>2</div>
            </div>
          </div>
        </div>
        <div id="four" class="e-panel" data-row="2" data-col="1"
data-sizeX="1" data-sizeY="1">
          <div class="e-panel-container">
            <div class="text-align">
              <div>3</div>
            </div>
          </div>
        </div>
        <div id="five" class="e-panel" data-row="2" data-col="2"
data-sizeX="2" data-sizeY="1">
          <div class="e-panel-container">
            <div class="text-align">
              <div>4</div>
            </div>
          </div>
        </div>
        <div id="six" class="e-panel" data-row="0" data-col="3"
data-sizeX="1" data-sizeY="1">
          <div class="e-panel-container">
            <div class="text-align">
              <div>5</div>
            </div>
          </div>
        </div>
      </div>
    </div>
  </div>
</template>
```

```

        </div>
        <div id="seven" class="e-panel" data-row="1" data-col="3"
data-sizeX="1" data-sizeY="1">
            <div class="e-panel-container">
                <div class="text-align">
                    <div>6</div>
                </div>
            </div>
        </div>
        <div id="eight" class="e-panel" data-row="0" data-col="4"
data-sizeX="1" data-sizeY="3">
            <div class="e-panel-container">
                <div class="text-align">
                    <div>7</div>
                </div>
            </div>
        </div>
    </ejs-dashboardlayout>
    <!-- end of dashboardlayout element -->
</div>
</div>
</template>
<script>
// Import syncfusion dashboardlayout component from layouts package
import { DashboardLayoutComponent } from "@syncfusion/ej2-vue-layouts";
export default {
    components: {
        'ejs-dashboardlayout': DashboardLayoutComponent
    },
    data: function() {
        return {
            count: 8,
            spacing: [10,10]
        };
    }
}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-layouts/styles/material.css";
/* DashboardLayout element styles */
#defaultLayout .e-panel .e-panel-container {
    vertical-align: middle;
    font-weight: 600;
    font-size: 20px;
    text-align: center;
}
.text-align {
    line-height: 80px;
}
#defaultLayout .e-panel {
    transition:none !important;
}
</style>

```

### Run the project

To run the project, use the following command:

```
`bash
```

```
npm run serve
```

```
`
```

or

```
`bash
```

```
yarn run serve
```

```
`
```

```
{% previewsample "page.domainurl/code-snippet/dashboard-layout/getting-started-cs1" %}
```

### Setting the `panels` property using helper

You can render the DashboardLayout component by using the **panels** property through `<e-panels>`.

### ~/SRC/APP.VUE

```

<template>
  <div>
    <div class="control-section">
      <!-- DashboardLayout element declaration -->
      <ejs-dashboardlayout id="defaultLayout" :cellSpacing="spacing"
:columns="6">
        <e-panels>
          <e-panel :row="0" :col="0" :sizeX="1" :sizeY="1"
content="<div class='panel-content'>0</div>"></e-panel>
          <e-panel :row="0" :col="1" :sizeX="3" :sizeY="2"
content="<div class='panel-content'>1</div>"></e-panel>
          <e-panel :row="0" :col="4" :sizeX="1" :sizeY="3"
content="<div class='panel-content'>2</div>"></e-panel>
          <e-panel :row="1" :col="0" :sizeX="1" :sizeY="1"
content="<div class='panel-content'>3</div>"></e-panel>
          <e-panel :row="2" :col="0" :sizeX="2" :sizeY="1"
content="<div class='panel-content'>4</div>"></e-panel>
          <e-panel :row="2" :col="2" :sizeX="1" :sizeY="1"
content="<div class='panel-content'>5</div>"></e-panel>
          <e-panel :row="2" :col="3" :sizeX="1" :sizeY="1"
content="<div class='panel-content'>6</div>"></e-panel>
        </e-panels>
      </ejs-dashboardlayout>
      <!-- end of dashboardlayout element -->
    </div>
  </div>
</template>
<script>
// Import syncfusion dashboardlayout component from layouts package
import { DashboardLayoutComponent, PanelDirective, PanelsDirective } from
"@syncfusion/ej2-vue-layouts";
export default {
  components: {
    'ejs-dashboardlayout': DashboardLayoutComponent,
    'e-panel': PanelDirective,

```

```
    'e-panels': PanelsDirective
  },
  data: function() {
    return {
      spacing: [10,10]
    };
  }
}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-layouts/styles/material.css";
/* DashboardLayout element styles */
#defaultLayout .e-panel .e-panel-container {
  vertical-align: middle;
  font-weight: 600;
  font-size: 20px;
  text-align: center;
}
.panel-content {
  line-height: 80px;
}
#defaultLayout .e-panel {
  transition:none !important;
}
</style>
```

{% previewsample "page.domainurl/code-snippet/dashboard-layout/getting-started-panel-cs1" %}

You can refer to our [Vue Dashboard Layout](#) feature tour page for its groundbreaking feature representations. You can also explore our [Vue Dashboard Layout example](#) to know how to present and manipulate data.

### Getting Started with the Vue Dashboard Layout Component in Vue 3

This article provides a step-by-step guide for setting up a [Vite](#) project with a JavaScript environment and integrating the Syncfusion Vue Dashboard Layout component using the [Composition API](#) / [Options API](#).

The **Composition API** is a new feature introduced in Vue.js 3 that provides an alternative way to organize and reuse component logic. It allows developers to write components as functions that use smaller, reusable functions called composition functions to manage their properties and behavior.

The **Options API** is the traditional way of writing Vue.js components, where the component logic is organized into a series of options that define the component's properties and behavior. These options include data, methods, computed properties, watchers, lifecycle hooks, and more.

#### Prerequisites

##### [System requirements for Syncfusion Vue UI components](#)

##### Set up the Vite project

A recommended approach for beginning with Vue is to scaffold a project using [Vite](#). To create a new Vite project, use one of the commands that are specific to either NPM or Yarn.

```
`bash
```

```
npm create vite@latest
```

`

or

```
`bash
```

```
yarn create vite
```

`

Using one of the above commands will lead you to set up additional configurations for the project as below:

1. Define the project name: We can specify the name of the project directly. Let's specify the name of the project as **my-project** for this article.

```
`bash
```

```
? Project name: » my-project
```

`

2. Select **Vue** as the framework. It will create a Vue 3 project.

```
`bash
```

```
? Select a framework: » - Use arrow-keys. Return to submit.
```

Vanilla

Vue

React

Preact

Lit

Svelte

Others

`

3. Choose **JavaScript** as the framework variant to build this Vite project using JavaScript and Vue.

```
`bash
```

```
? Select a variant: » - Use arrow-keys. Return to submit.
```

JavaScript

TypeScript

Customize with create-vue ↗

Nuxt ↗

`

4. Upon completing the aforementioned steps to create the **my-project**, run the following command to install its dependencies:

```
`bash
```

```
cd my-project
```

```
npm install
```

```
,
```

```
or
```

```
`bash
```

```
cd my-project
```

```
yarn install
```

```
,
```

Now that `my-project` is ready to run with default settings, let's add Syncfusion components to the project.

### Add Syncfusion Vue packages

Syncfusion Vue component packages are available at [npmjs.com](https://www.npmjs.com). To use Syncfusion Vue components in the project, install the corresponding npm package.

This article uses the [Vue Dashboard Layout component](#) as an example. To use the Vue Dashboard Layout component in the project, the `@syncfusion/ej2-vue-layouts` package needs to be installed using the following command:

```
`bash
```

```
npm install @syncfusion/ej2-vue-layouts --save
```

```
,
```

```
or
```

```
`bash
```

```
yarn add @syncfusion/ej2-vue-layouts
```

```
,
```

### Import Syncfusion CSS styles

You can import themes for the Syncfusion Vue component in various ways, such as using CSS or SASS styles from npm packages, CDN, [CRG](#) and [Theme Studio](#). Refer to [themes topic](#) to know more about built-in themes and different ways to refer to themes in a Vue project.

In this article, `Material` theme is applied using CSS styles, which are available in installed packages. The necessary `Material` CSS styles for the Dashboard Layout component and its dependents were imported into the `<style>` section of `src/App.vue` file.

### ~/SRC/APP.VUE

```
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-layouts/styles/material.css";
</style>
```

### Add Syncfusion Vue component

Follow the below steps to add the Vue Dashboard Layout component using **Composition API** or **Options API**:

1.First, import and register the Grid component and its child directives in the **script** section of the **src/App.vue** file. If you are using the **Composition API**, you should add the **setup** attribute to the **script** tag to indicate that Vue will be using the **Composition API**.Find the list of child directives and the tag names that can be used in the Dashboard Layout component in the following table.

Directive Name	Tag Name
-----	-----
<b>PanelsDirective</b>	<b>e-panels</b>
<b>PanelDirective</b>	<b>e-panel</b>

#### **COMPOSITION API (~SRC/APP.VUE)**

```
<script setup>
import { DashboardLayoutComponent as EjsDashboardlayout, PanelDirective as
EPanel, PanelsDirective as EPanels } from "@syncfusion/ej2-vue-layouts";
</script>
```

#### **OPTIONS API (~SRC/APP.VUE)**

```
<script>
import { DashboardLayoutComponent, PanelDirective, PanelsDirective } from
"@syncfusion/ej2-vue-layouts";
//Component registration
export default {
name: "App",
components: {
"ejs-dashboardlayout": DashboardLayoutComponent,
"e-panels":PanelsDirective,
"e-panel":PanelDirective,
}
}
</script>
```

2.Add the component definition in template section.

#### **~/SRC/APP.VUE**

```
<template>
<ejs-dashboardlayout id="defaultLayout" :columns="6"
:cellSpacing='cellSpacing' :cellAspectRatio='aspectRatio'>
<e-panels>
<e-panel :row="0" :col="0" :sizeX="1" :sizeY="1" content="<div class='panel-
content'>0</div>"></e-panel>
<e-panel :row="0" :col="1" :sizeX="3" :sizeY="2" content="<div class='panel-
content'>1</div>"></e-panel>
<e-panel :row="0" :col="4" :sizeX="1" :sizeY="3" content="<div class='panel-
content'>2</div>"></e-panel>
<e-panel :row="1" :col="0" :sizeX="1" :sizeY="1" content="<div class='panel-
content'>3</div>"></e-panel>
```

```

<e-panel :row="2" :col="0" :sizeX="2" :sizeY="1" content="<div class='panel-content'>4</div>"></e-panel>
<e-panel :row="2" :col="2" :sizeX="1" :sizeY="1" content="<div class='panel-content'>5</div>"></e-panel>
<e-panel :row="2" :col="3" :sizeX="1" :sizeY="1" content="<div class='panel-content'>6</div>"></e-panel>
</e-panels>
</ejs-dashboardlayout>
</template>

```

Here is the summarized code for the above steps in the **src/App.vue** file:

#### **COMPOSITION API (~ /SRC /APP.VUE)**

```

<template>
<ejs-dashboardlayout id="defaultLayout" :columns="6"
:cellSpacing='cellSpacing' :cellAspectRatio='aspectRatio'>
<e-panels>
<e-panel :row="0" :col="0" :sizeX="1" :sizeY="1" content="<div class='panel-content'>0</div>"></e-panel>
<e-panel :row="0" :col="1" :sizeX="3" :sizeY="2" content="<div class='panel-content'>1</div>"></e-panel>
<e-panel :row="0" :col="4" :sizeX="1" :sizeY="3" content="<div class='panel-content'>2</div>"></e-panel>
<e-panel :row="1" :col="0" :sizeX="1" :sizeY="1" content="<div class='panel-content'>3</div>"></e-panel>
<e-panel :row="2" :col="0" :sizeX="2" :sizeY="1" content="<div class='panel-content'>4</div>"></e-panel>
<e-panel :row="2" :col="2" :sizeX="1" :sizeY="1" content="<div class='panel-content'>5</div>"></e-panel>
<e-panel :row="2" :col="3" :sizeX="1" :sizeY="1" content="<div class='panel-content'>6</div>"></e-panel>
</e-panels>
</ejs-dashboardlayout>
</template>
<script setup>
import { DashboardLayoutComponent as EjsDashboardlayout, PanelDirective as EPanel, PanelsDirective as EPanels } from "@syncfusion/ej2-vue-layouts";
const cellSpacing = [10,10];
const aspectRatio = 100/85;
</script>
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-layouts/styles/material.css";
/* DashboardLayout element styles */
#defaultLayout .e-panel .e-panel-container {
vertical-align: middle;
font-weight: 600;
font-size: 20px;
text-align: center;
}
.panel-content {
line-height: 80px;
}
#defaultLayout .e-panel {
transition:none !important;

```



```
}
</style>
```

### OPTIONS API (~SRC/APP.VUE)

```
<template>
<ejs-dashboardlayout id="defaultLayout" :columns="6"
:cellSpacing='cellSpacing' :cellAspectRatio='aspectRatio'>
<e-panels>
<e-panel :row="0" :col="0" :sizeX="1" :sizeY="1" content="<div class='panel-
content'>0</div>"></e-panel>
<e-panel :row="0" :col="1" :sizeX="3" :sizeY="2" content="<div class='panel-
content'>1</div>"></e-panel>
<e-panel :row="0" :col="4" :sizeX="1" :sizeY="3" content="<div class='panel-
content'>2</div>"></e-panel>
<e-panel :row="1" :col="0" :sizeX="1" :sizeY="1" content="<div class='panel-
content'>3</div>"></e-panel>
<e-panel :row="2" :col="0" :sizeX="2" :sizeY="1" content="<div class='panel-
content'>4</div>"></e-panel>
<e-panel :row="2" :col="2" :sizeX="1" :sizeY="1" content="<div class='panel-
content'>5</div>"></e-panel>
<e-panel :row="2" :col="3" :sizeX="1" :sizeY="1" content="<div class='panel-
content'>6</div>"></e-panel>
</e-panels>
</ejs-dashboardlayout>
</template>
<script>
import { DashboardLayoutComponent, PanelDirective, PanelsDirective } from
"@syncfusion/ej2-vue-layouts";
//Component registration
export default {
name: "App",
components: {
"ejs-dashboardlayout": DashboardLayoutComponent,
"e-panels": PanelsDirective,
"e-panel": PanelDirective,
},
data() {
return {
cellSpacing: [10,10],
aspectRatio: 100/85
};
}
}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-layouts/styles/material.css";
/* DashboardLayout element styles */
#defaultLayout .e-panel .e-panel-container {
vertical-align: middle;
font-weight: 600;
font-size: 20px;
text-align: center;
}
.panel-content {
```

```
line-height: 80px;  
}  
#defaultLayout .e-panel {  
  transition:none !important;  
}  
</style>
```

### Run the project

To run the project, use the following command:

```
`bash
```

```
npm run dev
```

```
`
```

or

```
`bash
```

```
yarn run dev
```

```
`
```

The output will appear as follows:



For migrating from Vue 2 to Vue 3, refer to the [migration](#) documentation.

See also

- [Getting Started with Vue UI Components using Composition API and TypeScript](#)
- [Getting Started with Vue UI Components using Options API and TypeScript](#)

### Setting size of cells in Vue Dashboard layout component

The entire layout dimensions are assigned based on the height and width of the parent element. Hence, a responsive or static layout can be created by assigning a percentage or static dimension values to the

parent element. The layout adapts to mobile resolutions by transforming the entire layout into a stacked orientation, so that, the panels will be displayed in a vertical column.

The **Dashboard Layout** is a grid structured component which can be split into subsections of equal size known as cells. The total number of cells in each row is defined by using the [columns](#) property of the component. The width of each cell will be auto calculated based on the total number of cells placed in a row and the height of a cell will be same as that of its width. However, the height of these cells can also be configured to any desired size using the [cellAspectRatio](#) property (cellwidth/cellheight ratio) which defines the cell width to height ratio.

The number of rows within the layout has no limits and can have any number of rows based on the panels count and position. Panels which acts as data containers will be placed or positioned over these cells.

### Modifying cell size

In a dashboard, the data to be held by the panel in a cell may be of different size, hence different cell dimensions may be required in different scenarios. In this case, the size of these grid cells can be modified to the required size using the [columns](#) and [cellAspectRatio](#) properties.

The following sample demonstrates how to modify a cell size using the [columns](#) and [cellAspectRatio](#) properties. In the following sample the width of the parent element is divided into 5 equal cells based on the columns property value resulting the width of each cell as 100 px. The height of these cells will be 50 px based on the cellAspectRatio value 100/50 (i.e. for every 100 px of width, 50 px will be the height of the cell).

### APP.VUE

```
<template>
  <div class="control-section">
    <!-- DashboardLayout element declaration -->
    <ejs-dashboardlayout id='dashboard_layout' :columns="5"
:cellSpacing='cellSpacing' :cellAspectRatio='cellAspectRatio'>
      <e-panels>
        <e-panel :sizeX="1" :sizeY="1" :row="0" :col="0"
content="<div class='content'>0</div>"></e-panel>
        <e-panel :sizeX="3" :sizeY="2" :row="0" :col="1"
content="<div class='content'>1</div>"></e-panel>
        <e-panel :sizeX="1" :sizeY="3" :row="0" :col="4"
content="<div class='content'>2</div>"></e-panel>
        <e-panel :sizeX="1" :sizeY="1" :row="1" :col="0"
content="<div class='content'>3</div>"></e-panel>
        <e-panel :sizeX="2" :sizeY="1" :row="2" :col="0"
content="<div class='content'>4</div>"></e-panel>
        <e-panel :sizeX="1" :sizeY="1" :row="2" :col="2"
content="<div class='content'>5</div>"></e-panel>
        <e-panel :sizeX="1" :sizeY="1" :row="2" :col="3"
content="<div class='content'>6</div>"></e-panel>
      </e-panels>
    </ejs-dashboardlayout>
    <!-- end of dashboardlayout element -->
  </div>
</template>
<script>
import Vue from "vue";
// Import syncfusion dashboardlayout component from layouts package
```

```

import { DashboardLayoutPlugin } from "@syncfusion/ej2-vue-layouts";
Vue.use(DashboardLayoutPlugin);
export default {
  data: function() {
    return {
      cellSpacing: [10, 10],
      cellAspectRatio: 100/50,
    };
  }
}
</script>
<style>
@import
"https://ej2.syncfusion.com/vue/documentation/node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-layouts/styles/material.css";
/* DashboardLayout element styles */
#dashboard_layout .e-panel .e-panel-content {
  vertical-align: middle;
  font-weight: 600;
  font-size: 20px;
  text-align: center;
  line-height: 60px;
}
#dashboard_layout .e-panel {
  transition:none !important;
}
</style>

```

{% previewsample "page.domainurl/code-snippet/dashboard-layout/modifying-cell-size-cs1" %}

### Setting cell spacing

The spacing between each panel in a row and column can be defined using the [cellSpacing](#) property. Adding spacing between the panels will make the layout effective and provides a clear data representation.

The following sample demonstrates the usage of the [cellSpacing](#) property which helps in a neat and clear representation of a data.

### APP.VUE

```

<template>
  <div class="control-section">
    <!-- DashboardLayout element declaration -->
    <ejs-dashboardlayout id='dashboard_layout' :columns="5"
:cellSpacing='cellSpacing' >
      <e-panels>
        <e-panel :sizeX="1" :sizeY="1" :row="0" :col="0"
content="<div class='content'>0</div>"></e-panel>
        <e-panel :sizeX="3" :sizeY="2" :row="0" :col="1"
content="<div class='content'>1</div>"></e-panel>
        <e-panel :sizeX="1" :sizeY="3" :row="0" :col="4"
content="<div class='content'>2</div>"></e-panel>
        <e-panel :sizeX="1" :sizeY="1" :row="1" :col="0"
content="<div class='content'>3</div>"></e-panel>

```

```

        <e-panel :sizeX="2" :sizeY="1" :row="2" :col="0"
content="<div class='content'>4</div>"></e-panel>
        <e-panel :sizeX="1" :sizeY="1" :row="2" :col="2"
content="<div class='content'>5</div>"></e-panel>
        <e-panel :sizeX="1" :sizeY="1" :row="2" :col="3"
content="<div class='content'>6</div>"></e-panel>
    </e-panels>
</ejs-dashboardlayout>
<!-- end of dashboardlayout element -->
</div>
</template>
<script>
import Vue from "vue";
// Import syncfusion dashboardlayout component from layouts package
import { DashboardLayoutPlugin } from "@syncfusion/ej2-vue-layouts";
Vue.use(DashboardLayoutPlugin);
export default {
    data: function() {
        return {
            cellSpacing: [20, 20]
        };
    }
}
</script>
<style>
@import
"https://ej2.syncfusion.com/vue/documentation/node_modules/@syncfusion/ej2-
base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-layouts/styles/material.css";
/* DashboardLayout element styles */
#dashboard_layout .e-panel .e-panel-content {
    vertical-align: middle;
    font-weight: 600;
    font-size: 20px;
    text-align: center;
    line-height: 100px;
}
#dashboard_layout .e-panel {
    transition:none !important;
}
</style>

```

{% previewsample "page.domainurl/code-snippet/dashboard-layout/setting-cell-spacing-cs3" %}

### Graphical representation of layout

These cells combinedly forms a grid-structured layout which will be hidden initially. This grid structured layout can be made visible by enabling the [showGridLines](#) property, which clearly pictures the cells split-up within the layout. These gridlines will be helpful in panels sizing and placement within the layout during initial designing of a dashboard.

In the following sample, the grid lines indicate the cells split-up of the layout and the data containers placed over these cells are known as panels.

### APP.VUE

```
<template>
```

```

<div class="control-section">
  <!-- DashboardLayout element declaration -->
  <ejs-dashboardlayout id='dashboard_layout'
:cellSpacing='cellSpacing' :showGridLines='showGridLines' :columns="5">
    <e-panels>
      <e-panel :sizeX="3" :sizeY="2" :row="0" :col="1"
content="<div class='content'>1</div>"></e-panel>
      <e-panel :sizeX="1" :sizeY="3" :row="0" :col="4"
content="<div class='content'>2</div>"></e-panel>
      <e-panel :sizeX="1" :sizeY="1" :row="2" :col="2"
content="<div class='content'>3</div>"></e-panel>
      <e-panel :sizeX="1" :sizeY="1" :row="2" :col="3"
content="<div class='content'>4</div>"></e-panel>
    </e-panels>
  </ejs-dashboardlayout>
  <!-- end of dashboardlayout element -->
</div>
</template>
<script>
import Vue from "vue";
// Import syncfusion dashboardlayout component from layouts package
import { DashboardLayoutPlugin } from "@syncfusion/ej2-vue-layouts";
Vue.use(DashboardLayoutPlugin);
export default {
  data: function() {
    return {
      cellSpacing: [10, 10],
      showGridLines: true,
    };
  }
}
</script>
<style>
@import
"https://ej2.syncfusion.com/vue/documentation/node_modules/@syncfusion/ej2-
base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-layouts/styles/material.css";
/* DashboardLayout element styles */
#dashboard_layout .e-panel .e-panel-container .content {
  vertical-align: middle;
  font-weight: 600;
  font-size: 20px;
  text-align: center;
  line-height: 100px;
}
#dashboard_layout .e-panel {
  transition:none !important;
}
</style>

```

{% previewsample "page.domainurl/code-snippet/dashboard-layout/graphical-layout-cs1" %}

### Rendering component in right-to-left direction

It is possible to render the Dashboard Layout in right-to-left direction by setting the [enableRtl](#) API to true.

The following sample demonstrates Dashboard Layout in right-to-left direction.

### APP.VUE

```
<template>
  <div class="col-lg-8 control-section">
    <!-- DashboardLayout element declaration -->
    <ejs-dashboardlayout id="defaultLayout" :cellSpacing="spacing"
:columns="5" :enableRtl="enableRtl">
      <e-panels>
        <e-panel id="panel0" :row="0" :col="0" :sizeX="1" :sizeY="1"
header="<div>Panel 0</div>" content='<div class="content">Panel
Content<div>'></e-panel>
        <e-panel id="panel1" :row="0" :col="1" :sizeX="3" :sizeY="2"
header="<div>Panel 1</div>" content='<div class="content">Panel
Content<div>'></e-panel>
        <e-panel id="panel2" :row="0" :col="4" :sizeX="1" :sizeY="3"
header="<div>Panel 2</div>" content='<div class="content">Panel
Content<div>'></e-panel>
        <e-panel id="panel3" :row="1" :col="0" :sizeX="1" :sizeY="1"
header="<div>Panel 3</div>" content='<div class="content">Panel
Content<div>'></e-panel>
        <e-panel id="panel4" :row="2" :col="0" :sizeX="2" :sizeY="1"
header="<div>Panel 4</div>" content='<div class="content">Panel
Content<div>'></e-panel>
        <e-panel id="panel5" :row="2" :col="2" :sizeX="1" :sizeY="1"
header="<div>Panel 5</div>" content='<div class="content">Panel
Content<div>'></e-panel>
        <e-panel id="panel6" :row="2" :col="3" :sizeX="1" :sizeY="1"
header="<div>Panel 6</div>" content='<div class="content">Panel
Content<div>'></e-panel>
      </e-panels>
    </ejs-dashboardlayout>
    <!-- end of dashboardlayout element -->
  </div>
</template>
<script>
import Vue from "vue";
// Import syncfusion dashboardlayout component from layouts package
import { DashboardLayoutPlugin } from "@syncfusion/ej2-vue-layouts";
Vue.use(DashboardLayoutPlugin);
export default {
  data: function() {
    return {
      spacing: [10,10],
      enableRtl: true
    };
  }
}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-layouts/styles/material.css";
/* DashboardLayout element styles */
#defaultLayout .e-panel .e-panel-container .e-panel-header {
  vertical-align: middle;
  font-weight: 600;
}
```

```
font-size: 20px;
padding:10px
}
.e-panel-content {
padding:20px;
}
#defaultLayout .e-panel {
transition:none !important;
}
</style>
```

{% previewsample "page.domainurl/code-snippet/dashboard-layout/rtl-cs1" %}

You can refer to our [Vue Dashboard Layout](#) feature tour page for its groundbreaking feature representations. You can also explore our [Vue Dashboard Layout example](#) to know how to present and manipulate data.

## Panels

### Position sizing of panels in Vue Dashboard layout component

Panels are the basic building blocks of the dashboard layout component. They act as a container for the data to be visualized or presented. These panels can be positioned or resized for effective presentation of the data.

The following table represents all the available panel properties and the corresponding functionalities.

PanelObject	Description
-------------	-------------

---	---
-----	-----

id	Specifies the ID value of the panel.
----	--------------------------------------

row	Specifies the row value in which the panel to be placed.
-----	--

col	Specifies the column value in which the panel to be placed.
-----	---

sizeX	Specifies the width of the panel in cells count.
-------	--

sizeY	Specifies the height of the panel in cells count.
-------	---

minSizeX	Specifies the minimum width of the panel in cells count.
----------	--

minSizeY	Specifies the minimum height of the panel in cells count.
----------	---

maxSizeX	Specifies the maximum width of the panel in cells count.
----------	--

maxSizeY	Specifies the maximum height of the panel in cells count.
----------	---

header	Specifies the header template of the panel.
--------	---

content	Specifies the content template of the panel.
---------	--

cssClass	Specifies the CSS class name that can be appended with each panel element.
----------	--

### Positioning of panels

The panels within the layout can be easily positioned or ordered using the `row` and `col` properties of the panels. Positioning of panels will be beneficial to represent the data in any desired order.



The following sample demonstrates the positioning of panels within the dashboard layout using the row, and, column properties of the panels.

### APP.VUE

```
<template>
  <div class="control-section">
    <!-- DashboardLayout element declaration -->
    <ejs-dashboardlayout id='dashboard_default' :cellSpacing='cellSpacing'
:columns="5">
      <e-panels>
        <e-panel :row="0" :col="0" content="<div
class='content'>1</div>"></e-panel>
        <e-panel :row="0" :col="1" content="<div
class='content'>2</div>"></e-panel>
        <e-panel :row="0" :col="2" content="<div
class='content'>3</div>"></e-panel>
        <e-panel :row="1" :col="0" content="<div
class='content'>4</div>"></e-panel>
        <e-panel :row="1" :col="1" content="<div
class='content'>5</div>"></e-panel>
        <e-panel :row="1" :col="2" content="<div
class='content'>6</div>"></e-panel>
      </e-panels>
    </ejs-dashboardlayout>
    <!-- end of dashboardlayout element -->
  </div>
</template>
<script>
import Vue from "vue";
// Import syncfusion dashboardlayout component from layouts package
import { DashboardLayoutPlugin } from "@syncfusion/ej2-vue-layouts";
Vue.use(DashboardLayoutPlugin);
export default {
  data: function() {
    return {
      cellSpacing: [20, 20]
    };
  }
}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-layouts/styles/material.css";
/* DashboardLayout element styles */
#dashboard_default .e-panel .e-panel-container .content {
  vertical-align: middle;
  font-weight: 600;
  font-size: 20px;
  text-align: center;
  line-height: 100px;
}
#dashboard_default .e-panel {
  transition:none !important;
}
</style>
```

```
{% previewsample "page.domainurl/code-snippet/dashboard-layout/position-of-panels-cs1" %}
```

### Sizing of panels

A panel's size can be varied easily by defining the `sizeX` and `sizeY` properties. The `sizeX` property defines the width and the `sizeY` property defines height of a panel in cells count. These properties will be helpful in designing a dashboard, where the content of each panel may vary in size.

The following sample demonstrates the sizing of panels within the dashboard layout using the `sizeX` and `sizeY` properties of the panels.

### APP.VUE

```
<template>
  <div class="control-section">
    <!-- DashboardLayout element declaration -->
    <ejs-dashboardlayout id='dashboard_default' :columns="5">
      <e-panels>
        <e-panel :sizeX="1" :sizeY="1" :row="0" :col="0" content="<div
class='content'>0</div>"></e-panel>
        <e-panel :sizeX="3" :sizeY="2" :row="0" :col="1" content="<div
class='content'>1</div>"></e-panel>
        <e-panel :sizeX="1" :sizeY="3" :row="0" :col="4" content="<div
class='content'>2</div>"></e-panel>
        <e-panel :sizeX="1" :sizeY="1" :row="1" :col="0" content="<div
class='content'>3</div>"></e-panel>
        <e-panel :sizeX="2" :sizeY="1" :row="2" :col="0" content="<div
class='content'>4</div>"></e-panel>
        <e-panel :sizeX="1" :sizeY="1" :row="2" :col="2" content="<div
class='content'>5</div>"></e-panel>
        <e-panel :sizeX="1" :sizeY="1" :row="2" :col="3" content="<div
class='content'>6</div>"></e-panel>
      </e-panels>
    </ejs-dashboardlayout>
    <!-- end of dashboardlayout element -->
  </div>
</template>
<script>
import Vue from "vue";
// Import syncfusion dashboardlayout component from layouts package
import { DashboardLayoutPlugin } from "@syncfusion/ej2-vue-layouts";
Vue.use(DashboardLayoutPlugin);
export default {
  data: function() {
    return {
    };
  }
}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-layouts/styles/material.css";
/* DashboardLayout element styles */
#dashboard_default .e-panel .e-panel-container .content {
  vertical-align: middle;
  font-weight: 600;
  font-size: 20px;
  text-align: center;
}
```

```

    line-height: 100px;
  }
  #dashboard_default .e-panel {
    transition:none !important;
  }
</style>

```

{% previewsample "page.domainurl/code-snippet/dashboard-layout/setting-cell-spacing-cs2" %}

You can refer to our [Vue Dashboard Layout](#) feature tour page for its groundbreaking feature representations. You can also explore our [Vue Dashboard Layout example](#) to know how to present and manipulate data.

### Setting header of panels in Vue Dashboard layout component

The dashboard layout component is mostly used to represent the data used for monitoring or managing a process. These data or any HTML template can be placed as the content of a panel using the `content` property. Also, word or phrase that summarize the panel's content can be added as the header on the top of each panel using the `header` property of the panel.

The following sample demonstrates how to add content for each panel using the header and content properties of the panels.

### APP.VUE

```

<template>
  <div class="col-lg-8 control-section">
    <!-- DashboardLayout element declaration -->
    <ejs-dashboardlayout id="defaultLayout" :cellSpacing="spacing"
:columns="7">
      <e-panels>
        <e-panel id="panel0" :row="0" :col="0" :sizeX="1" :sizeY="1"
header="<div>Panel 0</div>" content='<div class="content">Panel
Content<div>'></e-panel>
        <e-panel id="panel1" :row="0" :col="1" :sizeX="3" :sizeY="2"
header="<div>Panel 1</div>" content='<div class="content">Panel
Content<div>'></e-panel>
        <e-panel id="panel2" :row="0" :col="4" :sizeX="1" :sizeY="3"
header="<div>Panel 2</div>" content='<div class="content">Panel
Content<div>'></e-panel>
        <e-panel id="panel3" :row="1" :col="0" :sizeX="1" :sizeY="1"
header="<div>Panel 3</div>" content='<div class="content">Panel
Content<div>'></e-panel>
        <e-panel id="panel4" :row="2" :col="0" :sizeX="2" :sizeY="1"
header="<div>Panel 4</div>" content='<div class="content">Panel
Content<div>'></e-panel>
        <e-panel id="panel5" :row="2" :col="2" :sizeX="1" :sizeY="1"
header="<div>Panel 5</div>" content='<div class="content">Panel
Content<div>'></e-panel>
        <e-panel id="panel6" :row="2" :col="3" :sizeX="1" :sizeY="1"
header="<div>Panel 6</div>" content='<div class="content">Panel
Content<div>'></e-panel>
      </e-panels>
    </ejs-dashboardlayout>
    <!-- end of dashboardlayout element -->
  </div>
</template>

```

```

<script>
import Vue from "vue";
// Import syncfusion dashboardlayout component from layouts package
import { DashboardLayoutPlugin } from "@syncfusion/ej2-vue-layouts";
Vue.use(DashboardLayoutPlugin);
export default {
  data: function() {
    return {
      spacing: [10,10]
    };
  }
}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-layouts/styles/material.css";
/* DashboardLayout element styles */
#defaultLayout .e-panel .e-panel-container .e-panel-header {
  vertical-align: middle;
  font-weight: 600;
  font-size: 20px;
  text-align:center;
  padding:10px
}
.e-panel-content {
  text-align:center;
  padding:20px;
  line-height:40px;
}
#defaultLayout .e-panel {
  transition:none !important;
}
</style>

```

{% previewsample "page.domainurl/code-snippet/dashboard-layout/header-cs1" %}

### Placing components as content of panels

In a dashboard, components like the chart, grids, maps, gauge, and more etc. can be used to present a complex data. Such components can be placed as the panel content by assigning the corresponding component element as the **content** of the panel.

The following sample demonstrates how to add EJ2 Chart components as the **content** for each panel in the dashboard layout component.

### APP.VUE

```

<template>
  <div className="control-section" id="control_dash">
    <div className="content-wrapper">
      <!-- DashboardLayout element declaration -->
      <ejs-dashboardlayout ref="DashbordInstance" :columns="6"
id='edit_dashboard' :allowResizing="false" :allowDragging="true" >
        <e-panels>
          <e-panel :row="0" :col="0" :sizeX="3" :sizeY="2"
header="<div>Product usage ratio</div>" :content="pie"></e-panel>

```

```

        <e-panel :row="0" :col="3" :sizeX="3" :sizeY="2"
header="<div>Mobile browsers usage</div>" :content="pie1"></e-panel>
        <e-panel :row="1" :col="0" :sizeX="3" :sizeY="2"
header="<div>Spline Chart</div>" :content="spline"></e-panel>
    </e-panels>
</ejs-dashboardlayout>
<!-- end of dashboardlayout element -->
</div>
</div>
</template>
<script>
import Vue from "vue";
// Import syncfusion dashboardlayout component from layouts package
import { DashboardLayoutPlugin } from "@syncfusion/ej2-vue-layouts";
// Import syncfusion chart component from charts package
import { AccumulationChartPlugin, PieSeries, AccumulationDataLabel,
AccumulationTooltip, ChartPlugin, SplineAreaSeries, Legend, DateTime } from
"@syncfusion/ej2-vue-charts";
Vue.use(ChartPlugin);
Vue.use(AccumulationChartPlugin);
Vue.use(DashboardLayoutPlugin);
var splineTemplate = Vue.component("contentTemp1", {
    template: `
        <div id="container" style='display:block;height:100%, width:100%;'>
            <!-- Chart element declaration -->
            <ejs-chart class="chart-content" ref="splineInstance"
style='display:block;height:100%, width:100%;':primaryXAxis='primaryXAxis'
:primaryYAxis='primaryYAxis'
            :chartArea='chartArea' :height='height' :width='width'
:header='header'>
                <e-series-collection>
                    <e-series :dataSource='seriesData' type='SplineArea' xName='x'
yName='y' name='Jan' width=2 opacity=0.5 :fill="fill10"></e-series>
                    <e-series :dataSource='seriesData1' type='SplineArea' xName='x'
yName='y' name='Feb' width=2 opacity=0.5 :fill="fill11"></e-series>
                </e-series-collection>
            </ejs-chart>
            <!-- end of chart element -->
        </div>`,
    data: function() {
        return {
            seriesData: [
                { x: new Date(2002, 0, 1), y: 2.2 }, { x: new Date(2003, 0, 1), y:
3.4 },
                { x: new Date(2004, 0, 1), y: 2.8 }, { x: new Date(2005, 0, 1), y:
1.6 },
                { x: new Date(2006, 0, 1), y: 2.3 }, { x: new Date(2007, 0, 1), y:
2.5 },
                { x: new Date(2008, 0, 1), y: 2.9 }, { x: new Date(2009, 0, 1), y:
3.8 },
                { x: new Date(2010, 0, 1), y: 1.4 }, { x: new Date(2011, 0, 1), y:
3.1 }
            ],
            seriesData1: [
                { x: new Date(2002, 0, 1), y: 2 }, { x: new Date(2003, 0, 1), y: 1.7
},

```

```

2.1 },
    { x: new Date(2004, 0, 1), y: 1.8 }, { x: new Date(2005, 0, 1), y:
1.6 },
    { x: new Date(2006, 0, 1), y: 2.3 }, { x: new Date(2007, 0, 1), y:
2.7 },
    { x: new Date(2008, 0, 1), y: 1.5 }, { x: new Date(2009, 0, 1), y:
2.2 },
    { x: new Date(2010, 0, 1), y: 1.5 }, { x: new Date(2011, 0, 1), y:
    },
    //Initializing Primary X Axis
    primaryXAxis: {
        valueType: 'DateTime',
        labelFormat: 'Y',
        majorGridLines: { width: 0 },
        intervalType: 'Years',
        edgeLabelPlacement: 'Shift'
    },
    //Initializing Primary Y Axis
    primaryYAxis: {
        labelFormat: '{value}%',
        lineStyle: { width: 0 },
        majorTickLines: { width: 0 },
        minorTickLines: { width: 0 }
    },
    chartArea: {
        border: {
            width: 0
        }
    },
    border: {
        color: 'transparent'
    },
    width: "100%",
    fill1: 'rgb(0, 189, 174)',
    fill0: 'rgb(239, 183, 202)',
    height: "99%"
    };
    },
    provide: {
        chart: [SplineAreaSeries, Legend, DateTime]
    },
    mounted() {
        this.$refs.splineInstance.height = "100%";
        this.$refs.splineInstance.width = "100%";
    }
    });
    var pietemplate = Vue.component("contentTemp2", {
        template: `
        <div id="app" style='display:block;height:100%; width:100%;'>
            <ejs-accumulationchart class="chart-content"
            ref="accumulationInstance" style='display:block;height:100%; width:100%;'
            :legendSettings="legendSettings" :tooltip="tooltip">
                <e-accumulation-series-collection>
                    <e-accumulation-series :dataSource='seriesData' xName='x'
                    yName='y' innerRadius="40%" :dataLabel="dataLabel"> </e-accumulation-series>
                </e-accumulation-series-collection>
            </ejs-accumulationchart>
        `
    });

```

```

</div>`,
data() {
  return {
    seriesData: [
      { x: 'TypeScript', y: 13, text: 'TS 13%' },
      { x: 'React', y: 12.5, text: 'React 12.5%' },
      { x: 'MVC', y: 12, text: 'MVC 12%' },
      { x: 'Core', y: 12.5, text: 'Core 12.5%' },
      { x: 'Vue', y: 10, text: 'Vue 10%' },
      { x: 'Angular', y: 40, text: 'Angular 40%' }
    ],
    legendSettings: { visible: false },
    dataLabel: { visible: true, position: 'Inside', name: 'value' },
    tooltip: {
      enable: true, header: '<b>${point.x}</b>', format: 'Composition:
<b>${point.y}</b>'
    },
  };
},
provide: {
  accumulationchart: [PieSeries, AccumulationDataLabel,
  AccumulationTooltip]
},
mounted() {
  this.$refs.accumulationInstance.height = "100%";
  this.$refs.accumulationInstance.width = "100%";
}
});
var pietemplatel = Vue.component("contentTemp3", {
  template: `
    <div id="app1" style='display:block;height:100%; width:100%;'>
      <ejs-accumulationchart class="chart-content"
      ref="accumulationInstance" style='display:block;height:100%; width:100%;'
      :legendSettings="legendSettings" :tooltip="tooltip">
        <e-accumulation-series-collection>
          <e-accumulation-series :dataSource='seriesData' xName='x'
          yName='y' innerRadius="40%" :dataLabel="dataLabel"> </e-accumulation-series>
        </e-accumulation-series-collection>
      </ejs-accumulationchart>
    </div>`,
  data() {
    return {
      seriesData: [
        { 'x': 'Chrome', y: 37, text: '37%' },
        { 'x': 'UC Browser', y: 17, text: '17%' },
        { 'x': 'iPhone', y: 19, text: '19%' },
        { 'x': 'Others', y: 4, text: '4%' },
        { 'x': 'Opera', y: 11, text: '11%' },
        { 'x': 'Android', y: 12, text: '12%' }
      ],
      legendSettings: { visible: false },
      dataLabel: { visible: true, position: 'Inside', name: 'value' },
      tooltip: {
        enable: true, header: '<b>${point.x}</b>', format: 'Composition:
        <b>${point.y}</b>'
      },
    };
  };

```

```

    },
    provide: {
      accumulationchart: [PieSeries, AccumulationDataLabel,
        AccumulationTooltip]
    },
    mounted() {
      this.$refs.accumulationInstance.height = "100%";
      this.$refs.accumulationInstance.width = "100%";
    }
  });
  export default {
    data: function() {
      return {
        spacing: [10,10],
        header: 'Add a Content',
        target: '.control-section',
        showCloseIcon: true,
        spline: function () {
          return { template : splineTemplate }
        },
        pie: function () {
          return { template : pietemplate }
        },
        pie1: function () {
          return { template: pietemplatel1}
        }
      };
    }
  }
</script>
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-layouts/styles/material.css";
@import "../node_modules/@syncfusion/ej2-buttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-inputs/styles/material.css";
/* DashboardLayout element styles */
#dashboard_default .e-panel .e-panel-content {
  vertical-align: middle;
  font-weight: 600;
  font-size: 20px;
  text-align: center;
  line-height: 100px;
}
.chart-content{
  height: 100%;
  width:100%;
}
#container{
  width: 100%;
  height: 100%;
}
#dashboard_default .e-panel {
  transition:none !important;
}
</style>

```



```
{% previewsample "page.domainurl/code-snippet/dashboard-layout/content-cs1" %}
```

You can refer to our [Vue Dashboard Layout](#) feature tour page for its groundbreaking feature representations. You can also explore our [Vue Dashboard Layout example](#) to know how to present and manipulate data.

#### Add remove panels in Vue Dashboard layout component

In real-time cases, the data being presented within the dashboard should be updated frequently which includes adding or removing the data dynamically within the dashboard. This can be easily achieved by using the [addPanel](#) and [removePanel](#) public methods of the component.

#### Add or remove panels dynamically

Panels can be added dynamically by using the [addPanel](#) public method by passing the [panel](#) property as parameter. Also, they can be removed dynamically by using the [removePanel](#) public method by passing the [panel id](#) value as a parameter.

It is also possible to remove all the panels in a Dashboard Layout by calling [removeAll](#) method.

```
`js
```

```
dashboard.removeAll();
```

The following sample demonstrates how to add and remove the panels dynamically in the dashboard layout component. Here, panels can be added in any desired position of required size by selecting them in the numeric boxes and clicking add button and remove them by selecting the ID of the panel.

#### APP.VUE

```
<template>
  <div>
    <div className="col-lg-8 control-section" id="control_dash">
      <div className="content-wrapper">
        <div id = "dashboardElement">
          <!-- DashboardLayout element declaration -->
          <ejs-dashboardlayout id='dashboard_default'
:columns="5">
            <e-panels>
              <e-panel id="Panel0" :sizeX="1" :sizeY="1"
:row="0" :col="0" content="<div class='content'>0</div>"></e-panel>
              <e-panel id="Panel1" :sizeX="3" :sizeY="2"
:row="0" :col="1" content="<div class='content'>1</div>"></e-panel>
              <e-panel id="Panel2" :sizeX="1" :sizeY="3"
:row="0" :col="4" content="<div class='content'>2</div>"></e-panel>
              <e-panel id="Panel3" :sizeX="1" :sizeY="1"
:row="1" :col="0" content="<div class='content'>3</div>"></e-panel>
              <e-panel id="Panel4" :sizeX="2" :sizeY="1"
:row="2" :col="0" content="<div class='content'>4</div>"></e-panel>
              <e-panel id="Panel5" :sizeX="1" :sizeY="1"
:row="2" :col="2" content="<div class='content'>5</div>"></e-panel>
              <e-panel id="Panel6" :sizeX="1" :sizeY="1"
:row="2" :col="3" content="<div class='content'>6</div>"></e-panel>
            </e-panels>
          </ejs-dashboardlayout>
          <!-- end of dashboardlayout element -->
        </div>
```

```

        </div>
    </div>
    <div className="col-lg-4 property-section dashboard" id="api_property">
        <div className="row property-panel-content">
            <div className="card-body">
                <div className="form-group row">
                    <table id ="add">
                        <tbody>
                            <tr><td id="property">Add Panel
properties</td></tr>
                                <tr>
                                    <td>SizeX</td>
                                    <td>
                                        <ejs-numerictextbox id="sizeX"
placeholder="Ex: 1" :value="value" :min="min" :max="max"
:floatLabelType="floatLabelType"></ejs-numerictextbox>
                                        </td>
                                    </tr>
                                    <tr>
                                        <td>SizeY</td>
                                        <td>
                                            <ejs-numerictextbox id="sizeY"
placeholder="Ex: 1" :value="value" :min="min" :max="max"
:floatLabelType="floatLabelType"></ejs-numerictextbox>
                                            </td>
                                        </tr>
                                    <tr>
                                        <td>Row</td>
                                        <td>
                                            <ejs-numerictextbox id="row"
placeholder="Ex: 1" :value="value" :min="rowmin" :max="rowmax"
:floatLabelType="floatLabelType"></ejs-numerictextbox>
                                            </td>
                                        </tr>
                                    <tr>
                                        <td>Column</td>
                                        <td>
                                            <ejs-numerictextbox id="column"
placeholder="Ex: 1" :value="value" :min="colmin" :max="colmax"
:floatLabelType="floatLabelType"></ejs-numerictextbox>
                                            </td>
                                        </tr>
                                    <tr>
                                        <td>
                                            <ejs-button id="add" class="e-primary"
v-on:click.native="onAdd" >Add Panel</ejs-button>
                                        </td>
                                    </tr>
                                </tbody>
                            </table>
                            <table id ="remove">
                                <tbody>
                                    <tr><td id="property">Remove panel
properties</td></tr>
                                        <tr>
                                            <td> Panel Id </td>
                                            <td>

```

```
<ejs-dropdownlist id='dropdown'
placeholder='Select a id value':dataSource='data'></ejs-dropdownlist>
</td>
</tr>
<tr>

```

```

        var panel = [{
            'id': "Panel"+ this.count.toString(),
            'sizeX': sizeX.value,
            'sizeY': sizeY.value,
            'row': row.value,
            'col': column.value,
            'content': "<div class='content'>" + this.count + "</div>"
        }];
        dropdownObject.dataSource.push("Panel" + this.count.toString());
        dropdownObject.refresh();
        this.count = this.count + 1;
        dashboardObj.addPanel(panel[0]);
    },
    onRemove: function(args) {
        var dashboardObj =
document.getElementById("dashboard_default").ej2_instances[0];
        var dropdownObject =
document.getElementById("dropdown").ej2_instances[0];
        dashboardObj.removePanel(dropdownObject.value);

        dropdownObject.dataSource.splice(dropdownObject.dataSource.indexOf(dropdownObject.value), 1);
        dropdownObject.value = null;
        dropdownObject.refresh();
    },
},
}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-layouts/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-dropdowns/styles/material.css";
@import "../node_modules/@syncfusion/ej2-buttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-inputs/styles/material.css";
/* DashboardLayout element styles */
#dashboard_default .e-panel .e-panel-content {
    vertical-align: middle;
    font-weight: 600;
    font-size: 20px;
    text-align: center;
    line-height: 100px;
}
#control_dash {
    display: block;
    width: 59%;
    float: left;
}
#api_property {
    display: inline-block;
    margin: 10px;
}
#add {
    border: 1px solid black;
    padding: 10px;
    margin: 10px;
}
#remove {

```

```

border: 1px solid black;
padding: 10px;
margin: 10px;
width: 377px;
}
#property {
padding:10px;
}
#removeButton {
border: 1px solid black;
padding: 10px;
margin: 10px;
width: 125px;
}
#dashboardElement {
padding-top: 30px;
}
#dashboard_default .e-panel {
transition:none !important;
}
</style>

```

{% previewsample "page.domainurl/code-snippet/dashboard-layout/add-remove-panel-cs1" %}

You can refer to our [Vue Dashboard Layout](#) feature tour page for its groundbreaking feature representations. You can also explore our [Vue Dashboard Layout example](#) to know how to present and manipulate data.

## Interaction With Panels

### Dragging moving of panels in Vue Dashboard layout component

The Dashboard Layout component is provided with dragging functionality to drag and reorder the panels within the layout. While dragging a panel, a holder will be highlighted below the panel indicating the panel placement on panel drop. This helps the user to decide whether to place the panel in the current position or revert to previous position without disturbing the layout.

If one or more panels collide while dragging, then the colliding panels will be pushed towards the left or right or top or bottom direction where an adaptive space for the collided panel is available. The position changes of these collided panels will be updated dynamically during dragging of a panel, so the user can conclude whether to place the panel in the current position or not.

While dragging a panel in Dashboard layout the following dragging events will be triggered,

- [dragStart](#) - Triggers when panel drag starts
- [drag](#) - Triggers when panel is being dragged
- [dragStop](#) - Triggers when panel drag stops

The following sample demonstrates dragging and pushing of panels. For example, while dragging the panel 0 over panel 1, these panels get collided and push the panel 1 towards the feasible direction, so that, the panel 0 gets placed in the panel 1 position.

### APP.VUE

```
<template>
```

```

<div class="control-section">
  <!-- DashboardLayout element declaration -->
  <ejs-dashboardlayout id='dashboard_layout' :columns="7"
:cellSpacing='cellSpacing' :dragStart="onDragStart" :drag="onDrag"
:dragStop="onDragStop" >
    <e-panels>
      <e-panel :sizeX="1" :sizeY="1" :row="0" :col="0"
content="<div class='content'>0</div>"></e-panel>
      <e-panel :sizeX="3" :sizeY="2" :row="0" :col="1"
content="<div class='content'>1</div>"></e-panel>
      <e-panel :sizeX="1" :sizeY="3" :row="0" :col="4"
content="<div class='content'>2</div>"></e-panel>
      <e-panel :sizeX="1" :sizeY="1" :row="1" :col="0"
content="<div class='content'>3</div>"></e-panel>
      <e-panel :sizeX="2" :sizeY="1" :row="2" :col="0"
content="<div class='content'>4</div>"></e-panel>
      <e-panel :sizeX="1" :sizeY="1" :row="2" :col="2"
content="<div class='content'>5</div>"></e-panel>
      <e-panel :sizeX="1" :sizeY="1" :row="2" :col="3"
content="<div class='content'>6</div>"></e-panel>
    </e-panels>
  </ejs-dashboardlayout>
  <!-- end of dashboardlayout element -->
</div>
</template>
<script>
import Vue from "vue";
// Import syncfusion dashboardlayout component from layouts package
import { DashboardLayoutPlugin } from "@syncfusion/ej2-vue-layouts";
Vue.use(DashboardLayoutPlugin);
export default {
  data: function() {
    return {
      cellSpacing: [20, 20]
    };
  },
  methods: {
    onDragStart: function(args) {
      console.log("Drag start");
    },
    onDrag: function(args) {
      console.log("Dragging");
    },
    onDragStop: function(args) {
      console.log("Drag Stop");
    }
  }
}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-layouts/styles/material.css";
/* DashboardLayout element styles */
#dashboard_layout .e-panel .e-panel-content {
  vertical-align: middle;
  font-weight: 600;
  font-size: 20px;

```

```

    text-align: center;
    line-height: 80px;
  }
  #dashboard_layout .e-panel {
    transition:none !important;
  }
</style>

```

{% previewsample "page.domainurl/code-snippet/dashboard-layout/setting-cell-spacing-cs1" %}

### *Customizing the dragging handler*

Initially, the complete panel will act as the handler for dragging the panel such that the dragging action occurs on clicking anywhere over a panel. However, this dragging handler for the panels can be customized using the [draggableHandle](#) property to restrict the dragging action within a particular element in the panel.

The following sample demonstrates customizing the dragging handler of the panels where the dragging action of panel occurs only with the header of the panel.

### **APP.VUE**

```

<template>
  <div className="control-section" id="control_dash">
    <div className="content-wrapper">
      <!-- DashboardLayout element declaration -->
      <ejs-dashboardlayout ref="DashbordInstance" :columns="6"
id='edit_dashboard' :allowResizing="false" :allowDragging="true"
:draggableHandle="draggable" >
        <e-panels>
          <e-panel :row="0" :col="0" :sizeX="3" :sizeY="2"
header="<div>Product usage ratio</div>" :content="pie"></e-panel>
          <e-panel :row="0" :col="3" :sizeX="3" :sizeY="2"
header="<div>Mobile browsers usage</div>" :content="pie1"></e-panel>
          <e-panel :row="1" :col="0" :sizeX="3" :sizeY="2"
header="<div>Spline Chart</div>" :content="spline"></e-panel>
        </e-panels>
      </ejs-dashboardlayout>
      <!-- end of dashboardlayout element -->
    </div>
  </div>
</template>
<script>
import Vue from "vue";
// Import syncfusion dashboardlayout component from layouts package
import { DashboardLayoutPlugin } from "@syncfusion/ej2-vue-layouts";
// Import syncfusion chart component from charts package
import { AccumulationChartPlugin, PieSeries, AccumulationDataLabel,
AccumulationTooltip, ChartPlugin, SplineAreaSeries, Legend, DateTime } from
"@syncfusion/ej2-vue-charts";
Vue.use(ChartPlugin);
Vue.use(AccumulationChartPlugin);
Vue.use(DashboardLayoutPlugin);
var splineTemplate = Vue.component("contentTemp1", {
  template: `
    <div id="container" style='display:block;height:100%, width:100%;'>
      <!-- Chart element declaration -->

```

```

<ejs-chart class="chart-content" ref="splineInstance"
style='display:block;height:100%, width:100%;':primaryXAxis='primaryXAxis'
:primaryYAxis='primaryYAxis'
:chartArea='chartArea' :height='height' :width='width'
:border='border'>
  <e-series-collection>
    <e-series :dataSource='seriesData' type='SplineArea' xName='x'
yName='y' name='Jan' width=2 opacity=0.5 :fill="fill10"></e-series>
    <e-series :dataSource='seriesData1' type='SplineArea' xName='x'
yName='y' name='Feb' width=2 opacity=0.5 :fill="fill11"></e-series>
  </e-series-collection>
</ejs-chart>
<!-- end of chart element -->
</div>`,
data: function() {
  return {
    seriesData: [
      { x: new Date(2002, 0, 1), y: 2.2 }, { x: new Date(2003, 0, 1), y:
3.4 },
      { x: new Date(2004, 0, 1), y: 2.8 }, { x: new Date(2005, 0, 1), y:
1.6 },
      { x: new Date(2006, 0, 1), y: 2.3 }, { x: new Date(2007, 0, 1), y:
2.5 },
      { x: new Date(2008, 0, 1), y: 2.9 }, { x: new Date(2009, 0, 1), y:
3.8 },
      { x: new Date(2010, 0, 1), y: 1.4 }, { x: new Date(2011, 0, 1), y:
3.1 }
    ],
    seriesData1: [
      { x: new Date(2002, 0, 1), y: 2 }, { x: new Date(2003, 0, 1), y: 1.7
},
      { x: new Date(2004, 0, 1), y: 1.8 }, { x: new Date(2005, 0, 1), y:
2.1 },
      { x: new Date(2006, 0, 1), y: 2.3 }, { x: new Date(2007, 0, 1), y:
1.6 },
      { x: new Date(2008, 0, 1), y: 1.5 }, { x: new Date(2009, 0, 1), y:
2.7 },
      { x: new Date(2010, 0, 1), y: 1.5 }, { x: new Date(2011, 0, 1), y:
2.2 }
    ],
    //Initializing Primary X Axis
    primaryXAxis: {
      valueType: 'DateTime',
      labelFormat: 'y',
      majorGridLines: { width: 0 },
      intervalType: 'Years',
      edgeLabelPlacement: 'Shift'
    },
    //Initializing Primary Y Axis
    primaryYAxis: {
      labelFormat: '{value}%',
      lineStyle: { width: 0 },
      majorTickLines: { width: 0 },
      minorTickLines: { width: 0 }
    },
    chartArea: {
      border: {

```



```

        width: 0
      }
    },
    border: {
      color: 'transparent'
    },
    width: "100%",
    fill1: 'rgb(0, 189, 174)',
    fill0: 'rgb(239, 183, 202)',
    height: "99%"
  };
},
provide: {
  chart: [SplineAreaSeries, Legend, DateTime]
},
mounted() {
  this.$refs.splineInstance.height = "100%";
  this.$refs.splineInstance.width = "100%";
}
});
var pietemplate = Vue.component("contentTemp2", {
  template: `
    <div id="app" style='display:block;height:100%; width:100%;'>
      <ejs-accumulationchart class="chart-content"
ref="accumulationInstance" style='display:block;height:100%; width:100%;'
:legendSettings="legendSettings" :tooltip="tooltip">
        <e-accumulation-series-collection>
          <e-accumulation-series :dataSource='seriesData' xName='x'
yName='y' innerRadius="40%" :dataLabel="dataLabel"> </e-accumulation-series>
        </e-accumulation-series-collection>
      </ejs-accumulationchart>
    </div>`,
  data() {
    return {
      seriesData: [
        { x: 'TypeScript', y: 13, text: 'TS 13%' },
        { x: 'React', y: 12.5, text: 'Reat 12.5%' },
        { x: 'MVC', y: 12, text: 'MVC 12%' },
        { x: 'Core', y: 12.5, text: 'Core 12.5%' },
        { x: 'Vue', y: 10, text: 'Vue 10%' },
        { x: 'Angular', y: 40, text: 'Angular 40%' }
      ],
      legendSettings: { visible: false },
      dataLabel: { visible: true, position: 'Inside', name: 'value' },
      tooltip: {
        enable: true, header: '<b>${point.x}</b>', format: 'Composition:
<b>${point.y}</b>'
      },
    };
  },
  provide: {
    accumulationchart: [PieSeries, AccumulationDataLabel,
AccumulationTooltip]
  },
  mounted() {
    this.$refs.accumulationInstance.height = "100%";
    this.$refs.accumulationInstance.width = "100%";
  }
});

```

```

    }
  });
  var pietemplate1 = Vue.component("contentTemp3", {
    template: `
      <div id="app1" style='display:block;height:100%; width:100%;'>
        <ejs-accumulationchart class="chart-content"
          ref="accumulationInstance" style='display:block;height:100%; width:100%;'
          :legendSettings="legendSettings" :tooltip="tooltip">
          <e-accumulation-series-collection>
            <e-accumulation-series :dataSource='seriesData' xName='x'
              yName='y' innerRadius="40%" :dataLabel="dataLabel"> </e-accumulation-series>
          </e-accumulation-series-collection>
        </ejs-accumulationchart>
      </div>`,
    data() {
      return {
        seriesData: [
          { 'x': 'Chrome', y: 37, text: '37%' },
          { 'x': 'UC Browser', y: 17, text: '17%' },
          { 'x': 'iPhone', y: 19, text: '19%' },
          { 'x': 'Others', y: 4, text: '4%' },
          { 'x': 'Opera', y: 11, text: '11%' },
          { 'x': 'Android', y: 12, text: '12%' }
        ],
        legendSettings: { visible: false },
        dataLabel: { visible: true, position: 'Inside', name: 'value' },
        tooltip: {
          enable: true, header: '<b>${point.x}</b>', format: 'Composition:
          <b>${point.y}</b>'
        },
      };
    },
    provide: {
      accumulationchart: [PieSeries, AccumulationDataLabel,
        AccumulationTooltip]
    },
    mounted() {
      this.$refs.accumulationInstance.height = "100%";
      this.$refs.accumulationInstance.width = "100%";
    }
  });
  export default {
    data: function() {
      return {
        spacing: [10,10],
        header: 'Add a Content',
        target: '.control-section',
        draggable: '.e-panel-header',
        showCloseIcon: true,
        spline: function () {
          return { template : splineTemplate }
        },
        pie: function () {
          return { template : pietemplate }
        },
        piel: function () {
          return { template: pietemplate1}
        }
      }
    }
  }

```

```

    }
    };
  }
}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-layouts/styles/material.css";
@import "../node_modules/@syncfusion/ej2-buttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-inputs/styles/material.css";
/* DashboardLayout element styles */
#dashboard_default .e-panel .e-panel-content {
  vertical-align: middle;
  font-weight: 600;
  font-size: 20px;
  text-align: center;
  line-height: 100px;
}
.chart-content{
  height: 100%;
  width:100%;
}
#container{
  width: 100%;
  height: 100%;
}
#dashboard_default .e-panel {
  transition:none !important;
}
</style>

```

{% previewsample "page.domainurl/code-snippet/dashboard-layout/drag-handler-cs1" %}

By default, the dragging of panels is enabled in Dashboard Layout. It can also be disabled with the help of [allowDragging](#) API. Setting [allowDragging](#) to false disables the dragging functionality in Dashboard Layout.

The following sample demonstrates Dashboard Layout with dragging support disabled.

#### APP.VUE

```

<template>
  <div class="control-section">
    <!-- DashboardLayout element declaration -->
    <ejs-dashboardlayout id='dashboard_layout' :columns="5"
:cellSpacing='cellSpacing' :allowDragging ='allowDragging' >
      <e-panels>
        <e-panel :sizeX="1" :sizeY="1" :row="0" :col="0"
content="<div class='content'>0</div>"></e-panel>
        <e-panel :sizeX="3" :sizeY="2" :row="0" :col="1"
content="<div class='content'>1</div>"></e-panel>
        <e-panel :sizeX="1" :sizeY="3" :row="0" :col="4"
content="<div class='content'>2</div>"></e-panel>
        <e-panel :sizeX="1" :sizeY="1" :row="1" :col="0"
content="<div class='content'>3</div>"></e-panel>
        <e-panel :sizeX="2" :sizeY="1" :row="2" :col="0"
content="<div class='content'>4</div>"></e-panel>
      </e-panels>
    </div>
  </div>

```

```

        <e-panel :sizeX="1" :sizeY="1" :row="2" :col="2"
content="<div class='content'>5</div>"></e-panel>
        <e-panel :sizeX="1" :sizeY="1" :row="2" :col="3"
content="<div class='content'>6</div>"></e-panel>
    </e-panels>
</ejs-dashboardlayout>
<!-- end of dashboardlayout element -->
</div>
</template>
<script>
import Vue from "vue";
// Import syncfusion dashboardlayout component from layouts package
import { DashboardLayoutPlugin } from "@syncfusion/ej2-vue-layouts";
Vue.use(DashboardLayoutPlugin);
export default {
  data: function() {
    return {
      cellSpacing: [20, 20],
      allowDragging: false
    };
  },
};
</script>
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-layouts/styles/material.css";
/* DashboardLayout element styles */
#dashboard_layout .e-panel .e-panel-content {
  vertical-align: middle;
  font-weight: 600;
  font-size: 20px;
  text-align: center;
  line-height: 80px;
}
#dashboard_layout .e-panel {
  transition:none !important;
}
</style>

```

{% previewsample "page.domainurl/code-snippet/dashboard-layout/disable-dragging-cs1" %}

You can refer to our [Vue Dashboard Layout](#) feature tour page for its groundbreaking feature representations. You can also explore our [Vue Dashboard Layout example](#) to know how to present and manipulate data.

### Moving panels in Vue Dashboard layout component

Other than drag and drop, it is possible to move the panels in Dashboard Layout programmatically. This can be achieved using [movePanel](#) method. The method is invoked as follows,

```

`js
movePanel(id, row, col)
`

```

Where,

- id - ID of the panel which needs to be moved.
- row - New row position for moving the panel.
- col - New column position for moving the panel.

Each time a panel's position is changed(Programatically or through UI interaction), the Dashboard Layout's [change](#) event will be triggered.

The following sample demonstrates moving a panel programatically to a new position in the Dashboard Layout's [created](#) event.

#### APP.VUE

```
<template>
  <div class="control-section">
    <!-- DashboardLayout element declaration -->
    <ejs-dashboardlayout id='dashboard_layout' ref="dashboard"
:columns="5" :cellSpacing='cellSpacing' :created="onCreated"
:change="onChange" >
      <e-panels>
        <e-panel :sizeX="1" :sizeY="1" :row="0" :col="0"
content="<div class='content'>0</div>"></e-panel>
        <e-panel :sizeX="3" :sizeY="2" :row="0" :col="1"
content="<div class='content'>1</div>"></e-panel>
        <e-panel :sizeX="1" :sizeY="3" :row="0" :col="4"
content="<div class='content'>2</div>"></e-panel>
        <e-panel :sizeX="1" :sizeY="1" :row="1" :col="0"
content="<div class='content'>3</div>"></e-panel>
        <e-panel :sizeX="2" :sizeY="1" :row="2" :col="0"
content="<div class='content'>4</div>"></e-panel>
        <e-panel :sizeX="1" :sizeY="1" :row="2" :col="2"
content="<div class='content'>5</div>"></e-panel>
        <e-panel :sizeX="1" :sizeY="1" :row="2" :col="3"
content="<div class='content'>6</div>"></e-panel>
      </e-panels>
    </ejs-dashboardlayout>
    <!-- end of dashboardlayout element -->
  </div>
</template>
<script>
import Vue from "vue";
// Import syncfusion dashboardlayout component from layouts package
import { DashboardLayoutPlugin } from "@syncfusion/ej2-vue-layouts";
Vue.use(DashboardLayoutPlugin);
export default {
  data: function() {
    return {
      cellSpacing: [10, 10]
    };
  },
  methods: {
    onCreate: function(args) {
      // movePanel("id", row, col)
      this.$refs.dashboard.$el.ej2_instances[0].movePanel("layout_0",1,0);
    },
    //Dashboard Layout's change event function
  }
}
```

```

        onChange: function(args) {
            console.log("Change event Triggered");
        },
    },
}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-layouts/styles/material.css";
/* DashboardLayout element styles */
#dashboard_layout .e-panel .e-panel-content {
    vertical-align: middle;
    font-weight: 600;
    font-size: 20px;
    text-align: center;
    line-height: 80px;
}
#dashboard_layout .e-panel {
    transition:none !important;
}
</style>

```

{% previewsample "page.domainurl/code-snippet/dashboard-layout/moving-cs1" %}

You can refer to our [Vue Dashboard Layout](#) feature tour page for its groundbreaking feature representations. You can also explore our [Vue Dashboard Layout example](#) to know how to present and manipulate data.

### Resizing of panels in Vue Dashboard layout component

The Dashboard Layout component is also provided with the panel resizing functionality which can be enabled or disabled using the [allowResizing](#) property. This functionality allows you to resize the panels dynamically through UI interactions using the resizing handlers which controls the panel resizing in various directions.

Initially, the panels can be resized only in south-east direction. However, panels can also be resized in east, west, north, south and south-west directions by defining the required directions with the [resizableHandles](#) property.

On resizing a panel in Dashboard layout the following events will be triggered,

- [resizeStart](#) - Triggers when panel resize starts
- [resize](#) - Triggers when panel is being resized
- [resizeStop](#) - Triggers when panel resize stops

The following sample demonstrates how to enable and disable the resizing of panels in the Dashboard Layout component in different directions.

### APP.VUE

```

<template>
  <div class="control-section">
    <!-- DashboardLayout element declaration -->
    <ejs-dashboardlayout id='dashboard_default' ref="dashboard"
      :cellSpacing='cellSpacing' :allowResizing='true'

```

```

:resizableHandles='resizableHandles' :columns="6"
:resizeStart="onResizeStart" :resize="onResize" :resizeStop="onResizeStop">
  <e-panels>
    <e-panel :sizeX="1" :sizeY="1" :row="0" :col="0" content="<div
class='content'>0</div>"></e-panel>
    <e-panel :sizeX="3" :sizeY="2" :row="0" :col="1" content="<div
class='content'>1</div>"></e-panel>
    <e-panel :sizeX="1" :sizeY="3" :row="0" :col="4" content="<div
class='content'>2</div>"></e-panel>
    <e-panel :sizeX="1" :sizeY="1" :row="1" :col="0" content="<div
class='content'>3</div>"></e-panel>
    <e-panel :sizeX="2" :sizeY="1" :row="2" :col="0" content="<div
class='content'>4</div>"></e-panel>
    <e-panel :sizeX="1" :sizeY="1" :row="2" :col="2" content="<div
class='content'>5</div>"></e-panel>
    <e-panel :sizeX="1" :sizeY="1" :row="2" :col="3" content="<div
class='content'>6</div>"></e-panel>
  </e-panels>
</ejs-dashboardlayout>
<!-- end of dashboardlayout element -->
</div>
</template>
<script>
import Vue from "vue";
// Import syncfusion dashboardlayout component from layouts package
import { DashboardLayoutPlugin } from "@syncfusion/ej2-vue-layouts";
Vue.use(DashboardLayoutPlugin);
export default {
  data: function() {
    return {
      cellSpacing: [10, 10],
      resizableHandles: ['e-south-east', 'e-east', 'e-west', 'e-north', 'e-
south'],
    };
  },
  methods: {
    //Dashboard Layout's resizestart event function
    onResizeStart: function(args) {
      console.log("Resize Start");
    },
    //Dashboard Layout's resize event function
    onResize: function(args) {
      console.log("Resizing");
    },
    //Dashboard Layout's resizestop event function
    onResizeStop: function(args) {
      console.log("Resize Stop")
    }
  }
}
</script>
<style>
@import "../../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../../node_modules/@syncfusion/ej2-vue-
layouts/styles/material.css";
/* DashboardLayout element styles */
#dashboard_default .e-panel .e-panel-container .content {

```

```

vertical-align: middle;
font-weight: 600;
font-size: 20px;
text-align: center;
line-height: 80px;
}
#dashboard_default .e-panel {
  transition:none !important;
}
</style>

```

{% previewsample "page.domainurl/code-snippet/dashboard-layout/resizing-cs1" %}

### *Resizing panels programatically*

The Dashboard Layout panels can also be resized programatically by using [resizePanel](#) method. The method is invoked as follows,

```
`js
```

```
resizePanel(id, sizeX, sizeY)
```

```
,
```

Where,

- id - ID of the panel which needs to be resized.
- sizeX - New panel width in cells count for resizing the panel.
- sizeY - New panel height in cells count for resizing the panel.

The following sample demonstrates resizing panels programatically in the Dashboard Layout's [created](#) event.

### **APP.VUE**

```

<template>
  <div class="control-section">
    <!-- DashboardLayout element declaration -->
    <ejs-dashboardlayout id='dashboard_default' ref="dashboard"
:cellSpacing='cellSpacing' :columns="5" :created="onCreated" >
      <e-panels>
        <e-panel :sizeX="1" :sizeY="1" :row="0" :col="0" content="<div
class='content'>0</div>"></e-panel>
        <e-panel :sizeX="3" :sizeY="2" :row="0" :col="1" content="<div
class='content'>1</div>"></e-panel>
        <e-panel :sizeX="1" :sizeY="3" :row="0" :col="4" content="<div
class='content'>2</div>"></e-panel>
        <e-panel :sizeX="1" :sizeY="1" :row="1" :col="0" content="<div
class='content'>3</div>"></e-panel>
        <e-panel :sizeX="2" :sizeY="1" :row="2" :col="0" content="<div
class='content'>4</div>"></e-panel>
        <e-panel :sizeX="1" :sizeY="1" :row="2" :col="2" content="<div
class='content'>5</div>"></e-panel>
        <e-panel :sizeX="1" :sizeY="1" :row="2" :col="3" content="<div
class='content'>6</div>"></e-panel>
      </e-panels>
    </ejs-dashboardlayout>
  </div>

```



```

    <!-- end of dashboardlayout element -->
  </div>
</template>
<script>
import Vue from "vue";
// Import syncfusion dashboardlayout component from layouts package
import { DashboardLayoutPlugin } from "@syncfusion/ej2-vue-layouts";
Vue.use(DashboardLayoutPlugin);
export default {
  data: function() {
    return {
      cellSpacing: [10, 10]
    };
  },
  methods: {
    //Dashboard Layout's create event function
    onCreated: function(args) {
      // resizePanel("id", sizeX, sizeY)
      this.$refs.dashboard.$el.ej2_instances[0].resizePanel("layout_4", 1,
1);
      this.$refs.dashboard.$el.ej2_instances[0].resizePanel("layout_5", 2,
1);
    },
  }
}
</script>
<style>
@import "../..../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../..../node_modules/@syncfusion/ej2-vue-
layouts/styles/material.css";
/* DashboardLayout element styles */
#dashboard_default .e-panel .e-panel-container .content {
  vertical-align: middle;
  font-weight: 600;
  font-size: 20px;
  text-align: center;
  line-height: 80px;
}
#dashboard_default .e-panel {
  transition:none !important;
}
</style>

```

{% previewsample "page.domainurl/code-snippet/dashboard-layout/resize-panel-cs1" %}

You can refer to our [Vue Dashboard Layout](#) feature tour page for its groundbreaking feature representations. You can also explore our [Vue Dashboard Layout example](#) to know how to present and manipulate data.

### Floating of panels in Vue Dashboard layout component

The floating functionality of the component allows you to effectively use the entire layout for the panel's placement. If the floating functionality is enabled, the panels within the layout get floated upwards automatically to occupy the empty cells available in previous rows. This functionality can be enabled or disabled using the [allowFloating](#) property of the component.

The following sample demonstrates how to enable or disable the floating of panels in the Dashboard Layout component.

#### APP.VUE

```
<template>
  <div>
    <div>
      <!-- Button element declaration -->
      <ejs-button id="toggle" ref="toggle" class="e-flat e-primary
e-outline" :isToggle="true" v-on:click.native="onChange" >Enable
Floating</ejs-button>
      <!-- end of button element -->
    </div>
    <div id="control_dash">
      <!-- DashboardLayout element declaration -->
      <ejs-dashboardlayout id='dashboard_default' ref="dashboard"
:allowFloating="false" :cellSpacing='cellSpacing' :columns="6">
        <e-panels>
          <e-panel :sizeX="2" :sizeY="2" :row="1" :col="0"
content="<div class='content'>0</div>"></e-panel>
          <e-panel :sizeX="2" :sizeY="2" :row="2" :col="2"
content="<div class='content'>1</div>"></e-panel>
          <e-panel :sizeX="2" :sizeY="2" :row="3" :col="4"
content="<div class='content'>2</div>"></e-panel>
        </e-panels>
      </ejs-dashboardlayout>
      <!-- end of dashboardlayout element -->
    </div>
  </div>
</template>
<script>
import Vue from "vue";
// Import syncfusion dashboardlayout component from layouts package
import { DashboardLayoutPlugin } from "@syncfusion/ej2-vue-layouts";
// Import syncfusion button component from buttons package
import { ButtonPlugin } from "@syncfusion/ej2-vue-buttons";
Vue.use(DashboardLayoutPlugin);
Vue.use(ButtonPlugin);
export default {
  data: function() {
    return {
      cellSpacing: [10, 10],
      resetPanels:[],
    };
  },
  methods: {
    onChange: function(args) {
      if (this.$refs.toggle.content == "Disable Floating and Reset") {
        this.$refs.toggle.content = 'Enable Floating';
        this.$refs.dashboard.allowFloating = false;
        this.$refs.dashboard.panels = this.resetPanels;
      } else {
        this.$refs.toggle.content = 'Disable Floating and Reset';
        this.$refs.dashboard.allowFloating = true;
      }
    }
  }
}
```

```

    },
    mounted() {
      this.resetPanels = this.$refs.dashboard.serialize();
      this.resetPanels[0].content = '<div class="content">0</div>';
      this.resetPanels[1].content = '<div class="content">1</div>';
      this.resetPanels[2].content = '<div class="content">2</div>';
    }
  }
</script>
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-layouts/styles/material.css";
@import "../node_modules/@syncfusion/ej2-buttons/styles/material.css";
#dashboard_default .e-panel .e-panel-content {
  vertical-align: middle;
  font-weight: 600;
  font-size: 20px;
  text-align: center;
  line-height: 100px;
}
#control_dash {
  display: block;
  width: 60%;
  float: left;
  padding-top: 30px;
}
#dashboard_default .e-panel {
  transition:none !important;
}
</style>

```

{% previewsample "page.domainurl/code-snippet/dashboard-layout/floating-cs1" %}

You can refer to our [Vue Dashboard Layout](#) feature tour page for its groundbreaking feature representations. You can also explore our [Vue Dashboard Layout example](#) to know how to present and manipulate data.

### Responsive adaptive in Vue Dashboard layout component

The control is provided with built-in responsive support, where panels within the layout get adjusted based on their parent element's dimensions to accommodate any resolution which relieves the burden of building responsive dashboards.

The dashboard layout is designed to automatically adapt with lower resolutions by transforming the entire layout into a stacked one, so that, the panels will be displayed in a vertical column. By default, whenever the screen resolution meets 600 px or lower resolutions this layout transformation occurs. This transformation can be modified for any user defined resolution by defining the `mediaQuery` property of the component.

The following sample demonstrates the usage of the `mediaQuery` property to turn out the layout into a stacked one in user defined resolution. Here, whenever, the window size reaches 700 px or lesser, the layout becomes a stacked layout.

#### APP.VUE

```
<template>
```

```

<div className="control-section">
  <!-- DashboardLayout element declaration -->
  <ejs-dashboardlayout id='dashboard_layout' ref="dashboard"
:cellSpacing='cellSpacing' :mediaQuery='mediaQuery' :columns="6">
    <e-panels>
      <e-panel :sizeX="1" :sizeY="1" :row="0" :col="0"
content="<div>0</div>"></e-panel>
      <e-panel :sizeX="2" :sizeY="2" :row="0" :col="1"
content="<div>1</div>"></e-panel>
      <e-panel :sizeX="1" :sizeY="3" :row="0" :col="4"
content="<div>2</div>"></e-panel>
      <e-panel :sizeX="1" :sizeY="1" :row="1" :col="0"
content="<div>3</div>"></e-panel>
    </e-panels>
  </ejs-dashboardlayout>
  <!-- end of dashboardlayout element -->
</div>
</template>
<script>
import Vue from "vue";
// Import syncfusion dashboardlayout component from layouts package
import { DashboardLayoutPlugin } from "@syncfusion/ej2-vue-layouts";
Vue.use(DashboardLayoutPlugin);
export default {
  data: function() {
    return {
      cellSpacing: [20, 20],
      mediaQuery: 'max-width: 700px',
    };
  }
}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-layouts/styles/material.css";
/* DashboardLayout element styles */
#dashboard_layout .e-panel .e-panel-container {
  vertical-align: middle;
  font-weight: 600;
  font-size: 20px;
  text-align: center;
  line-height: 80px;
}
#dashboard_layout .e-panel {
  transition:none !important;
}
</style>

```

{% previewsample "page.domainurl/code-snippet/dashboard-layout/responsive-adaptive-cs1" %}

You can refer to our [Vue Dashboard Layout](#) feature tour page for its groundbreaking feature representations. You can also explore our [Vue Dashboard Layout example](#) to know how to present and manipulate data.

### State maintenance in Vue Dashboard layout component

The current layout structure of the Dashboard Layout component can be obtained and saved to construct another dashboard with same panel structure using the `serialize` public method of the component. This method returns the component's current panel setting which can be used to construct a dashboard with the same layout settings.

The following sample demonstrates how to save and restore the state of the panels using the `serialize` method. Click Save to store the panel's settings and click Restore to restore the previously saved panel settings.

#### APP.VUE

```
<template>
  <div>
    <div className="col-lg-8 control-section" id="control_dash">
      <div className="content-wrapper">
        <!-- DashboardLayout element declaration -->
        <ejs-dashboardlayout id='dashboard_default' ref="dashboard"
:cellSpacing='cellSpacing' :columns="5" :created="onSave">
          <e-panels>
            <e-panel :sizeX="1" :sizeY="1" :row="0" :col="0"
content="<div class='content'>0</div>"></e-panel>
            <e-panel :sizeX="3" :sizeY="2" :row="0" :col="1"
content="<div class='content'>1</div>"></e-panel>
            <e-panel :sizeX="1" :sizeY="3" :row="0" :col="4"
content="<div class='content'>2</div>"></e-panel>
            <e-panel :sizeX="1" :sizeY="1" :row="1" :col="0"
content="<div class='content'>3</div>"></e-panel>
            <e-panel :sizeX="2" :sizeY="1" :row="2" :col="0"
content="<div class='content'>4</div>"></e-panel>
            <e-panel :sizeX="1" :sizeY="1" :row="2" :col="2"
content="<div class='content'>5</div>"></e-panel>
            <e-panel :sizeX="1" :sizeY="1" :row="2" :col="3"
content="<div class='content'>6</div>"></e-panel>
          </e-panels>
        </ejs-dashboardlayout>
        <!-- end of dashboardlayout element -->
      </div>
    <div className="col-lg-4 property-section dashboard"
id="api_property">
      <div className="row property-panel-content">
        <div className="card-body">
          <div className="form-group row">
            <table id="remove">
              <tbody>
                <tr><td> Properties Panel </td></tr>
                <tr>
                  <td>
                    <!-- Button element declaration -->
                    <ejs-button id="save" cssClass="e-
primary" v-on:click.native="onSave" >Save Panel</ejs-button>
                  </td>
                </tr>
              </tbody>
            </table>
            <!-- Button element declaration -->
          </div>
        </div>
      </div>
    </div>
  </div>
```

```

<ej2-button id="restore"
cssClass="e-flat e-outline" v-on:click.native="onRestore">Restore
Panel</ej2-button>

</td>
</tr>
</tbody>
</table>
</div>
</div>
</div>
</div>
</div>
</template>
<script>
import Vue from "vue";
// Import syncfusion dashboardlayout component from layouts package
import { DashboardLayoutPlugin } from "@syncfusion/ej2-vue-layouts";
// Import syncfusion button component from buttons package
import { ButtonPlugin } from "@syncfusion/ej2-vue-buttons";
Vue.use(DashboardLayoutPlugin);
Vue.use(ButtonPlugin);
export default {
  data: function() {
    return {
      cellSpacing: [20, 20],
      restoreModel: []
    };
  },
  methods: {
    // Restore the initial panels
    onRestore: function(args) {
      // Create instances for dashboardlayout element
      this.$refs.dashboard.$el.ej2_instances[0].panels =
this.$refs.restoreModel;
    },
    // Save the current panels
    onSave: function(args) {
      // Create instances for dashboardlayout element
      this.$refs.restoreModel =
this.$refs.dashboard.$el.ej2_instances[0].serialize();
      this.$refs.restoreModel[0].content = '<div
class="content">0</div>';
      this.$refs.restoreModel[1].content = '<div
class="content">1</div>';
      this.$refs.restoreModel[2].content = '<div
class="content">2</div>';
      this.$refs.restoreModel[3].content = '<div
class="content">3</div>';
      this.$refs.restoreModel[4].content = '<div
class="content">4</div>';
      this.$refs.restoreModel[5].content = '<div
class="content">5</div>';
      this.$refs.restoreModel[6].content = '<div
class="content">6</div>';
    },
  },
}

```

```

</script>
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-layouts/styles/material.css";
@import "../node_modules/@syncfusion/ej2-buttons/styles/material.css";
/* DashboardLayout element styles */
#dashboard_default .e-panel .e-panel-content {
    vertical-align: middle;
    font-weight: 600;
    font-size: 20px;
    text-align: center;
    line-height: 100px;
}
#control_dash {
    display: block;
    width: 60%;
    float: left;
}
#api_property {
    display: inline-block;
}
#float_id {
    border: 1px solid black;
    padding: 20px;
    margin: 30px;
}
#remove {
    border: 1px solid black;
    margin: 30px;
}
td {
    padding: 20px;
}
#dashboard_default .e-panel {
    transition: none !important;
}
</style>

```

{% previewsample "page.domainurl/code-snippet/dashboard-layout/state-maintenance-cs1" %}

You can refer to our [Vue Dashboard Layout](#) feature tour page for its groundbreaking feature representations. You can also explore our [Vue Dashboard Layout example](#) to know how to present and manipulate data.

### Style in Vue Dashboard layout component

The following content provides the exact CSS structure that can be used to modify the control's appearance based on the user preference.

#### Customizing the dashboard layout panel header

Use the following CSS to customize the dashboard layout panel header.

```

.e-dashboardlayout.e-control .e-panel .e-panel-container .e-panel-header {
color: #754131;

```

```
background-color: #c9e2f7;
text-align: center;
}
`
```

### Customizing the dashboard layout panel content

Use the following CSS to customize the dashboard layout panel content.

```
`
.e-dashboardlayout.e-control .e-panel .e-panel-container .e-panel-content {
background-color: #c9e2f7;
padding: 50px;
}
`
```

### Customizing the dashboard layout panel resize icon

Use the following CSS to customize the dashboard layout resize icon.

```
`
.e-dashboardlayout.e-control .e-panel .e-panel-container .e-resize.e-double{
color: #0378d5;
font-size: 30px;
height: 20px;
width: 20px;
}
`
```

### Customizing the dashboard layout panel background

Use the following CSS to customize the dashboard layout panel background.

```
`
.e-dashboardlayout.e-control.e-responsive {
background: #b3d3ed;
}
`
```

You can refer to our [Vue Dashboard Layout](#) feature tour page for its groundbreaking feature representations. You can also explore our [Vue Dashboard Layout example](#) to know how to present and manipulate data.

### Accessibility in Vue Dashboard Layout component

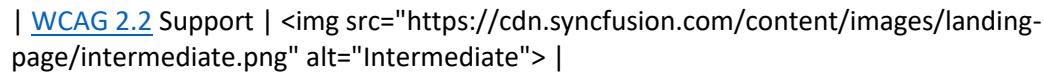
The Dashboard Layout component followed the accessibility guidelines and standards, including [ADA](#), [Section 508](#), [WCAG 2.2](#) standards, and [WCAG roles](#) that are commonly used to evaluate accessibility.

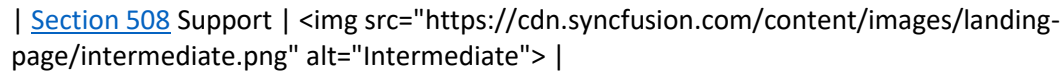


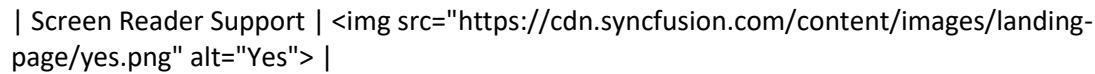
The accessibility compliance for the Dashboard Layout component is outlined below.

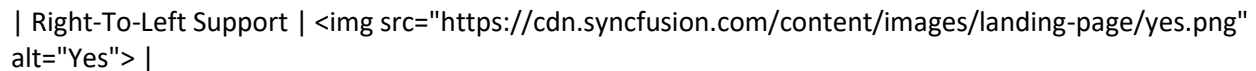
| Accessibility Criteria | Compatibility |

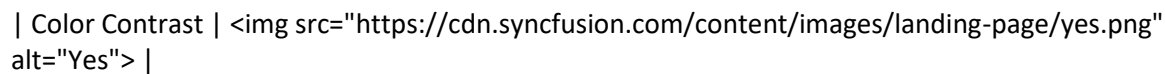
| -- | -- |

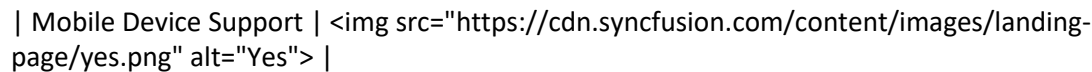
| [WCAG 2.2](#) Support |  |

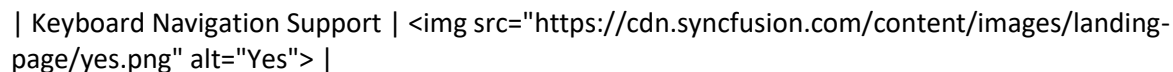
| [Section 508](#) Support |  |

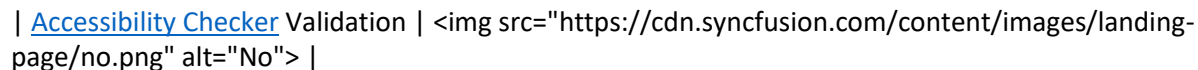
| Screen Reader Support |  |

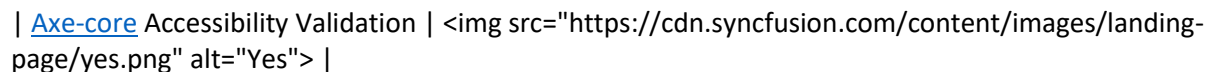
| Right-To-Left Support |  |

| Color Contrast |  |

| Mobile Device Support |  |

| Keyboard Navigation Support |  |

| [Accessibility Checker](#) Validation |  |

| [Axe-core](#) Accessibility Validation |  |

<style>

.post .post-content img {

display: inline-block;

margin: 0.5em 0;

}

</style>

<div> - All features of the component meet the requirement.</div>

<div> - Some features of the component do not meet the requirement.</div>

<div> - The component does not meet the requirement.</div>

### [WAI-ARIA attributes](#)

The Dashboard Layout component followed the [WAI-ARIA](#) patterns to meet the accessibility. The following ARIA attributes are used in the Dashboard Layout component:

| **Attributes** | **Purpose** |

| --- | --- |

| **role=list** | Indicates the role as a list for the Dashboard Layout element. |

| **role=listitem** | Indicates the role as a listitem for the Dashboard panels. |

| **role=presentation** | Indicates the role as a presentation for the table when the **showGridLines** property is enabled. |

| **aria-grabbed** | When the panel is chosen for dragging, the aria-grabbed attribute is set to "true." If it's set to "false," the element can be grabbed for drag-and-drop, but it won't be actively held. |

### Keyboard interaction

Keyboard support is not applicable for the Dashboard Layout.

### Ensuring accessibility

The Dashboard Layout component's accessibility levels are ensured through an [accessibility-checker](#) and [axe-core](#) software tools during automated testing.

The accessibility compliance of the Dashboard Layout component is shown in the following sample. Open the [sample](#) in a new window to evaluate the accessibility of the Dashboard Layout component with accessibility tools.

```
{% previewsample "https://ej2.syncfusion.com/accessibility/dashboard-layout.html" %}
```

See also

- [Accessibility in Syncfusion Vue components](#)

## DatePicker

### Getting Started with the Vue Datepicker Component in Vue 2

This article provides a step-by-step guide for setting up a Vue 2 project using [Vue-CLI](#) and integrating the Syncfusion Vue Datepicker component using the [Composition API](#) / [Options API](#).

### Prerequisites

[System requirements for Syncfusion Vue UI components](#)

### Dependencies

The list of dependencies required to use the DatePicker component in your application is given below:

```
`javascript
|-- @syncfusion/ej2-vue-calendars
|-- @syncfusion/ej2-base
|-- @syncfusion/ej2-data
|-- @syncfusion/ej2-vue-base
|-- @syncfusion/ej2-calendars
|-- @syncfusion/ej2-inputs
|-- @syncfusion/ej2-splitbuttons
|-- @syncfusion/ej2-lists
```

```
|-- @syncfusion/ej2-popups  
|-- @syncfusion/ej2-buttons  
,
```

### Setting up the Vue 2 project

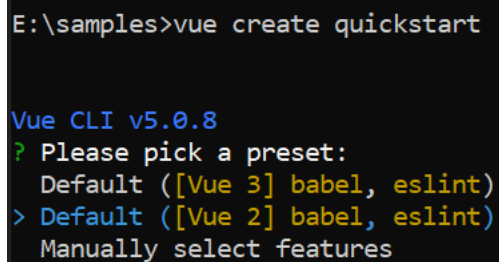
To generate a Vue 2 project using Vue-CLI, use the [vue create](#) command. Follow these steps to install Vue CLI and create a new project:

```
`bash  
npm install -g @vue/cli  
vue create quickstart  
cd quickstart  
npm run serve  
,
```

or

```
`bash  
yarn global add @vue/cli  
vue create quickstart  
cd quickstart  
yarn run serve  
,
```

When creating a new project, choose the option **Default ([Vue 2] babel, eslint)** from the menu.



```
E:\samples>vue create quickstart  
  
Vue CLI v5.0.8  
? Please pick a preset:  
  Default ([Vue 3] babel, eslint)  
> Default ([Vue 2] babel, eslint)  
  Manually select features
```

Once the **quickstart** project is set up with default settings, proceed to add Syncfusion components to the project.

### Add Syncfusion Vue packages

Syncfusion packages are available at [npmjs.com](https://www.syncfusion.com). To use Vue components, install the required npm package.

This article uses the [Vue DatePicker component](#) as an example. Install the **@syncfusion/ej2-vue-calendars** package by running the following command:

```
`bash
```

```
npm install @syncfusion/ej2-vue-calendars --save
```

```
,
```

```
or
```

```
`bash
```

```
yarn add @syncfusion/ej2-vue-calendars
```

```
,
```

### Import Syncfusion CSS styles

You can import themes for the Syncfusion Vue component in various ways, such as using CSS or SASS styles from npm packages, CDN, [CRG](#) and [Theme Studio](#). Refer to [themes topic](#) to know more about built-in themes and different ways to refer to themes in a Vue project.

In this article, the **Material** theme is applied using CSS styles, which are available in installed packages. The necessary **Material** CSS styles for the DatePicker component and its dependents were imported into the `<style>` section of **src/App.vue** file.

#### ~/SRC/APP.VUE

```
<style>
@import '../node_modules/@syncfusion/ej2-base/styles/material.css';
@import '../node_modules/@syncfusion/ej2-buttons/styles/material.css';
@import '../node_modules/@syncfusion/ej2-inputs/styles/material.css';
@import '../node_modules/@syncfusion/ej2-popups/styles/material.css';
@import "../node_modules/@syncfusion/ej2-vue-calendars/styles/material.css";
</style>
```

Note: If you want to refer the combined component styles, please make use of our [CRG](#) (Custom Resource Generator) in your application.

### Add Syncfusion Vue component

Follow the below steps to add the Vue DatePicker component using **Composition API** or **Options API**:

1\ First, import and register the DatePicker component in the **script** section of the **src/App.vue** file. If you are using the **Composition API**, you should add the **setup** attribute to the **script** tag to indicate that Vue will be using the **Composition API**.

#### COMPOSITION API (~/SRC/APP.VUE)

```
<script>
import { DatePickerComponent as EjsDatepicker } from '@syncfusion/ej2-vue-calendars';
</script>
```

#### OPTIONS API (~/SRC/APP.VUE)

```
<script>
import { DatePickerComponent } from '@syncfusion/ej2-vue-calendars';
export default {
  components: {
    'ejs-datepicker': DatePickerComponent
  }
}
```

```
}  
</script>
```

2\ In the **template** section, define the Datepicker component with the [placeholder](#) property.

#### ~/SRC/APP.VUE

```
<template>  
  <div id="app">  
    <div class='wrapper'>  
      <ejs-datepicker :placeholder="waterMark"></ejs-datepicker>  
    </div>  
  </div>  
</template>
```

3\ Declare the value for the **placeholder** property in the **script** section.

#### COMPOSITION API (~/SRC/APP.VUE)

```
<script>  
  const waterMark = 'Select a date';  
</script>
```

#### OPTIONS API (~/SRC/APP.VUE)

```
<script>  
  data () {  
    return {  
      waterMark : 'Select a date'  
    }  
  }  
</script>
```

Here is the summarized code for the above steps in the **src/App.vue** file:

#### COMPOSITION API (~/SRC/APP.VUE)

```
<template>  
  <div id="app">  
    <div class='wrapper'>  
      <ejs-datepicker :placeholder="waterMark"></ejs-datepicker>  
    </div>  
  </div>  
</template>  
<script setup>  
import { DatePickerComponent as EjsDatepicker } from '@syncfusion/ej2-vue-calendars';  
const waterMark = 'Select a date';  
</script>  
<style>  
@import '../node_modules/@syncfusion/ej2-base/styles/material.css';  
@import '../node_modules/@syncfusion/ej2-buttons/styles/material.css';  
@import '../node_modules/@syncfusion/ej2-inputs/styles/material.css';  
@import '../node_modules/@syncfusion/ej2-popups/styles/material.css';  
@import "../node_modules/@syncfusion/ej2-vue-calendars/styles/material.css";
```

```
.wrapper {  
  max-width: 250px;  
  margin: 0 auto;  
}  
</style>
```

### OPTIONS API (~SRC/APP.VUE)

```
<template>  
  <div id="app">  
    <div class='wrapper'>  
      <ejs-datepicker :placeholder="waterMark"></ejs-datepicker>  
    </div>  
  </div>  
</template>  
<script>  
import { DatePickerComponent } from '@syncfusion/ej2-vue-calendars';  
export default {  
  components: {  
    'ejs-datepicker': DatePickerComponent  
  },  
  data () {  
    return {  
      waterMark : 'Select a date'  
    }  
  }  
}  
</script>  
<style>  
@import '../node_modules/@syncfusion/ej2-base/styles/material.css';  
@import '../node_modules/@syncfusion/ej2-buttons/styles/material.css';  
@import '../node_modules/@syncfusion/ej2-inputs/styles/material.css';  
@import '../node_modules/@syncfusion/ej2-popups/styles/material.css';  
@import "../node_modules/@syncfusion/ej2-vue-calendars/styles/material.css";  
  .wrapper {  
    max-width: 250px;  
    margin: 0 auto;  
  }  
</style>
```

### Run the project

To run the project, use the following command:

```
`bash
```

```
npm run serve
```

```
,
```

or

```
`bash
```

```
yarn run serve
```

```
,
```

```
{% previewsample "page.domainurl/code-snippet/datepicker/getting-started-cs3" %}
```

### Setting the value, min and max dates

The following example demonstrates how to set the value, min and max dates on initializing the DatePicker. Here you can able to select a date within a range from 9th to 15th in a month of May 2017. To know more about range restriction in DatePicker, please refer this [page](#).

#### COMPOSITION API (~SRC/APP.VUE)

```
<template>
  <div id="app">
    <div class='wrapper'>
      <ejs-datepicker :min="minDate" :max="maxDate" :value="dateVal" ></ejs-
datepicker>
    </div>
  </div>
</template>
<script setup>
import { DatePickerComponent as EjsDatepicker } from '@syncfusion/ej2-vue-
calendars';
const minDate = new Date("05/09/2017");
const maxDate = new Date("05/15/2017");
const dateVal = new Date("05/11/2017");
</script>
<style>
  @import '../node_modules/@syncfusion/ej2-base/styles/material.css';
  @import '../node_modules/@syncfusion/ej2-buttons/styles/material.css';
  @import '../node_modules/@syncfusion/ej2-inputs/styles/material.css';
  @import '../node_modules/@syncfusion/ej2-popups/styles/material.css';
  @import "../node_modules/@syncfusion/ej2-vue-
calendars/styles/material.css";
  .wrapper {
    max-width: 250px;
    margin: 0 auto;
  }
</style>
```

#### OPTIONS API (~SRC/APP.VUE)

```
<template>
  <div id="app">
    <div class='wrapper'>
      <ejs-datepicker :min="minDate" :max="maxDate" :value="dateVal" ></ejs-
datepicker>
    </div>
  </div>
</template>
<script>
import { DatePickerComponent } from '@syncfusion/ej2-vue-calendars';
export default {
  components: {
    'ejs-datepicker': DatePickerComponent
  },
  data () {
    return {
      minDate : new Date("05/09/2017"),
```

```
      maxDate : new Date("05/15/2017"),
      dateVal : new Date("05/11/2017")
    }
  }
}
</script>
<style>
  @import '../node_modules/@syncfusion/ej2-base/styles/material.css';
  @import '../node_modules/@syncfusion/ej2-buttons/styles/material.css';
  @import '../node_modules/@syncfusion/ej2-inputs/styles/material.css';
  @import '../node_modules/@syncfusion/ej2-popups/styles/material.css';
  @import "../node_modules/@syncfusion/ej2-vue-
calendars/styles/material.css";
  .wrapper {
    max-width: 250px;
    margin: 0 auto;
  }
</style>
```

{% previewsample "page.domainurl/code-snippet/datepicker/min-max-cs3" %}

See Also

- [Change the format of selected date](#)
- [Render DatePicker with specific culture](#)
- [How to achieve validation with DatePicker](#)

**Note:** You can also explore our [Vue DatePicker example](#) that shows you how to render the DatePicker in Vue.

## Getting Started with the Vue DatePicker Component in Vue 3

This article provides a step-by-step guide for setting up a [Vite](#) project with a JavaScript environment and integrating the Syncfusion Vue DatePicker component using the [Composition API](#) / [Options API](#).

The **Composition API** is a new feature introduced in Vue.js 3 that provides an alternative way to organize and reuse component logic. It allows developers to write components as functions that use smaller, reusable functions called composition functions to manage their properties and behavior.

The **Options API** is the traditional way of writing Vue.js components, where the component logic is organized into a series of options that define the component's properties and behavior. These options include data, methods, computed properties, watchers, lifecycle hooks, and more.

### Prerequisites

#### [System requirements for Syncfusion Vue UI components](#)

#### Set up the Vite project

A recommended approach for beginning with Vue is to scaffold a project using [Vite](#). To create a new Vite project, use one of the commands that are specific to either NPM or Yarn.

```
`bash
```

```
npm create vite@latest
```

```
,
```



or

```
`bash
```

```
yarn create vite
```

```
,
```

Using one of the above commands will lead you to set up additional configurations for the project as below:

1. Define the project name: We can specify the name of the project directly. Let's specify the name of the project as `my-project` for this article.

```
`bash
```

```
? Project name: » my-project
```

```
,
```

2. Select `Vue` as the framework. It will create a Vue 3 project.

```
`bash
```

```
? Select a framework: » - Use arrow-keys. Return to submit.
```

```
Vanilla
```

```
Vue
```

```
React
```

```
Preact
```

```
Lit
```

```
Svelte
```

```
Others
```

```
,
```

3. Choose `JavaScript` as the framework variant to build this Vite project using JavaScript and Vue.

```
`bash
```

```
? Select a variant: » - Use arrow-keys. Return to submit.
```

```
JavaScript
```

```
TypeScript
```

```
Customize with create-vue ↗
```

```
Nuxt ↗
```

```
,
```

4. Upon completing the aforementioned steps to create the `my-project`, run the following command to install its dependencies:

```
`bash
```

```
cd my-project
```

```
npm install
`
or
`bash
cd my-project
yarn install
`
```

Now that `my-project` is ready to run with default settings, let's add Syncfusion components to the project.

#### Add Syncfusion Vue packages

Syncfusion Vue component packages are available at [npmjs.com](https://www.npmjs.com). To use Syncfusion Vue components in the project, install the corresponding npm package.

This article uses the [Vue DatePicker component](#) as an example. To use the Vue DatePicker component in the project, the `@syncfusion/ej2-vue-calendars` package needs to be installed using the following command:

```
`bash
npm install @syncfusion/ej2-vue-calendars --save
`
or
`bash
yarn add @syncfusion/ej2-vue-calendars
`
```

#### Import Syncfusion CSS styles

You can import themes for the Syncfusion Vue component in various ways, such as using CSS or SASS styles from npm packages, CDN, [CRG](#) and [Theme Studio](#). Refer to [themes topic](#) to know more about built-in themes and different ways to refer to themes in a Vue project.

In this article, `Material` theme is applied using CSS styles, which are available in installed packages. The necessary `Material` CSS styles for the DatePicker component and its dependents were imported into the `<style>` section of `src/App.vue` file.

#### ~/SRC/APP.VUE

```
<style>
@import '../node_modules/@syncfusion/ej2-base/styles/material.css';
@import '../node_modules/@syncfusion/ej2-buttons/styles/material.css';
@import '../node_modules/@syncfusion/ej2-inputs/styles/material.css';
@import '../node_modules/@syncfusion/ej2-popups/styles/material.css';
@import "../node_modules/@syncfusion/ej2-vue-calendars/styles/material.css";
</style>
```

The order of importing CSS styles should be in line with its dependency graph.

### Add Syncfusion Vue component

Follow the below steps to add the Vue DatePicker component using **Composition API** or **Options API**:

1.First, import and register the DatePicker component in the **script** section of the **src/App.vue** file. If you are using the **Composition API**, you should add the **setup** attribute to the **script** tag to indicate that Vue will be using the **Composition API**.

#### **COMPOSITION API (~/SRC/APP.VUE)**

```
<script setup>
import { DatePickerComponent as EjsDatepicker } from '@syncfusion/ej2-vue-
calendars';
</script>
```

#### **OPTIONS API (~/SRC/APP.VUE)**

```
<script>
import { DatePickerComponent } from "@syncfusion/ej2-vue-calendars";
//Component registration
export default {
  name: "App",
  components: {
    'ejs-datepicker' : DatePickerComponent,
  }
}
</script>
```

2.In the **template** section, define the DatePicker component with the [dataSource](#) property and column definitions.

#### **~/SRC/APP.VUE**

```
<template>
<div class="control_wrapper">
<ejs-datepicker></ejs-datepicker>
</div>
</template>
```

Here is the summarized code for the above steps in the **src/App.vue** file:

#### **COMPOSITION API (~/SRC/APP.VUE)**

```
<template>
<div class="control_wrapper">
<ejs-datepicker></ejs-datepicker>
</div>
</template>
<script setup>
import { DatePickerComponent as EjsDatepicker } from '@syncfusion/ej2-vue-
calendars';
</script>
<style>
@import '../node_modules/@syncfusion/ej2-base/styles/material.css';
@import '../node_modules/@syncfusion/ej2-buttons/styles/material.css';
@import '../node_modules/@syncfusion/ej2-inputs/styles/material.css';
```

```
@import '../node_modules/@syncfusion/ej2-popups/styles/material.css';
@import "../node_modules/@syncfusion/ej2-vue-calendars/styles/material.css";
.control_wrapper {
max-width: 250px;
margin: 0 auto;
}
</style>
```

### OPTIONS API (~SRC/APP.VUE)

```
<template>
<div class="control_wrapper">
<ejs-datepicker></ejs-datepicker>
</div>
</template>
<script>
import { DatePickerComponent } from "@syncfusion/ej2-vue-calendars";
//Component registration
export default {
name: 'App',
components: {
"ejs-datepicker": DatePickerComponent
},
}
</script>
<style>
@import '../node_modules/@syncfusion/ej2-base/styles/material.css';
@import '../node_modules/@syncfusion/ej2-buttons/styles/material.css';
@import '../node_modules/@syncfusion/ej2-inputs/styles/material.css';
@import '../node_modules/@syncfusion/ej2-popups/styles/material.css';
@import "../node_modules/@syncfusion/ej2-vue-calendars/styles/material.css";
.control_wrapper {
max-width: 250px;
margin: 0 auto;
}
</style>
```

### Run the project

To run the project, use the following command:

```
`bash
```

```
npm run dev
```

```
,
```

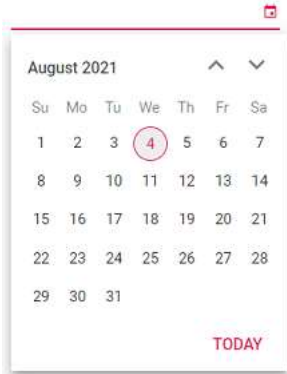
or

```
`bash
```

```
yarn run dev
```

```
,
```

The output will appear as follows:



### Setting the value, min and max dates

The following example demonstrates how to set the value, min and max dates on initializing the DatePicker. Here you can able to select a date within a range from 9th to 15th in a month of May 2017. To know more about range restriction in DatePicker, please refer this [page](#).

#### COMPOSITION API (~SRC/APP.VUE)

```
<template>
<div id="app">
<div class='wrapper'>
<ejs-datepicker :min="data[0].minDate" :max="data[0].maxDate"
:value="data[0].dateVal" ></ejs-datepicker>
</div>
</div>
</template>
<script setup>
import { DatePickerComponent as EjsDatepicker } from '@syncfusion/ej2-vue-
calendars';
const data = [{minDate: new Date("05/04/2017"),
maxDate: new Date("05/16/2017"),
dateVal: new Date("05/10/2017")}];
</script>
<style>
@import '../node_modules/@syncfusion/ej2-base/styles/material.css';
@import '../node_modules/@syncfusion/ej2-buttons/styles/material.css';
@import '../node_modules/@syncfusion/ej2-inputs/styles/material.css';
@import '../node_modules/@syncfusion/ej2-popups/styles/material.css';
@import "../node_modules/@syncfusion/ej2-vue-calendars/styles/material.css";
.control_wrapper {
max-width: 250px;
margin: 0 auto;
}
</style>
```

#### OPTIONS API (~SRC/APP.VUE)

```
<template>
<div id="app">
<div class='wrapper'>
<ejs-datepicker :min="minDate" :max="maxDate" :value="dateVal" ></ejs-
datepicker>
</div>
</div>
```

```

</template>
<script>
import { DatePickerComponent } from "@syncfusion/ej2-vue-calendars";
//Component registration
export default {
  name: 'App',
  components: {
    "ejs-datepicker": DatePickerComponent
  },
  data () {
    return {
      minDate : new Date("05/09/2017"),
      maxDate : new Date("05/15/2017"),
      dateVal : new Date("05/11/2017")
    }
  }
}
</script>
<style>
@import '../node_modules/@syncfusion/ej2-base/styles/material.css';
@import '../node_modules/@syncfusion/ej2-buttons/styles/material.css';
@import '../node_modules/@syncfusion/ej2-inputs/styles/material.css';
@import '../node_modules/@syncfusion/ej2-popups/styles/material.css';
@import "../node_modules/@syncfusion/ej2-vue-calendars/styles/material.css";
.wrapper {
  max-width: 250px;
  margin: 0 auto;
}
</style>

```

The output will appear as follows:



See Also

- [Change the format of selected date](#)
- [Render DatePicker with specific culture](#)
- [How to achieve validation with DatePicker](#)

### Date range in Vue Datepicker component

DatePicker provides an option to select a date value within a specified range by using the [min](#) and [max](#) properties. Always the min value has to be lesser than the max value.

When the min and max properties are configured and the selected date value is out-of-range or invalid, then the model value will be set to **out of range** date value or **null** respectively with highlighted **error** class to indicates the date is out of range or invalid.

The value property depends on the min/max with respect to [strictMode](#) property.

The below example allows selecting a date within the range from 7th to 27th day in a month.

#### APP.VUE

```
<template>
  <div id="app">
    <div class='wrap'>
      <ejs-datepicker id='datepicker' :value='calVal' :min='minVal'
: max='maxVal'></ejs-datepicker>
    </div>
  </div>
</template>
<script>
import Vue from "vue";
import { DatePickerPlugin } from "@syncfusion/ej2-vue-calendars";
Vue.use(DatePickerPlugin);
export default {
  data () {
    var today = new Date();
    var currentYear = today.getFullYear();
    var currentMonth = today.getMonth();
    var currentDay = today.getDate();
    return {
      calVal: new Date(new Date().setDate(14)),
      minVal: new Date(currentYear, currentMonth, 7),
      maxVal: new Date(currentYear, currentMonth, 27),
    }
  }
}
</script>
<style>
  @import '../node_modules/@syncfusion/ej2-base/styles/material.css';
  @import '../node_modules/@syncfusion/ej2-buttons/styles/material.css';
  @import '../node_modules/@syncfusion/ej2-inputs/styles/material.css';
  @import '../node_modules/@syncfusion/ej2-popups/styles/material.css';
  @import "../node_modules/@syncfusion/ej2-vue-
calendars/styles/material.css";
  .wrap {
    margin: 35px auto;
    width: 240px;
  }
</style>
```

{% previewsample "page.domainurl/code-snippet/datepicker/min-max-cs2" %}

If the value of **min** or **max** properties changed through code behind, then you have to update the **value** property to set within the range.

#### Date format in Vue Datepicker component

Date format is a way of representing the date value in different string format in the textbox.

By default, the DatePicker's format is based on the culture. You can also set the own custom format by using the [format](#) property.

Once the date format property has been defined it will be common to all the cultures.

To know more about the date format standards, refer to the [Internationalization Date Format](#) section.

The following example demonstrates the DatePicker with the custom format (yyyy-MM-dd).

#### APP.VUE

```
<template>
  <div id="app">
    <div class='wrapper'>
      <ejs-datepicker :placeholder="waterMark" :value="dateVal"
      :format="dateFormat"></ejs-datepicker>
    </div>
  </div>
</template>
<script>
import Vue from 'vue';
import { DatePickerPlugin } from '@syncfusion/ej2-vue-calendars';
Vue.use(DatePickerPlugin);
export default {
  data () {
    return {
      waterMark : 'Select a date',
      dateVal : new Date(),
      dateFormat : 'yyyy-MM-dd'
    }
  }
}
</script>
<style>
@import '../node_modules/@syncfusion/ej2-base/styles/material.css';
@import '../node_modules/@syncfusion/ej2-buttons/styles/material.css';
@import '../node_modules/@syncfusion/ej2-inputs/styles/material.css';
@import '../node_modules/@syncfusion/ej2-popups/styles/material.css';
@import "../node_modules/@syncfusion/ej2-vue-calendars/styles/material.css";
.wrapper {
  max-width: 250px;
  margin: 0 auto;
}
</style>
```

{% previewsample "page.domainurl/code-snippet/datepicker/getting-started-cs2" %}

#### Date masking in Vue Datepicker component

DatePicker has `enableMask` property that provides the option to enable the built-in date masking support. Also, you must inject the MaskedDateTime module to enable the masking support.

#### APP.VUE

```
<template>
<div id="app">
  <div class='wrapper'>
```



```

        <ejs-datepicker id="datepicker" :enableMask="true"></ejs-
datepicker>
    </div>
</div>
</template>
<script>
import Vue from "vue";
import { DatePickerPlugin, DatePicker, MaskedDateTime } from
"@syncfusion/ej2-vue-calendars";
DatePicker.Inject(MaskedDateTime)
Vue.use(DatePickerPlugin);
export default Vue.extend({
    data: function() {
        return {
        };
    },
    provide: {
        datepicker: [MaskedDateTime]
    }
});
</script>
<style>
@import '../node_modules/@syncfusion/ej2-base/styles/material.css';
@import '../node_modules/@syncfusion/ej2-buttons/styles/material.css';
@import '../node_modules/@syncfusion/ej2-inputs/styles/material.css';
@import '../node_modules/@syncfusion/ej2-popups/styles/material.css';
@import "../node_modules/@syncfusion/ej2-vue-calendars/styles/material.css";
.wrapper {
    max-width: 250px;
    margin: 0 auto;
}
</style>

```

{% previewsample "page.domainurl/code-snippet/datepicker/mask-module-cs1" %}

The mask pattern is defined based on the provided date format to the component. If the format is not specified, the mask pattern is formed based on the default format of the current culture.

**| Keys | Actions |**

**| --- | --- |**

**| Up / Down arrows | To increment and decrement the selected portion of the date. |**

**| Left / Right arrows and Tab | To navigate the selection from one portion to next portion |**

The following example demonstrates default and custom format of DatePicker component with mask module.

#### APP.VUE

```

<template>
<div id="app">
    <div class='wrapper1'>
        <!-- Specifies the masked datepicker without format property. -->
        <ejs-datepicker id="datepicker" :enableMask='true' ></ejs-
datepicker>
    </div>

```

```

        <div class='wrapper2'>
            <!-- Specifies the masked datepicker with format property. -->
            <ejs-datepicker id="format" :enableMask='true'
:format='dateFormat'></ejs-datepicker>
        </div>
    </div>
</template>
<script>
import Vue from "vue";
import { DatePickerPlugin, DatePicker, MaskedDateTime } from
"@syncfusion/ej2-vue-calendars";
DatePicker.Inject(MaskedDateTime)
Vue.use(DatePickerPlugin);
export default Vue.extend({
    data: function() {
        return {
            dateFormat: 'M/d/yyyy'
        };
    },
    provide: {
        datepicker: [MaskedDateTime]
    }
});
</script>
<style>
@import '../node_modules/@syncfusion/ej2-base/styles/material.css';
@import '../node_modules/@syncfusion/ej2-buttons/styles/material.css';
@import '../node_modules/@syncfusion/ej2-inputs/styles/material.css';
@import '../node_modules/@syncfusion/ej2-popups/styles/material.css';
@import "../node_modules/@syncfusion/ej2-vue-calendars/styles/material.css";
#wrapper1{
    min-width: 250px;
    float: left;
    margin-left: 100px;
}
#wrapper2{
    min-width: 250px;
    float: right;
    margin-right: 100px;
}
</style>

```

{% previewsample "page.domainurl/code-snippet/datepicker/mask-module-cs2" %}

### Configure Mask Placeholder

You can change mask placeholder value through property `maskPlaceholder`. By default , it takes the full name of date and time co-ordinates such as `day`, `month`, `year`, `hour` etc.

While changing to a culture other than `English`, ensure that locale text for the concerned culture is loaded through load method of `L10n` class for mask placeholder values like below.

```
`ts
```

```
//Load the L10n from ej2-base
```

```
import { L10n } from '@syncfusion/ej2-base';
```

//load the locale object to set the localized mask placeholder value

```
L10n.load({
  'de': {
    'datepicker': { day: 'Tag', month: 'Monat', year: 'Jahr' }
  }
});
```

The following example demonstrates default and customized mask placeholder value.

### APP.VUE

```
<template>
<div id="app">
  <div class='wrapper1'>
    <!-- Specifies the masked datepicker without mask placeholder. -->
    <ejs-datepicker id="datepicker" :enableMask="true" ></ejs-
datepicker>
  </div>
  <div class='wrapper2'>
    <!-- Specifies the masked datepicker with mask placeholder. -->
    <ejs-datepicker id="placeholder" :enableMask="true"
:maskPlaceholder='maskPlaceholderValue'></ejs-datepicker>
  </div>
</div>
</template>
<script>
import Vue from "vue";
import { DatePickerPlugin, MaskedDateTime, DatePicker } from
"@syncfusion/ej2-vue-calendars";
DatePicker.Inject(MaskedDateTime)
Vue.use(DatePickerPlugin);
export default Vue.extend({
  data: function() {
    return {
      maskPlaceholderValue: {day: 'd', month: 'M', year: 'y'}
    }
  },
  provide: {
    datepicker: [MaskedDateTime]
  }
});
</script>
<style>
@import '../node_modules/@syncfusion/ej2-base/styles/material.css';
@import '../node_modules/@syncfusion/ej2-buttons/styles/material.css';
@import '../node_modules/@syncfusion/ej2-inputs/styles/material.css';
@import '../node_modules/@syncfusion/ej2-popups/styles/material.css';
@import "../node_modules/@syncfusion/ej2-vue-calendars/styles/material.css";
#wrapper1{
  min-width: 250px;
  float: left;
  margin-left: 100px;
}
```

```

}
#wrapper2{
  min-width: 250px;
  float: right;
  margin-right: 100px;
}
</style>

```

{% previewsample "page.domainurl/code-snippet/datepicker/mask-module-cs3" %}

### Globalization in Vue Datepicker component

Globalization is the combination of adapting the component to various languages by means of parsing and formatting the date or number [Internationalization](#) and also by adding cultural specific customizations and translating the text [localization](#)

By default, DatePicker date format, week and month names are specific to English culture. It utilizes the [Essential JavaScript 2 Internationalization](#) package to parse and format the date object based on the culture by using the official [UNICODE CLDR](#) JSON data and it allows to load the culture specific CLDR JSON data by using `loadCldr` method

The DatePicker component supports only the Gregorian type of calendar. All the Essential JS 2 component are specific to English cultur ('en-US'). If you want to go with the different culture other than English, follow the below steps.

- Install the `CLDR-Data` package by using the below command (it installs the CLDR JSON data). To know more about CLDR-Data refer the

[CLDR-Data](#) link.

,

`npm install cldr-data --save`

,

Once the package installed, you can find the culture specific JSON data under the location `/node_modules/cldr-data`.

- Now import the installed CLDR JSON data into the `app.vue` file.
- Now use the `loadCldr` method to load the culture specific CLDR JSON data from the installed location to `app.vue` file.
- DatePicker displayed **Sunday** as the first day of week based on default culture ("en-US"). If you want to display the DatePicker with loaded culture's first day of week, you need to import `weekdata.json` file from the `cldr-data/supplemental` as given in the code example.

`ts

`//Load the loadCldr from ej2-base`

`import { loadCldr } from '@syncfusion/ej2-base';`

`import * as numberingSystems from 'cldr-data/supplemental/numberingSystems.json';`

```
import * as gregorian from 'cldr-data/main/de/ca-gregorian.json';
import * as numbers from 'cldr-data/main/de/numbers.json';
import * as timeZoneNames from 'cldr-data/main/de/timeZoneNames.json';
import * as weekData from 'cldr-data/supplemental/weekdata.json'; // To load the culture based first day of week

loadCldr(numberingSystems, gregorian, numbers, timeZoneNames, weekData);
,
```

The **Localization** library allows you to localize default text content of the DatePicker. The DatePicker component has static text for **today** feature that can be changed to other cultures (Arabic, Deutsch, French, etc.) by defining the [locale](#) value and translation object.

Locale keywords | Text----- | -----today | Name of the button to choose Today date. placeholder | Hint to describe expected value in input element.

- Before changing the culture other than **English**, ensure that locale text for the concerned culture is loaded through **load** method of

[L10n](#) class.

```
`ts
//Load the L10n from ej2-base
import { L10n } from '@syncfusion/ej2-base';
//load the locale object to set the localized placeholder value
L10n.load({
  'de': {
    'datepicker': { placeholder: 'Wählen Sie ein Datum aus', today: 'heute' }
  }
});
,
```

- Set the culture by using the [locale](#) property. The following example demonstrates the DatePicker in **German** culture.

#### APP.VUE

```
<template>
  <div id="app">
    <div class='wrapper'>
      <ejs-datepicker id="datepicker" locale='de' ></ejs-datepicker>
    </div>
  </div>
</template>
<script>
```

```

import Vue from "vue";
import { loadCldr, L10n } from '@syncfusion/ej2-base';
import { DatePickerPlugin } from "@syncfusion/ej2-vue-calendars";
// Here we have referred local json files for preview purpose
import * as numberingSystems from './numberingSystems.json';
import * as gregorian from './ca-gregorian.json';
import * as numbers from './numbers.json';
import * as timeZoneNames from './timeZoneNames.json';
Vue.use(DatePickerPlugin);
loadCldr(numberingSystems, gregorian, numbers, timeZoneNames);
L10n.load({
  'de': {
    'datepicker': { placeholder: "Wählen Sie Datum und Uhrzeit",
      today: "heute" }
  }
});
export default {}
</script>
<style>
  @import '../node_modules/@syncfusion/ej2-base/styles/material.css';
  @import '../node_modules/@syncfusion/ej2-buttons/styles/material.css';
  @import '../node_modules/@syncfusion/ej2-inputs/styles/material.css';
  @import '../node_modules/@syncfusion/ej2-popups/styles/material.css';
  @import "../node_modules/@syncfusion/ej2-vue-
calendars/styles/material.css";
  .wrapper {
    margin: 0px auto;
    width: 240px;
  }
</style>

```

{% previewsample "page.domainurl/code-snippet/datepicker/locale-cs1" %}

### Right-To-Left

The DatePicker supports right-to-left functionality for languages like Arabic, Hebrew to displays the text in the right-to-left direction. Use

[enableRtl](#) property to set the RTL direction.

### APP.VUE

```

<template>
  <div id="app">
    <div class='wrapper'>
      <ejs-datepicker id="datepicker" :locale="locale" :enableRtl="rtl"
    ></ejs-datepicker>
    </div>
  </div>
</template>
<script>
import Vue from "vue";
import { loadCldr, L10n } from '@syncfusion/ej2-base';
import { DatePickerPlugin } from "@syncfusion/ej2-vue-calendars";
// Here we have referred local json files for preview purpose
import * as numberingSystems from './numberingSystems.json';
import * as gregorian from './ca-gregorian.json';

```

```

import * as numbers from './numbers.json';
import * as timeZoneNames from './timeZoneNames.json';
Vue.use(DatePickerPlugin);
loadCldr(numberingSystems, gregorian, numbers, timeZoneNames);
L10n.load({
  'he': {
    'datepicker': { placeholder: 'הזן תאריך',
      today: 'היום' }
  }
});
export default {
  data () {
    return {
      locale: "he",
      rtl: true
    }
  }
}
</script>
<style>
  @import '../node_modules/@syncfusion/ej2-base/styles/material.css';
  @import '../node_modules/@syncfusion/ej2-buttons/styles/material.css';
  @import '../node_modules/@syncfusion/ej2-inputs/styles/material.css';
  @import '../node_modules/@syncfusion/ej2-popups/styles/material.css';
  @import "../node_modules/@syncfusion/ej2-vue-
calendars/styles/material.css";
  .wrapper {
    margin: 0px auto;
    width: 240px;
  }
</style>

```

{% previewsample "page.domainurl/code-snippet/datepicker/rtl-cs1" %}

## Strict mode in Vue Datepicker component

### Enable Strict Mode

The [strictMode](#) is an act, that allows the user to enter only the valid date within the specified min/max range in textbox. If the date is invalid, then the component will stay with the previous value. Else, if the date is out of range, then the component will set the date to the min/max date.

The following example demonstrates the DatePicker in `strictMode` with min/max range of 5th to 25th in a month of May. Here, it allows to enter

only the valid date within the specified range. If you are trying to enter the out-of-range value as like 28th of May, then the value will set to the max date of 25th May. Since the value 28th is greater than to max value of 25th. Or else if you are trying to enter the invalid date, then the value will stay with the previous value.

### APP.VUE

```

<template>
  <div id="app">
    <div class='wrap'>
      <ejs-datepicker id='datepicker' :strictMode='strict'
        :value='dateVal' :min='minVal' :max='maxVal'></ejs-datepicker>
    </div>
  </div>

```

```

        </div>
    </div>
</template>
<script>
import Vue from "vue";
import { DatePickerPlugin } from "@syncfusion/ej2-vue-calendars";
Vue.use(DatePickerPlugin);
export default {
    data () {
        return {
            strict:true,
            dateVal: new Date('5/28/2017'),
            minVal: new Date('5/5/2017'),
            maxVal: new Date('5/25/2017')
        }
    }
}
</script>
<style>
    @import '../node_modules/@syncfusion/ej2-base/styles/material.css';
    @import '../node_modules/@syncfusion/ej2-buttons/styles/material.css';
    @import '../node_modules/@syncfusion/ej2-inputs/styles/material.css';
    @import '../node_modules/@syncfusion/ej2-popups/styles/material.css';
    @import "../node_modules/@syncfusion/ej2-vue-
calendars/styles/material.css";
    .wrap {
        margin: 35px auto;
        width: 240px;
    }
</style>

```

{% previewsample "page.domainurl/code-snippet/datepicker/strict-mode-cs1" %}

### Disable Strict Mode

By default, the DatePicker act in strictMode **false** state, that allows to enter the invalid or out-of-range date in textbox.

If the date is out-of-range or invalid, then the model value will be set to **out of range** date value or **null** respectively with highlighted **error** class to indicates the date is out of range or invalid.

The following example demonstrates the **strictMode** as **false**. Here, it allows to enter the valid or invalid value in textbox. If you are entering out-of-range or invalid date value, then the model value will be set to **out of range** date value or **null** respectively with highlighted **error** class to indicates the date is out of range or invalid.

The following example demonstrates the DatePicker with strictMode **false**.

### APP.VUE

```

<template>
    <div id="app">
        <div class='wrap'>
            <ejs-datepicker id='datepicker' :value='dateVal' :min='minVal'
:max='maxVal'></ejs-datepicker>
        </div>
    </div>

```



```

</template>
<script>
import Vue from "vue";
import { DatePickerPlugin } from "@syncfusion/ej2-vue-calendars";
Vue.use(DatePickerPlugin);
export default {
  data () {
    return {
      dateVal: new Date('5/28/2017'),
      minVal: new Date('5/5/2017'),
      maxVal: new Date('5/25/2017')
    }
  }
}
</script>
<style>
@import '../node_modules/@syncfusion/ej2-base/styles/material.css';
@import '../node_modules/@syncfusion/ej2-buttons/styles/material.css';
@import '../node_modules/@syncfusion/ej2-inputs/styles/material.css';
@import '../node_modules/@syncfusion/ej2-popups/styles/material.css';
@import "../node_modules/@syncfusion/ej2-vue-calendars/styles/material.css";
.wrap {
  margin: 35px auto;
  width: 240px;
}
</style>

```

{% previewsample "page.domainurl/code-snippet/datepicker/strict-mode-cs2" %}

### Customization in Vue Datepicker component

You can customize the entire appearance of the input element and Calendar by using custom [cssClass](#) property.

and also you can use the calendar's [renderDayCell](#) event to customize the appearance of the each day cell.

Below is the list of classes that provides flexible way to customize the DatePicker component.

Class Name	Description
---	---
e-date-wrapper	Applied to DatePicker wrapper
e-datepicker	Applied to the DatePicker element.
e-float-text	Applied to the floating label.
e-date-icon	Applied to the DatePicker icon.
e-popup-wrapper	Applied to DatePicker popup wrapper.
e-calendar	Applied to Calendar element.
e-header	Applied to Calendar header.
e-title	Applied to Calendar title.
e-icon-container	Applied to Calendar previous and next icon container.

- | e-prev | Applied to Calendar previous icon.|
- | e-next | Applied to Calendar next icon.|
- | e-weekend | Applied to Calendar weekend dates.|
- | e-other-month | Applied to Calendar other month dates.|
- | e-day | Applied to each day cell of the Calendar.|
- | e-selected | Applied to Calendar selected dates.|
- | e-disabled | Applied to Calendar disabled dates.|

The following example disables the weekends of every month using `renderDayCell` event. Here we have used the `e-disabled` class to highlight the disabled date.

#### APP.VUE

```
<template>
  <div id="app">
    <div class='wrap'>
      <ejs-datepicker id='datepicker' :renderDayCell="disableDate"
placeholder='Select a Date'></ejs-datepicker>
    </div>
  </div>
</template>
<script>
import Vue from "vue";
import { DatePickerPlugin } from "@syncfusion/ej2-vue-calendars";
Vue.use(DatePickerPlugin);
export default {
  methods: {
    disableDate: function(args) {
      if (args.date.getDay() === 0 || args.date.getDay() === 6) {
        args.isDisabled = true;
      }
    }
  }
}
</script>
<style>
  @import '../node_modules/@syncfusion/ej2-base/styles/material.css';
  @import '../node_modules/@syncfusion/ej2-buttons/styles/material.css';
  @import '../node_modules/@syncfusion/ej2-inputs/styles/material.css';
  @import '../node_modules/@syncfusion/ej2-popups/styles/material.css';
  @import "../node_modules/@syncfusion/ej2-vue-
calendars/styles/material.css";
  .wrap {
    margin: 35px auto;
    width: 240px;
  }
</style>
```

{% previewsample "page.domainurl/code-snippet/datepicker/min-max-cs1" %}

### Adding mandatory asterisk to placeholder and float label

You can add a mandatory asterisk(\*) to placeholder and float label using `<b>.e-input-group.e-control-wrapper.e-float-input .e-float-text::after</b>` class.

#### APP.VUE

```
<template>
  <div id="app">
    <div class='wrapper'>
      <ejs-datepicker :placeholder="waterMark"
      :floatLabelType="Auto"></ejs-datepicker>
    </div>
  </div>
</template>
<script>
import Vue from 'vue';
import { DatePickerPlugin } from '@syncfusion/ej2-vue-calendars';
Vue.use(DatePickerPlugin);
export default {
  data () {
    return {
      waterMark : 'Select a date'
    }
  }
}
</script>
<style>
@import '../node_modules/@syncfusion/ej2-base/styles/material.css';
@import '../node_modules/@syncfusion/ej2-buttons/styles/material.css';
@import '../node_modules/@syncfusion/ej2-inputs/styles/material.css';
@import '../node_modules/@syncfusion/ej2-popups/styles/material.css';
@import "../node_modules/@syncfusion/ej2-vue-calendars/styles/material.css";
.wrapper {
  max-width: 250px;
  margin: 0 auto;
}
.e-input-group.e-control-wrapper.e-float-input .e-float-text::after {
  content: '*';
  color: red;
}
</style>
```

{% previewsample "page.domainurl/code-snippet/datepicker/getting-started-cs1" %}

See Also

- [How to disable the DatePicker component](#)
- [How to set read-only for DatePicker](#)
- [How to customize the DatePicker day header](#)

### View in Vue Datepicker component

The [Vue DatePicker](#) has the following predefined views that provides a flexible way to navigate back and forth to select the date.

**| View | Description |****| --- | --- |****| month (default) | Displays the days in a month |****| year | Displays the months in a year |****| decade | Displays the years in a decade |****Start view**

You can use the **start** property to define the initial rendering view.

The following example demonstrates how to create a DatePicker with **decade** as initial rendering view.

**APP.VUE**

```
<template>
  <div id="app">
    <div class='wrap'>
      <ejs-datepicker id='datepicker' start='Decade'
placeholder='Select a date'></ejs-datepicker>
    </div>
  </div>
</template>
<script>
import Vue from "vue";
import { DatePickerPlugin } from "@syncfusion/ej2-vue-calendars";
Vue.use(DatePickerPlugin);
export default {
}
</script>
<style>
  @import '../node_modules/@syncfusion/ej2-base/styles/material.css';
  @import '../node_modules/@syncfusion/ej2-buttons/styles/material.css';
  @import '../node_modules/@syncfusion/ej2-inputs/styles/material.css';
  @import '../node_modules/@syncfusion/ej2-popups/styles/material.css';
  @import "../node_modules/@syncfusion/ej2-vue-
calendars/styles/material.css";
  .wrap {
    margin: 35px auto;
    width: 240px;
  }
</style>
```

{% previewsample "page.domainurl/code-snippet/datepicker/view-cs1" %}

**Depth view**

Define the **depth** property to control the view navigation.

Always the depth view has to be smaller than the start view, otherwise the view restriction will be not restricted.

The following example demonstrates how to create a DatePicker that allows users to select a month.

**APP.VUE**

```
<template>
```

```

<div id="app">
  <div class='wrap'>
    <ejs-datepicker id='datepicker' start='Decade' depth='Year'
placeholder='Select a date'></ejs-datepicker>
  </div>
</div>
</template>
<script>
import Vue from "vue";
import { DatePickerPlugin } from "@syncfusion/ej2-vue-calendars";
Vue.use(DatePickerPlugin);
export default {
}
</script>
<style>
  @import '../node_modules/@syncfusion/ej2-base/styles/material.css';
  @import '../node_modules/@syncfusion/ej2-buttons/styles/material.css';
  @import '../node_modules/@syncfusion/ej2-inputs/styles/material.css';
  @import '../node_modules/@syncfusion/ej2-popups/styles/material.css';
  @import "../node_modules/@syncfusion/ej2-vue-
calendars/styles/material.css";
  .wrap {
    margin: 35px auto;
    width: 240px;
  }
</style>

```

{% previewsample "page.domainurl/code-snippet/datepicker/depth-cs1" %}

### Accessibility in Vue Datepicker component

The DatePicker component followed the accessibility guidelines and standards, including [ADA, Section 508](#), [WCAG 2.2](#) standards, and [WCAG roles](#) that are commonly used to evaluate accessibility.

The accessibility compliance for the DatePicker component is outlined below.

| Accessibility Criteria | Compatibility |

| -- | -- |

| [WCAG 2.2 Support](#) |  |

| [Section 508 Support](#) |  |

| [Screen Reader Support](#) |  |

| [Right-To-Left Support](#) |  |

| [Color Contrast](#) |  |

| [Mobile Device Support](#) |  |

| [Keyboard Navigation Support](#) |  |

| [Accessibility Checker Validation](#) |  |

| [Axe-core Accessibility Validation](#) |  |

```
<style>
.post .post-content img {
display: inline-block;
margin: 0.5em 0;
}
</style>

<div> - All
features of the component meet the requirement.</div>

<div> - Some features of the component do not meet the requirement.</div>

<div> - The
component does not meet the requirement.</div>
```

### WAI-ARIA attributes

The Web accessibility defines a way to make web content and web applications more accessible to disabled people. It especially helps the dynamic content change and advanced user interface controls developed with Ajax, HTML, JavaScript, and related technologies.

DatePicker provides built-in compliance with the [WAI-ARIA](#) specifications. WAI-ARIA supports is achieved through the attributes like `aria-expanded`, `aria-disabled`, `aria-activedescendant` applied to the input element.

To know about the accessibility of Calendar refer to the Calendar's [Accessibility](#) section.

It helps to provide information about the widget for assistive technology to the disabled person in screen reader.

- **Aria-expanded:** attributes indicates the state of a collapsible element.
- **Aria-disabled:** attribute indicates the disabled state of this DatePicker component.
- **Aria-activedescendent:** attribute helps in managing the current active child of the DatePicker component.

### Keyboard Interaction

You can use the following keys to interact with the DatePicker.

The component implements the keyboard navigation support by following the [WAI-ARIA practices](#).

It supports the below list of shortcut keys.

#### Input Navigation

Before opening the popup, use the below list of keys to control the popup element.

| **Press** | **To do this** |

| --- | --- |

| **Alt + Down Arrow** | **Opens the popup.** |

| **Alt + Up Arrow** | **Closes the popup.** |

| **Esc** | **Closes the popup.** |

#### Calendar Navigation

Use the below list of keys to navigate the Calendar after the popup has opened.

| **Press** | **To do this** |

| --- | --- |

| **Upper Arrow** | **Focus the previous week date.** |

| **Down Arrow** | **Focus the next week date.** |

| **Left Arrow** | **Focus the previous date.** |

| **Right Arrow** | **Focus the next date.** |

| **Home** | **Focus the first date in the month.** |

| **End** | **Focus the last date in the month.** |

| **Page Up** | **Focus the same date in the previous month.** |

| **Page Down** | **Focus the same date in the next month.** |

| **Enter** | **Select the currently focused date.** |

| **Shift + Page Up** | **Focus the same date in the previous year.** |

| **Shift + Page Down** | **Focus the same date in the previous year.** |

| **Control + Upper Arrow** | **Moves into the inner level of view like month-year, year-decade** |

| **Control + Down Arrow** | **Moves out from the depth level view like decade-year, year-month** |

| **Control + Home** | **Focus the starting date in the current year.** |

| **Control + End** | **Focus the ending date in the current year.** |

To focus the DatePicker component use the **alt+t** keys.

#### APP.VUE

```
<template>
  <div id="app">
    <div class='wrap'>
      <ejs-datepicker id='date' placeholder='Select Date'
        ref="DateInstance"></ejs-datepicker>
    </div>
  </div>
</template>
```

```

<script>
import Vue from "vue";
import { DatePickerPlugin } from "@syncfusion/ej2-vue-calendars";
Vue.use(DatePickerPlugin);
export default {
  mounted () {
    let DateObj = this.$refs.DateInstance;
    document.onkeyup = function (e) {
      if (e.altKey && e.keyCode === 84) {
        // press alt+t to focus the control.
        DateObj.$el.focus();
      }
    }
  }
}
</script>
<style>
@import '../node_modules/@syncfusion/ej2-base/styles/material.css';
@import '../node_modules/@syncfusion/ej2-buttons/styles/material.css';
@import '../node_modules/@syncfusion/ej2-inputs/styles/material.css';
@import '../node_modules/@syncfusion/ej2-popups/styles/material.css';
@import "@syncfusion/ej2-vue-calendars/styles/material.css";
.wrap {
  margin: 35px auto;
  width: 240px;
}
</style>

```

{% previewsample "page.domainurl/code-snippet/datepicker/access-cs1" %}

### Ensuring accessibility

The DatePicker component's accessibility levels are ensured through an [accessibility-checker](#) and [axe-core](#) software tools during automated testing.

The accessibility compliance of the DatePicker component is shown in the following sample. Open the [sample](#) in a new window to evaluate the accessibility of the DatePicker component with accessibility tools.

{% previewsample "https://ej2.syncfusion.com/accessibility/date-picker.html" %}

See also

- [Accessibility in Syncfusion Vue components](#)

### Two way binding in Vue Datepicker component

Two-way binding can be achieved by using the `v-model` directive in Vue. In the following sample, when you change the value in one DatePicker component, v-model will automatically update the value in the other DatePicker.

The following example demonstrates how to set the `two-way-binding` in the DatePicker.

#### APP.VUE

```
<template>
```



```

<div id="app">
  <div id="wrapper1">
    <ejs-datepicker id="datepicker" v-model="date"></ejs-datepicker>
  </div>
  <div id="wrapper2">
    <ejs-datepicker id="datepicker1" v-model="date"></ejs-datepicker>
  </div>
</div>
</template>
<script>
import Vue from "vue";
import { DatePickerPlugin } from "@syncfusion/ej2-vue-calendars";
Vue.use(DatePickerPlugin);
export default {
  data: function() {
    return {
      date: new Date()
    };
  }
}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-buttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-calendars/styles/material.css";
#wrapper1{
  min-width: 250px;
  float: left;
  margin-left: 100px;
}
#wrapper2{
  min-width: 250px;
  float: right;
  margin-right: 100px;
}
</style>

```

{% previewsample "page.domainurl/code-snippet/datepicker/two-way-cs1" %}

### Style appearance in Vue DatePicker component

The following content provides the exact CSS structure that can be used to modify the control's appearance based on the user preference.

#### Customizing the appearance of DatePicker wrapper element

Use the following CSS to customize the appearance of wrapper element.

,

*/ To specify height and font size /*

```

.e-input-group input.e-input, .e-input-group.e-control-wrapper input.e-input {
height: 40px;
font-size: 20px;

```

```
}
,
```

### Customizing the DatePicker icon element

Use the following CSS to customize the DatePicker icon element

```
,
```

*/ To specify background color and font size /*

```
.e-input-group .e-input-group-icon:last-child, .e-input-group.e-control-wrapper .e-input-group-icon:last-child {
```

```
font-size: 12px;
```

```
background-color: darkgray;
```

```
}
```

```
,
```

### Customizing the Calendar popup of the DatePicker

Please check the below section, to customize the style and appearance of the Calendar component

#### [Customizing Calendar's style and appearance](#)

#### Full screen mode support in mobiles and tablets

The DatePicker component's full-screen mode feature enables users to view the component popup element in full-screen mode on mobile devices with improved visibility and a better user experience. It is important to mention that this feature is exclusively available for mobile and tablet devices in both landscape and portrait orientations. To activate the full screen mode within the DatePicker component, simply set the [fullScreenMode](#) API value to `true`. This action will extend the calendar element to occupy the entire screen on mobile devices.

```
`html
```

```
<template>
```

```
<div id="app">
```

```
<ejs-datepicker :fullScreenMode="mobileMode" ></ejs-datepicker>
```

```
</div>
```

```
</template>
```

```
<script>
```

```
import Vue from 'vue';
```

```
import { DatePickerPlugin } from '@syncfusion/ej2-vue-calendars';
```

```
Vue.use(DatePickerPlugin);
```

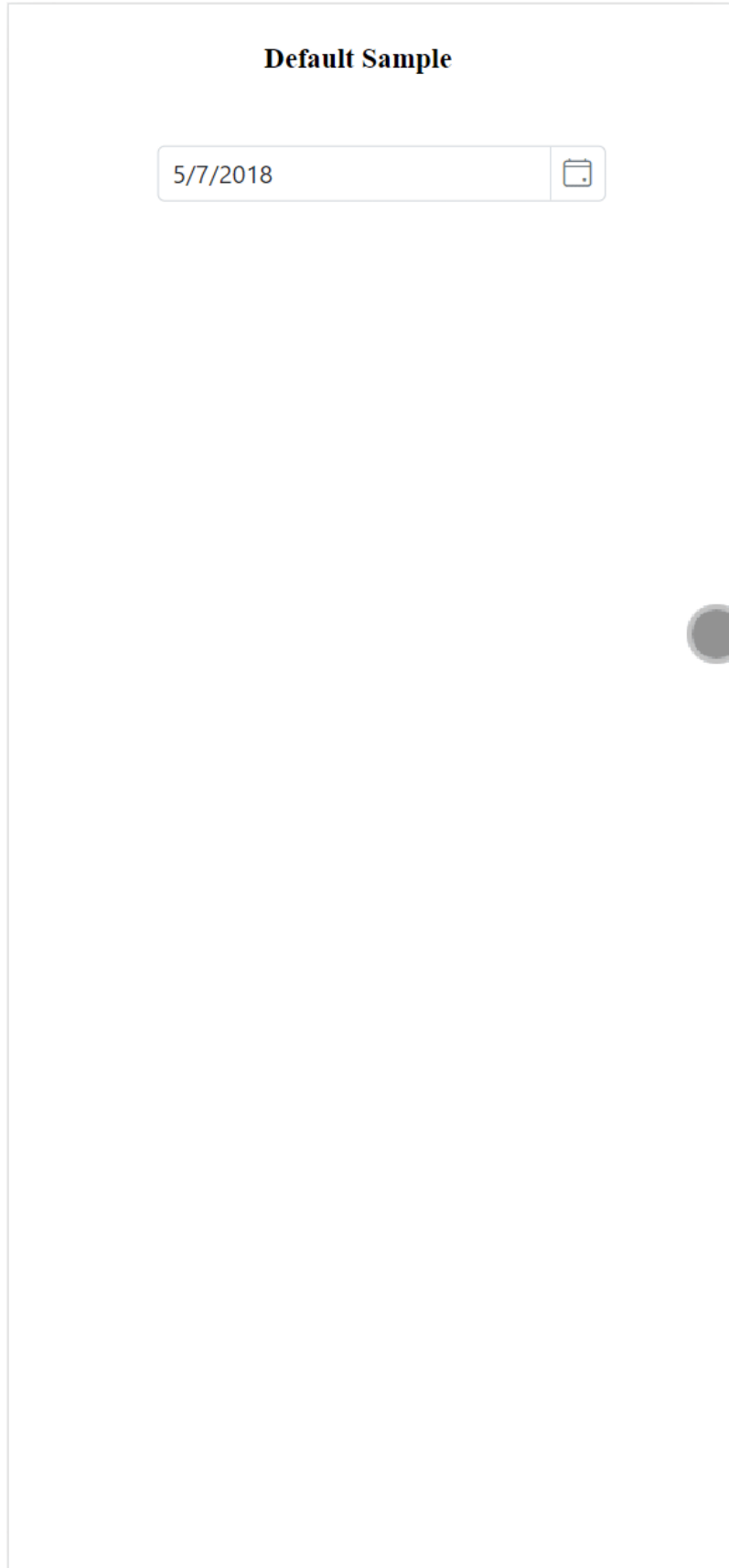
```
export default {
```

```
data () {
```

```
return {
```

```
mobileMode : true
```

```
}  
}  
}  
</script>  
,
```



## How To

Disabled the datepicker component in Vue Datepicker component

To disable the DatePicker, use its [enable](#) property.

The following example demonstrates the DatePicker in a disabled state.

### APP.VUE

```
<template>
  <div id="app">
    <div class='wrap'>
      <ejs-datepicker id='date' :enabled="enable"></ejs-datepicker>
    </div>
  </div>
</template>
<script>
import Vue from "vue";
import { DatePickerPlugin } from "@syncfusion/ej2-vue-calendars";
Vue.use(DatePickerPlugin);
export default {
  data () {
    return{
      enable: false
    }
  }
}
</script>
<style>
  @import '../node_modules/@syncfusion/ej2-base/styles/material.css';
  @import '../node_modules/@syncfusion/ej2-buttons/styles/material.css';
  @import '../node_modules/@syncfusion/ej2-inputs/styles/material.css';
  @import '../node_modules/@syncfusion/ej2-popups/styles/material.css';
  @import "../node_modules/@syncfusion/ej2-vue-calendars/styles/material.css";
  .wrap {
    margin: 35px auto;
    width: 240px;
  }
</style>
```

{% previewsample "page.domainurl/code-snippet/datepicker/access-cs2" %}

Set the placeholder in Vue Datepicker component

The following example demonstrates how to set **placeholder** in the DatePicker component.

Using **placeholder** you can display a short hint in the input element.

### APP.VUE

```
<template>
  <div id="app">
    <div class='wrap'>
      <ejs-datepicker id='date' placeholder="Select Date"></ejs-datepicker>
    </div>
  </div>
</template>
```

```

</template>
<script>
import Vue from "vue";
import { DatePickerPlugin } from "@syncfusion/ej2-vue-calendars";
Vue.use(DatePickerPlugin);
export default {
}
</script>
<style>
  @import '../node_modules/@syncfusion/ej2-base/styles/material.css';
  @import '../node_modules/@syncfusion/ej2-buttons/styles/material.css';
  @import '../node_modules/@syncfusion/ej2-inputs/styles/material.css';
  @import '../node_modules/@syncfusion/ej2-popups/styles/material.css';
  @import "../node_modules/@syncfusion/ej2-vue-
calendars/styles/material.css";
  .wrap {
    margin: 35px auto;
    width: 240px;
  }
</style>

```

{% previewsample "page.domainurl/code-snippet/datepicker/access-cs3" %}

### Set the readonly in Vue DatePicker component

The following example demonstrates how to set **readonly** in DatePicker component. You can achieve this by using **readonly** property.

#### APP.VUE

```

<template>
  <div id="app">
    <div class='wrap'>
      <ejs-datepicker id='date' placeholder="Select Date"
:readonly='read' :value='dateval'></ejs-datepicker>
    </div>
  </div>
</template>
<script>
import Vue from "vue";
import { DatePickerPlugin } from "@syncfusion/ej2-vue-calendars";
Vue.use(DatePickerPlugin);
export default {
  data () {
    return{
      dateval: new Date(),
      read: true
    }
  }
}
</script>
<style>
  @import '../node_modules/@syncfusion/ej2-base/styles/material.css';
  @import '../node_modules/@syncfusion/ej2-buttons/styles/material.css';
  @import '../node_modules/@syncfusion/ej2-inputs/styles/material.css';
  @import '../node_modules/@syncfusion/ej2-popups/styles/material.css';

```

```
@import "../node_modules/@syncfusion/ej2-vue-
calendars/styles/material.css";
.wrap {
  margin: 35px auto;
  width: 240px;
}
</style>
```

{% previewsample "page.domainurl/code-snippet/datepicker/access-cs4" %}

### Open datepicker popup on input click in Vue Datepicker component

You can open the DatePicker popup on input focus by calling the `show` method in the input `focus` event.

The following example demonstrates how to open the DatePicker popup when the input is focused.

#### APP.VUE

```
<template>
  <div id="app">
    <div class='wrap'>
      <ejs-datepicker ref="dateObj" id='date' placeholder="Choose a
date" :focus='onFocus'></ejs-datepicker>
    </div>
  </div>
</template>
<script>
import Vue from "vue";
import { DatePickerPlugin } from "@syncfusion/ej2-vue-calendars";
Vue.use(DatePickerPlugin);
export default {
  methods:{
    onFocus: function(args){
      this.$refs.dateObj.show();
    }
  }
}
</script>
<style>
  @import '../node_modules/@syncfusion/ej2-base/styles/material.css';
  @import '../node_modules/@syncfusion/ej2-buttons/styles/material.css';
  @import '../node_modules/@syncfusion/ej2-inputs/styles/material.css';
  @import '../node_modules/@syncfusion/ej2-popups/styles/material.css';
  @import "../node_modules/@syncfusion/ej2-vue-
calendars/styles/material.css";
  .wrap {
    margin: 35px auto;
    width: 240px;
  }
</style>
```

{% previewsample "page.domainurl/code-snippet/datepicker/open-popup-cs1" %}

### Prevent the popup close in Vue Datepicker component

To prevent the DatePicker popup from closing, use the `preventDefault` method from the `PreventableEventArgs`.

The following example demonstrates how to prevent the popup from closing.

#### APP.VUE

```
<template>
  <div id="app">
    <div class='wrap'>
      <ejs-datepicker id='date' placeholder="Select Date"
      :close='onClose' :value='dateval'></ejs-datepicker>
    </div>
  </div>
</template>
<script>
import Vue from "vue";
import { DatePickerPlugin } from "@syncfusion/ej2-vue-calendars";
Vue.use(DatePickerPlugin);
export default {
  methods:{
    onClose: function(args){
      args.preventDefault();
    }
  },
  data () {
    return{
      dateval: new Date()
    }
  }
}
</script>
<style>
  @import '../node_modules/@syncfusion/ej2-base/styles/material.css';
  @import '../node_modules/@syncfusion/ej2-buttons/styles/material.css';
  @import '../node_modules/@syncfusion/ej2-inputs/styles/material.css';
  @import '../node_modules/@syncfusion/ej2-popups/styles/material.css';
  @import "../node_modules/@syncfusion/ej2-vue-calendars/styles/material.css";
  .wrap {
    margin: 35px auto;
    width: 240px;
  }
</style>
```

{% previewsample "page.domainurl/code-snippet/datepicker/min-max-cs4" %}

#### Client side validation in Vue DatePicker component

To achieve the client side validation in a DatePicker component by using [Essential JavaScript 2 FormValidator](#). It provides an option to customize the feedback error messages to the corresponding fields to take action to resolve the issue.

In the below example, the required field validation is implemented by mapping the name attribute value to the rules property. It will validate the DatePicker component and display the validation message when the textbox value is empty during form post back or focus out.

#### APP.VUE

```
<template>
```



```

<div id="app">
  <div class='wrap'>
    <form id="form-element" class="form-horizontal">
      <div class="form-group" style="padding-top: 11px;">
        <div class="col-lg-8">
          <ejs-datepicker id="datepicker" name="date"
:change="onValueChanged" class="form-control" placeholder='Select a
date'></ejs-datepicker>
        </div>
      </div>
    </form>
  </div>
</div>
</template>
<script>
import Vue from "vue";
import { DatePickerPlugin } from "@syncfusion/ej2-vue-calendars";
import { FormValidator } from '@syncfusion/ej2-inputs';
Vue.use(DatePickerPlugin);
export default {
  data: function() {
    return {
      formObj: '',
      options : {
        customPlacement: function(inputelement, errorElement) {
          var parentElement = inputelement.closest('.form-group');
          parentElement.appendChild(errorElement);
        },
        rules: {
          'date': {
            required: true
          }
        }
      }
    },
    mounted: function () {
      this.formObj = new FormValidator('#form-element', this.options);
    },
    methods: {
      onValueChanged: function(args) {
        this.formObj.validate();
      }
    }
  }
}
</script>
<style>
@import '../node_modules/@syncfusion/ej2-base/styles/material.css';
@import '../node_modules/@syncfusion/ej2-buttons/styles/material.css';
@import '../node_modules/@syncfusion/ej2-inputs/styles/material.css';
@import '../node_modules/@syncfusion/ej2-popups/styles/material.css';
@import "../node_modules/@syncfusion/ej2-vue-
calendars/styles/material.css";
.wrap {
  margin: 0 auto;
  width: 240px;
}

```

```
</style>
```

```
{% previewsample "page.domainurl/code-snippet/datepicker/getting-started-cs4" %}
```

Customize the datepicker day header in Vue Datepicker component

You can change the format of the day that to be displayed in header using [dayHeaderFormat](#) property.

By default, the format is **Short**.

You can find the possible formats on below.

Name	Description
Short	Sets the short format of day name (like Su ) in day header.
Narrow	Sets the single character of day name (like S ) in day header.
Abbreviated	Sets the min format of day name (like Sun ) in day header.
Wide	Sets the long format of day name (like Sunday ) in day header.

### APP.VUE

```
<template>
  <div id="container">
    <div id='datepicker'>
      <ejs-datepicker dayHeaderFormat="Short"
ref="datepickerObj"></ejs-datepicker>
    </div>
    <div id="format">
      <label id="custom-input-label">Header Format Types</label>
      <ejs-dropdownlist id="select" :fields = "fields" :index =
"currentIndex" :dataSource = "items" :popupHeight= "popupHeight" :change=
"formatHandler">
      </ejs-dropdownlist>
    </div>
  </div>
</template>
<script>
import Vue from "vue";
import { DatePickerPlugin } from "@syncfusion/ej2-vue-calendars";
import { DropDownListPlugin } from "@syncfusion/ej2-vue-dropdowns";
Vue.use(DatePickerPlugin);
Vue.use(DropDownListPlugin);
export default {
  data () {
    return {
      items: [
        { sizeVal: 'Short' , sizeTxt: 'Short' },
        { sizeVal: 'Narrow' , sizeTxt: 'Narrow' },
        { sizeVal: 'Abbreviated' , sizeTxt: 'Abbreviated' },
        { sizeVal: 'Wide' , sizeTxt: 'Wide' },
      ],
      fields: { text: 'sizeTxt', value: 'sizeVal' },
      popupHeight: 200,
      currentIndex: 0,
      formatHandler: (args) => {
```

```

        this.$refs.datepickerObj.dayHeaderFormat = args.value;
    }
}
}
}
</script>
<style>
    @import "../node_modules/@syncfusion/ej2-base/styles/material.css";
    @import "../node_modules/@syncfusion/ej2-buttons/styles/material.css";
    @import "../node_modules/@syncfusion/ej2-vue-
calendars/styles/material.css";
    @import "../node_modules/@syncfusion/ej2-vue-
dropdowns/styles/material.css";
    #container{
        width: 100%;
    }
    #datepicker {
        width: 25%;
        margin-left: 30px;
        margin-top: 30px;
        display: inline-flex;
    }
    #format {
        width: 180px;
        margin-left: 400px;
        display: inline-block;
    }
    #select {
        height: 30px;
        width: 150px;
        margin-top: 10px;
    }
    #custom-input-label{
        font-size: 15px;
        font-weight: bold
    }
</style>

```

{% previewsample "page.domainurl/code-snippet/datepicker/header-format-cs1" %}

## DateRangePicker

### Getting Started with the Vue Daterangepicker Component in Vue 2

This article provides a step-by-step guide for setting up a Vue 2 project using [Vue-CLI](#) and integrating the Syncfusion Vue Daterangepicker component using the [Composition API](#) / [Options API](#).

#### Prerequisites

[System requirements for Syncfusion Vue UI components](#)

#### Dependencies

The list of dependencies required to use the DateRangePicker component in your application is given below:

```
`javascript`
```

```
|-- @syncfusion/ej2-vue-calendars
|-- @syncfusion/ej2-base
|-- @syncfusion/ej2-data
|-- @syncfusion/ej2-vue-base
|-- @syncfusion/ej2-calendars
|-- @syncfusion/ej2-inputs
|-- @syncfusion/ej2-splitbuttons
|-- @syncfusion/ej2-lists
|-- @syncfusion/ej2-popups
|-- @syncfusion/ej2-buttons
`
```

### Setting up the Vue 2 project

To generate a Vue 2 project using Vue-CLI, use the [vue create](#) command. Follow these steps to install Vue CLI and create a new project:

```
`bash
npm install -g @vue/cli
vue create quickstart
cd quickstart
npm run serve
`
```

or

```
`bash
yarn global add @vue/cli
vue create quickstart
cd quickstart
yarn run serve
`
```

When creating a new project, choose the option **Default ([Vue 2] babel, eslint)** from the menu.

```
E:\samples>vue create quickstart

Vue CLI v5.0.8
? Please pick a preset:
  Default ([Vue 3] babel, eslint)
> Default ([Vue 2] babel, eslint)
  Manually select features
```

Once the `quickstart` project is set up with default settings, proceed to add Syncfusion components to the project.

### Add Syncfusion Vue packages

Syncfusion packages are available at [npmjs.com](https://www.npmjs.com). To use Vue components, install the required npm package.

This article uses the [Vue Daterangepicker component](#) as an example. Install the `@syncfusion/ej2-vue-calendars` package by running the following command:

```
`bash
```

```
npm install @syncfusion/ej2-vue-calendars --save
```

```
,
```

or

```
`bash
```

```
yarn add @syncfusion/ej2-vue-calendars
```

```
,
```

### Import Syncfusion CSS styles

You can import themes for the Syncfusion Vue component in various ways, such as using CSS or SASS styles from npm packages, CDN, [CRG](#) and [Theme Studio](#). Refer to [themes topic](#) to know more about built-in themes and different ways to refer to themes in a Vue project.

In this article, the `Material` theme is applied using CSS styles, which are available in installed packages. The necessary `Material` CSS styles for the Daterange component and its dependents were imported into the `<style>` section of `src/App.vue` file.

### ~/SRC/APP.VUE

```
<style>
@import '../node_modules/@syncfusion/ej2-base/styles/material.css';
@import '../node_modules/@syncfusion/ej2-buttons/styles/material.css';
@import '../node_modules/@syncfusion/ej2-inputs/styles/material.css';
@import '../node_modules/@syncfusion/ej2-popups/styles/material.css';
@import '../node_modules/@syncfusion/ej2-lists/styles/material.css';
@import "../node_modules/@syncfusion/ej2-vue-calendars/styles/material.css";
</style>
```

Note: If you want to refer the combined component styles, please make use of our [CRG](#) (Custom Resource Generator) in your application.

### Add Syncfusion Vue component

Follow the below steps to add the Vue Daterangepicker component using **Composition API** or **Options API**:

1\ First, import and register the Daterangepicker component in the **script** section of the **src/App.vue** file. If you are using the **Composition API**, you should add the **setup** attribute to the **script** tag to indicate that Vue will be using the **Composition API**.

#### **COMPOSITION API (~SRC/APP.VUE)**

```
<script>
import { DateRangePickerComponent as EjsDaterangepicker } from
 '@syncfusion/ej2-vue-calendars';
</script>
```

#### **OPTIONS API (~SRC/APP.VUE)**

```
<script>
import { DateRangePickerComponent } from '@syncfusion/ej2-vue-calendars';
export default {
  components: {
    'ejs-daterangepicker': DateRangePickerComponent
  }
}
</script>
```

2\ In the **template** section, define the Daterangepicker component with the [placeholder](#) property.

#### **~/SRC/APP.VUE**

```
<template>
<div id="app">
<div class='wrapper'>
<ejs-daterangepicker :placeholder="waterMark"></ejs-daterangepicker>
</div>
</div>
</template>
```

3\ Declare the value for the **placeholder** property in the **script** section.

#### **COMPOSITION API (~SRC/APP.VUE)**

```
<script setup>
const waterMark = 'Select a Range';
</script>
```

#### **OPTIONS API (~SRC/APP.VUE)**

```
<script>
data () {
  return {
    waterMark : 'Select a Range'
  }
}
```

```
</script>
```

Here is the summarized code for the above steps in the **src/App.vue** file:

### COMPOSITION API (~SRC/APP.VUE)

```
<template>
  <div id="app">
    <div class='wrapper'>
      <ejs-daterangepicker :placeholder="waterMark"></ejs-
daterangepicker>
    </div>
  </div>
</template>
<script setup>
import { DateRangePickerComponent as EjsDaterangepicker } from
'@syncfusion/ej2-vue-calendars';
const waterMark = 'Select a Range';
</script>
<style>
@import '../node_modules/@syncfusion/ej2-base/styles/material.css';
@import '../node_modules/@syncfusion/ej2-buttons/styles/material.css';
@import '../node_modules/@syncfusion/ej2-inputs/styles/material.css';
@import '../node_modules/@syncfusion/ej2-popups/styles/material.css';
@import '../node_modules/@syncfusion/ej2-lists/styles/material.css';
@import "../node_modules/@syncfusion/ej2-vue-calendars/styles/material.css";
.wrapper {
  max-width: 250px;
  margin: 0 auto;
}
</style>
```

### OPTIONS API (~SRC/APP.VUE)

```
<template>
  <div id="app">
    <div class='wrapper'>
      <ejs-daterangepicker :placeholder="waterMark"></ejs-
daterangepicker>
    </div>
  </div>
</template>
<script>
import { DateRangePickerComponent } from '@syncfusion/ej2-vue-calendars';
export default {
  components: {
    'ejs-daterangepicker': DateRangePickerComponent
  },
  data () {
    return {
      waterMark : 'Select a Range'
    }
  }
}
</script>
<style>
```

```

@import '../node_modules/@syncfusion/ej2-base/styles/material.css';
@import '../node_modules/@syncfusion/ej2-buttons/styles/material.css';
@import '../node_modules/@syncfusion/ej2-inputs/styles/material.css';
@import '../node_modules/@syncfusion/ej2-popups/styles/material.css';
@import '../node_modules/@syncfusion/ej2-lists/styles/material.css';
@import "../node_modules/@syncfusion/ej2-vue-calendars/styles/material.css";
.wrapper {
    max-width: 250px;
    margin: 0 auto;
}
</style>

```

### Run the project

To run the project, use the following command:

```
`bash
```

```
npm run serve
```

```
,
```

or

```
`bash
```

```
yarn run serve
```

```
,
```

```
{% previewsample "page.domainurl/code-snippet/daterangepicker/getting-started-cs5" %}
```

### Setting the start date and end date

The start and end date in a range can be defined with the help of startDate and endDate property. The following example demonstrates to set the start and end date on initializing the DateRangePicker. To know more about range restriction in DateRangePicker, please refer this [page](#).

### COMPOSITION API (~ / SRC / APP.VUE)

```

<template>
  <div id="app">
    <div class='wrapper'>
      <ejs-daterangepicker :startDate="startVal" :endDate="endVal"
:placeholder="waterMark"></ejs-daterangepicker>
    </div>
  </div>
</template>
<script setup>
import { DateRangePickerComponent as EjsDaterangepicker } from
'@syncfusion/ej2-vue-calendars';
const startVal = new Date("11/12/2019 12:00 PM");
const endVal = new Date("11/25/2019 5:00 PM");
const waterMark = 'Select a Range';

</script>
<style>
@import '../node_modules/@syncfusion/ej2-base/styles/material.css';
@import '../node_modules/@syncfusion/ej2-buttons/styles/material.css';
@import '../node_modules/@syncfusion/ej2-inputs/styles/material.css';

```



```
@import '../node_modules/@syncfusion/ej2-popups/styles/material.css';
@import '../node_modules/@syncfusion/ej2-lists/styles/material.css';
@import "../node_modules/@syncfusion/ej2-vue-calendars/styles/material.css";
.wrapper {
    max-width: 250px;
    margin: 0 auto;
}
</style>
```

### **OPTIONS API (~SRC/APP.VUE)**

```
<template>
  <div id="app">
    <div class='wrapper'>
      <ejs-daterangepicker :startDate="startVal" :endDate="endVal"
:placeholder="waterMark"></ejs-daterangepicker>
    </div>
  </div>
</template>
<script>
import { DateRangePickerComponent } from '@syncfusion/ej2-vue-calendars';
export default {
  components: {
    'ejs-daterangepicker': DateRangePickerComponent
  },
  data () {
    return {
      startVal : new Date("11/12/2019 12:00 PM"),
      endVal : new Date("11/25/2019 5:00 PM"),
      waterMark : 'Select a Range'
    }
  }
}
</script>
<style>
@import '../node_modules/@syncfusion/ej2-base/styles/material.css';
@import '../node_modules/@syncfusion/ej2-buttons/styles/material.css';
@import '../node_modules/@syncfusion/ej2-inputs/styles/material.css';
@import '../node_modules/@syncfusion/ej2-popups/styles/material.css';
@import '../node_modules/@syncfusion/ej2-lists/styles/material.css';
@import "../node_modules/@syncfusion/ej2-vue-calendars/styles/material.css";
.wrapper {
    max-width: 250px;
    margin: 0 auto;
}
</style>
```

{% previewsample "page.domainurl/code-snippet/daterangepicker/min-max-cs1" %}

You can refer to our [Vue DateRangePicker](#) feature tour page for its groundbreaking feature representations. You can also explore our [Vue DateRangePicker example](#) that shows how to render the DateRangePicker in Vue.

## See Also

- [Render DateRangePicker with pre-defined ranges](#)
- [Render DateRangePicker with specific culture](#)

## Getting Started with the Vue DateRangePicker Component in Vue 3

This article provides a step-by-step guide for setting up a [Vite](#) project with a JavaScript environment and integrating the Syncfusion Vue DateRangePicker component using the [Composition API](#) / [Options API](#).

The **Composition API** is a new feature introduced in Vue.js 3 that provides an alternative way to organize and reuse component logic. It allows developers to write components as functions that use smaller, reusable functions called composition functions to manage their properties and behavior.

The **Options API** is the traditional way of writing Vue.js components, where the component logic is organized into a series of options that define the component's properties and behavior. These options include data, methods, computed properties, watchers, lifecycle hooks, and more.

### Prerequisites

#### [System requirements for Syncfusion Vue UI components](#)

#### Set up the Vite project

A recommended approach for beginning with Vue is to scaffold a project using [Vite](#). To create a new Vite project, use one of the commands that are specific to either NPM or Yarn.

```
`bash
npm create vite@latest
```

,

or

```
`bash
yarn create vite
```

,

Using one of the above commands will lead you to set up additional configurations for the project as below:

1. Define the project name: We can specify the name of the project directly. Let's specify the name of the project as **my-project** for this article.

```
`bash
? Project name: » my-project
```

,

2. Select **Vue** as the framework. It will create a Vue 3 project.

```
`bash
? Select a framework: » - Use arrow-keys. Return to submit.
```

Vanilla

Vue

React

Preact

Lit

Svelte

Others

,

3. Choose **JavaScript** as the framework variant to build this Vite project using JavaScript and Vue.

```
`bash
```

? Select a variant: » - Use arrow-keys. Return to submit.

JavaScript

TypeScript

Customize with create-vue ↗

Nuxt ↗

,

4. Upon completing the aforementioned steps to create the **my-project**, run the following command to install its dependencies:

```
`bash
```

```
cd my-project
```

```
npm install
```

,

or

```
`bash
```

```
cd my-project
```

```
yarn install
```

,

Now that **my-project** is ready to run with default settings, let's add Syncfusion components to the project.

#### Add Syncfusion Vue packages

Syncfusion Vue component packages are available at [npmjs.com](https://www.npmjs.com). To use Syncfusion Vue components in the project, install the corresponding npm package.

This article uses the [Vue DateRangePicker component](#) as an example. To use the Vue DateRangePicker component in the project, the **@syncfusion/ej2-vue-calendars** package needs to be installed using the following command:

```
`bash
```

```
npm install @syncfusion/ej2-vue-calendars --save
```

```
,
```

```
or
```

```
`bash
```

```
yarn add @syncfusion/ej2-vue-calendars
```

```
,
```

### Import Syncfusion CSS styles

You can import themes for the Syncfusion Vue component in various ways, such as using CSS or SASS styles from npm packages, CDN, [CRG](#) and [Theme Studio](#). Refer to [themes topic](#) to know more about built-in themes and different ways to refer to themes in a Vue project.

In this article, **Material** theme is applied using CSS styles, which are available in installed packages. The necessary **Material** CSS styles for the DateRangePicker component and its dependents were imported into the `<style>` section of **src/App.vue** file.

#### ~/SRC/APP.VUE

```
<style>
@import '../node_modules/@syncfusion/ej2-base/styles/material.css';
@import '../node_modules/@syncfusion/ej2-buttons/styles/material.css';
@import '../node_modules/@syncfusion/ej2-inputs/styles/material.css';
@import '../node_modules/@syncfusion/ej2-popups/styles/material.css';
@import '../node_modules/@syncfusion/ej2-lists/styles/material.css';
@import "../node_modules/@syncfusion/ej2-vue-calendars/styles/material.css";
</style>
```

The order of importing CSS styles should be in line with its dependency graph.

### Add Syncfusion Vue component

Follow the below steps to add the Vue DateRangePicker component using **Composition API** or **Options API**:

1. First, import and register the DateRangePicker component in the **script** section of the **src/App.vue** file. If you are using the **Composition API**, you should add the **setup** attribute to the **script** tag to indicate that Vue will be using the **Composition API**.

#### COMPOSITION API (~/SRC/APP.VUE)

```
<script setup>
import { DateRangePickerComponent as EjsDaterangepicker } from
"@syncfusion/ej2-vue-calendars";
</script>
```

#### OPTIONS API (~/SRC/APP.VUE)

```
<script>
import { DateRangePickerComponent } from "@syncfusion/ej2-vue-calendars";
//Component registration
export default {
  name: "App",
```

```
components: {
  'ejs-daterangepicker' : DateRangePickerComponent,
}
}
</script>
```

2. In the **template** section, define the DateRangePicker component with the [dataSource](#) property and column definitions.

### ~/SRC/APP.VUE

```
<template>
<div class="control_wrapper">
<ejs-daterangepicker></ejs-daterangepicker>
</div>
</template>
```

Here is the summarized code for the above steps in the **src/App.vue** file:

### COMPOSITION API (~/SRC/APP.VUE)

```
<template>
<div class="control_wrapper">
<ejs-daterangepicker></ejs-daterangepicker>
</div>
</template>
<script setup>
import { DateRangePickerComponent as EjsDaterangepicker } from
"@syncfusion/ej2-vue-calendars";
</script>
<style>
@import '../node_modules/@syncfusion/ej2-base/styles/material.css';
@import '../node_modules/@syncfusion/ej2-buttons/styles/material.css';
@import '../node_modules/@syncfusion/ej2-inputs/styles/material.css';
@import '../node_modules/@syncfusion/ej2-popups/styles/material.css';
@import '../node_modules/@syncfusion/ej2-lists/styles/material.css';
@import "../node_modules/@syncfusion/ej2-vue-calendars/styles/material.css";
.control_wrapper {
max-width: 250px;
margin: 0 auto;
}
</style>
```

### OPTIONS API (~/SRC/APP.VUE)

```
<template>
<div class="control_wrapper">
<ejs-daterangepicker></ejs-daterangepicker>
</div>
</template>
<script>
import { DateRangePickerComponent } from "@syncfusion/ej2-vue-calendars";
//Component registration
export default {
name: 'App',
```

```
components: {  
  "ejs-daterangepicker": DateRangePickerComponent  
},  
}  
</script>  
<style>  
@import '../node_modules/@syncfusion/ej2-base/styles/material.css';  
@import '../node_modules/@syncfusion/ej2-buttons/styles/material.css';  
@import '../node_modules/@syncfusion/ej2-inputs/styles/material.css';  
@import '../node_modules/@syncfusion/ej2-popups/styles/material.css';  
@import '../node_modules/@syncfusion/ej2-lists/styles/material.css';  
@import "../node_modules/@syncfusion/ej2-vue-calendars/styles/material.css";  
.control_wrapper {  
  max-width: 250px;  
  margin: 0 auto;  
}  
</style>
```

### Run the project

To run the project, use the following command:

```
`bash
```

```
npm run dev
```

```
,
```

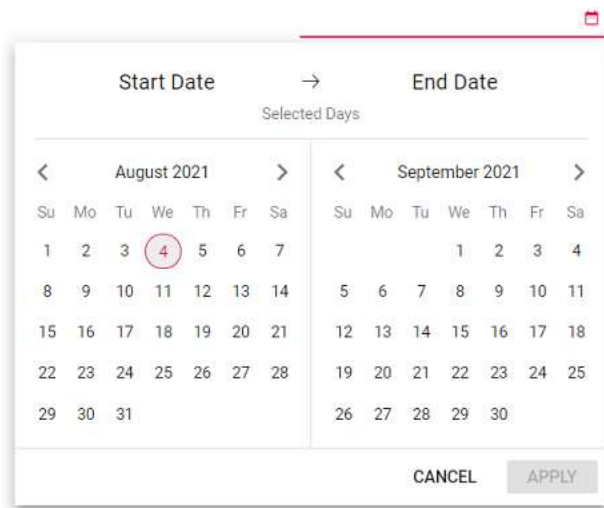
or

```
`bash
```

```
yarn run dev
```

```
,
```

The output will appear as follows:



### Setting the start date and end date

The start and end date in a range can be defined with the help of startDate and endDate property. The following example demonstrates to set the start and end date on initializing the DateRangePicker. To know more about range restriction in DateRangePicker, please refer this [page](#).

#### COMPOSITION API (~SRC/APP.VUE)

```
<template>
<div id="app">
<div class='wrapper'>
<ejs-daterangepicker :startDate="startVal" :endDate="endVal"
:placeholder="waterMark"></ejs-daterangepicker>
</div>
</div>
</template>
<script setup>
import { DateRangePickerComponent as EjsDaterangepicker } from
"@syncfusion/ej2-vue-calendars";
const startVal = new Date("11/12/2019 12:00 PM");
const endVal = new Date("11/25/2019 5:00 PM");
const waterMark = 'Select a Range';
</script>
<style>
@import '../node_modules/@syncfusion/ej2-base/styles/material.css';
@import '../node_modules/@syncfusion/ej2-buttons/styles/material.css';
@import '../node_modules/@syncfusion/ej2-inputs/styles/material.css';
@import '../node_modules/@syncfusion/ej2-popups/styles/material.css';
@import '../node_modules/@syncfusion/ej2-lists/styles/material.css';
@import "../node_modules/@syncfusion/ej2-vue-calendars/styles/material.css";
.control_wrapper {
max-width: 250px;
margin: 0 auto;
}
</style>
```

#### OPTIONS API (~SRC/APP.VUE)

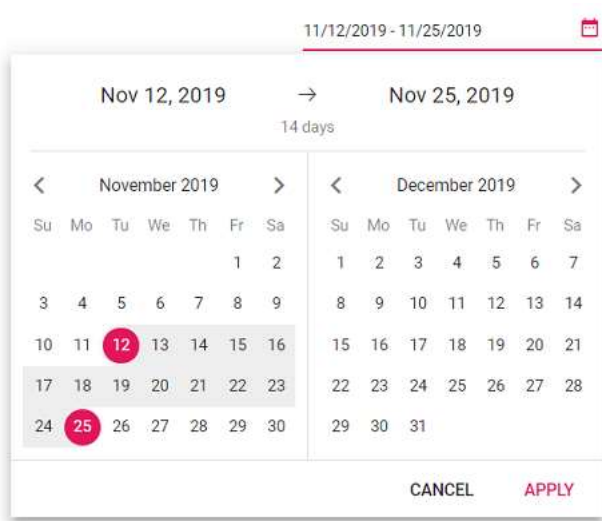
```
<template>
<div id="app">
<div class='wrapper'>
<ejs-daterangepicker :startDate="startVal" :endDate="endVal"
:placeholder="waterMark"></ejs-daterangepicker>
</div>
</div>
</template>
<script>
import { DateRangePickerComponent } from "@syncfusion/ej2-vue-calendars";
//Component registration
export default {
name: 'App',
components: {
"ejs-daterangepicker": DateRangePickerComponent
},
data () {
return {
startVal : new Date("11/12/2019 12:00 PM"),
```

```

endVal : new Date("11/25/2019 5:00 PM"),
waterMark : 'Select a Range'
}
}
}
</script>
<style>
@import '../node_modules/@syncfusion/ej2-base/styles/material.css';
@import '../node_modules/@syncfusion/ej2-buttons/styles/material.css';
@import '../node_modules/@syncfusion/ej2-inputs/styles/material.css';
@import '../node_modules/@syncfusion/ej2-popups/styles/material.css';
@import '../node_modules/@syncfusion/ej2-lists/styles/material.css';
@import "../node_modules/@syncfusion/ej2-vue-calendars/styles/material.css";
.wrapper {
max-width: 250px;
margin: 0 auto;
}
</style>

```

The output will appear as follows:



See Also

- [Render DateRangePicker with pre-defined ranges](#)
- [Render DateRangePicker with specific culture](#)

### Range restriction in Vue Daterangepicker component

Range selection in a DateRangePicker can be made-to-order with desire restrictions based on the application needs.

#### Restrict the range within a range

You can restrict the minimum and maximum date that can be allowed as start and end date in a range selection with the help of [min](#), [max](#) properties.

- **min** – sets the minimum date that can be selected as startDate.



- **max** – sets the maximum date that can be selected as endDate.

In the following sample, you can select a range from 15th day of this month to 15th day of next month.

#### APP.VUE

```
<template>
  <div id="app">
    <div class='wrapper'>
      <ejs-daterangepicker :min="minDate" :max="maxDate"
:placeholder="waterMark"></ejs-daterangepicker>
    </div>
  </div>
</template>
<script>
import Vue from 'vue';
import { DateRangePickerPlugin } from '@syncfusion/ej2-vue-calendars';
Vue.use(DateRangePickerPlugin);
export default {
  data () {
    return {
      waterMark: 'Select a Range',
      minDate : new Date(new Date().getFullYear(), new
Date().getMonth(), 15),
      maxDate : new Date(new Date().getFullYear(), new
Date().getMonth()+1, 15)
    }
  }
}
</script>
<style>
@import '../node_modules/@syncfusion/ej2-base/styles/material.css';
@import '../node_modules/@syncfusion/ej2-buttons/styles/material.css';
@import '../node_modules/@syncfusion/ej2-inputs/styles/material.css';
@import '../node_modules/@syncfusion/ej2-popups/styles/material.css';
@import '../node_modules/@syncfusion/ej2-lists/styles/material.css';
@import "../node_modules/@syncfusion/ej2-vue-calendars/styles/material.css";
.wrapper {
  max-width: 250px;
  margin: 0 auto;
}
</style>
```

{% previewsample "page.domainurl/code-snippet/daterangepicker/getting-started-cs10" %}

If the value of **min** or **max** property is changed through code behind, update the **start date** and **end date** properties to set within the range. Or else , if the **start** and **end** date is out of specified date range, a validation error class will be appended to the input element.

If **strictMode** is enabled, and both the start, end date is lesser than the min date then start and end date will be updated with **min** date. If both the start and end date is higher than the max date then start and end date will be updated with **max** date. Or else, if startDate is less than **min** date, startDate will be updated with **min** date or if endDate is greater than **max** date, endDate will be updated with the **max** date.

### Range span

Span between ranges can be limited to avoid excess or less days selection towards the required days in a range.

In this, minimum and maximum span allowed within the date range can be customized by [minDays](#) and [maxDays](#) properties.

- minDays- Sets the minimum number of days between start and end date.
- maxDays- Sets the maximum number of days between start and end date.

In the following sample, the range selection should be greater than 3 days and less than 8 days else it will not set.

### APP.VUE

```
<template>
  <div id="app">
    <div class='wrapper'>
      <ejs-daterangepicker :minDays="min" :maxDays="max"
:placeholder="waterMark"></ejs-daterangepicker>
    </div>
  </div>
</template>
<script>
import Vue from 'vue';
import { DateRangePickerPlugin } from '@syncfusion/ej2-vue-calendars';
Vue.use(DateRangePickerPlugin);
export default {
  data () {
    return {
      waterMark : 'Select a Range',
      min : 4,
      max : 8
    }
  }
}
</script>
<style>
@import '../node_modules/@syncfusion/ej2-base/styles/material.css';
@import '../node_modules/@syncfusion/ej2-buttons/styles/material.css';
@import '../node_modules/@syncfusion/ej2-inputs/styles/material.css';
@import '../node_modules/@syncfusion/ej2-popups/styles/material.css';
@import '../node_modules/@syncfusion/ej2-lists/styles/material.css';
@import "../node_modules/@syncfusion/ej2-vue-calendars/styles/material.css";
.wrapper {
  max-width: 250px;
  margin: 0 auto;
}
</style>
```

{% previewsample "page.domainurl/code-snippet/daterangepicker/getting-started-cs11" %}

### Strict mode

DateRangePicker provides an option to limit the user towards entering the valid date. With strict mode, you can set only the valid date.

If any invalid range is specified, the date range value resets to previous value. This restriction can be availed by enabling [strictMode](#) property as true.

### APP.VUE

```
<template>
  <div id="app">
    <div class='wrapper'>
      <ejs-daterangepicker :strictMode="mode"
:placeholder="waterMark"></ejs-daterangepicker>
    </div>
  </div>
</template>
<script>
import Vue from 'vue';
import { DateRangePickerPlugin } from '@syncfusion/ej2-vue-calendars';
Vue.use(DateRangePickerPlugin);
export default {
  data () {
    return {
      waterMark : 'Select a Range',
      mode : true
    }
  }
}
</script>
<style>
@import '../node_modules/@syncfusion/ej2-base/styles/material.css';
@import '../node_modules/@syncfusion/ej2-buttons/styles/material.css';
@import '../node_modules/@syncfusion/ej2-inputs/styles/material.css';
@import '../node_modules/@syncfusion/ej2-popups/styles/material.css';
@import '../node_modules/@syncfusion/ej2-lists/styles/material.css';
@import "../node_modules/@syncfusion/ej2-vue-calendars/styles/material.css";
.wrapper {
  max-width: 250px;
  margin: 0 auto;
}
</style>
```

{% previewsample "page.domainurl/code-snippet/daterangepicker/getting-started-cs12" %}

### Globalization in Vue Daterangepicker component

Globalization is the combination of internationalization and localization. You can adapt the component to various languages by parsing and formatting the date or number (Internationalization), and also add culture specific customization and translation to the text (Localization).

By default, DateRangePicker date format, week, and month names are specific to the English culture. It utilizes the [Essential JavaScript 2 Internationalization](#) package to parse and format the date object based on the culture by using the official [UNICODE CLDR](#) JSON data. It allows to load culture specific CLDR JSON data by using `loadCldr` method.

To use a different culture other than **English**, follow the below steps:

- Install the **CLDR-Data** package by using the below command (Installs the CLDR JSON data). To know more about CLDR-Data refer to the [CLDR-Data](#) link.

```
npm install cldr-data --save
```

Once the package is installed, you can find the culture specific JSON data under the location `/node_modules/cldr-data`.

- Import the installed CLDR JSON data into the `app.vue` file.
- Use the `loadCldr` method to load the culture specific CLDR JSON data from the installed location to `app.vue` file.
- DateRangePicker displayed **Sunday** as the first day of week based on default culture ("en-US"). If you want to display the DateRangePicker with loaded culture's first day of week, you need to import `weekdata.json` file from the `cldr-data/supplemental` as given in the code example.

```
`ts
```

```
//Load the loadCldr from ej2-base
```

```
import { loadCldr } from '@syncfusion/ej2-base';
```

```
import * as numberingSystems from 'cldr-data/supplemental/numberingSystems.json';
```

```
import * as gregorian from 'cldr-data/main/de/ca-gregorian.json';
```

```
import * as numbers from 'cldr-data/main/de/numbers.json';
```

```
import * as timeZoneNames from 'cldr-data/main/de/timeZoneNames.json';
```

```
import * as weekData from 'cldr-data/supplemental/weekdata.json'; // To load the culture based first day of week
```

```
loadCldr(numberingSystems, gregorian, numbers, timeZoneNames, weekData);
```

The **Localization** library allows you to localize default text content of the DateRangePicker. The DateRangePicker component has static text for **today** feature that can be changed to other cultures (Arabic, Deutsch, French, etc.) by defining the [locale](#) value and translation object.

Locale keywords | Text

placeholder | Hint to describe expected value in input element.

startLabel | Label to represent the start date.

endLabel | Label to represent the end date.

applyText | Text present in the apply button.

cancelText | Text present in the cancel button.

selectedDays | Text to represent selected days.

days | Text represents days.

customRange | Text present in the custom range button in presets container.

- Before changing to a culture other than **English**, ensure that locale text for the concerned culture is loaded through **load** method of

[L10n](#) class.

```
`ts
//Load the L10n, loadCldr from ej2-base
import { loadCldr, L10n } from '@syncfusion/ej2-base';
//load the locale object to set the localized placeholder value
L10n.load({
  'de': {
    'daterangepicker': { placeholder: 'Wählen Sie ein Datum aus' }
  }
});
`
```

- Set the culture by using the [locale](#) property.

The following example demonstrates the DateRangePicker in **German** culture.

#### **APP.VUE**

```
<template>
  <div id="app">
    <div class='wrapper'>
      <ejs-daterangepicker id="daterange" locale="de" ></ejs-
daterangepicker>
    </div>
  </div>
</template>
<script>
import Vue from 'vue';
import { loadCldr, L10n } from '@syncfusion/ej2-base';
import { DateRangePickerPlugin } from '@syncfusion/ej2-vue-calendars';
// Here we have referred local json files for preview purpose
import * as numberingSystems from './numberingSystems.json';
import * as gregorian from './ca-gregorian.json';
import * as numbers from './numbers.json';
import * as timeZoneNames from './timeZoneNames.json';
Vue.use(DateRangePickerPlugin);
loadCldr(numberingSystems, gregorian, numbers, timeZoneNames);
L10n.load({
  'de': {
```

```

    'daterangepicker': {
      placeholder: 'Wählen Sie einen Bereich aus',
      startLabel: 'Wählen Sie Startdatum',
      endLabel: 'Wählen Sie Enddatum',
      applyText: 'Sich bewerben',
      cancelText: 'Stornieren',
      selectedDays: 'Ausgewählte Tage',
      days: 'Tage',
      customRange: 'benutzerdefinierten Bereich'
    }
  });
export default {}
</script>
<style>
@import '../node_modules/@syncfusion/ej2-base/styles/material.css';
@import '../node_modules/@syncfusion/ej2-buttons/styles/material.css';
@import '../node_modules/@syncfusion/ej2-inputs/styles/material.css';
@import '../node_modules/@syncfusion/ej2-popups/styles/material.css';
@import '../node_modules/@syncfusion/ej2-lists/styles/material.css';
@import "../node_modules/@syncfusion/ej2-vue-calendars/styles/material.css";
.wrapper {
  max-width: 250px;
  margin: 0 auto;
}
</style>

```

{% previewsample "page.domainurl/code-snippet/daterangepicker/locale-cs1" %}

### Right-To-Left

The DateRangePicker supports RTL (right-to-left) functionality for languages like Arabic and Hebrew to displays the text in the right-to-left direction. Use [enableRtl](#) property to set the RTL direction. The following code example initialize the DateRangePicker component in **Hebrew** culture and also explains how to set the localized text to the placeholder using **load** method of

[L10n](#) class.

### APP.VUE

```

<template>
  <div id="app">
    <div class='wrapper'>
      <ejs-daterangepicker id="daterange" :locale="locale"
      :enableRtl="rtl" ></ejs-daterangepicker>
    </div>
  </div>
</template>
<script>
import Vue from 'vue';
import { loadCldr, L10n } from '@syncfusion/ej2-base';
import { DateRangePickerPlugin } from '@syncfusion/ej2-vue-calendars';
// Here we have referred local json files for preview purpose
import * as numberingSystems from './numberingSystems.json';
import * as gregorian from './ca-gregorian.json';
import * as numbers from './numbers.json';
import * as timeZoneNames from './timeZoneNames.json';

```

```

Vue.use(DateRangePickerPlugin);
loadCldr(numberingSystems, gregorian, numbers, timeZoneNames);
L10n.load({
  'he': {
    'daterangepicker': {
      placeholder: 'בחר טווח',
      startLabel: 'תווית התחלה',
      endLabel: 'סוף',
      applyText: 'להחיל טקסט',
      cancelText: 'בטל טקסט',
      selectedDays: 'ימים נבחרים',
      days: 'ימים',
      customRange: 'טווח מותאם אישית'
    }
  }
});
export default {
  data () {
    return {
      locale: "he",
      rtl: true
    }
  }
}
</script>
<style>
@import '../node_modules/@syncfusion/ej2-base/styles/material.css';
@import '../node_modules/@syncfusion/ej2-buttons/styles/material.css';
@import '../node_modules/@syncfusion/ej2-inputs/styles/material.css';
@import '../node_modules/@syncfusion/ej2-popups/styles/material.css';
@import '../node_modules/@syncfusion/ej2-lists/styles/material.css';
@import "../node_modules/@syncfusion/ej2-vue-calendars/styles/material.css";
.wrapper {
  max-width: 250px;
  margin: 0 auto;
}
</style>

```

{% previewsample "page.domainurl/code-snippet/daterangepicker/locale-rtl-cs1" %}

### Customize the date format

Representation of start and end date strings can be customized to required format by using [format](#) property.

By default, the format is based on the culture. To know more about the date format standards, refer to the Internationalization Date Format section. In the following sample, the date strings are formatted to `yyyy-MM-dd` and in between the string "to" is set as a [separator](#).

### APP.VUE

```

<template>
  <div id="app">
    <div class='wrapper'>
      <ejs-daterangepicker :format="dateFormat" :separator="seperate"
      :placeholder="waterMark"></ejs-daterangepicker>
    </div>
  </div>

```

```

    </div>
  </template>
  <script>
    import Vue from 'vue';
    import { DateRangePickerPlugin } from '@syncfusion/ej2-vue-calendars';
    Vue.use(DateRangePickerPlugin);
    export default {
      data () {
        return {
          waterMark : 'Select a Range',
          dateFormat : "yyyy-MM-dd",
          seperate : "to"
        }
      }
    }
  </script>
  <style>
    @import '../node_modules/@syncfusion/ej2-base/styles/material.css';
    @import '../node_modules/@syncfusion/ej2-buttons/styles/material.css';
    @import '../node_modules/@syncfusion/ej2-inputs/styles/material.css';
    @import '../node_modules/@syncfusion/ej2-popups/styles/material.css';
    @import '../node_modules/@syncfusion/ej2-lists/styles/material.css';
    @import "../node_modules/@syncfusion/ej2-vue-calendars/styles/material.css";
    .wrapper {
      max-width: 250px;
      margin: 0 auto;
    }
  </style>

```

{% previewsample "page.domainurl/code-snippet/daterangepicker/getting-started-cs6" %}

### Customization in Vue Daterangepicker component

The DateRangePicker is available for UI customization that can be achieved by using available properties and events in the component.

#### Day cell format

The DateRangePicker is available for UI customization based on your application requirements. It can be achieved by using [renderDayCell](#) event that provides an option to customize each day cell on rendering.

The following example disables the weekends of every month by using `renderDayCell` event.

#### APP.VUE

```

<template>
  <div id="app">
    <div class='wrapper'>
      <ejs-daterangepicker :placeholder="waterMark"
      :renderDayCell="disableDate" ></ejs-daterangepicker>
    </div>
  </div>
</template>
<script>
    import Vue from 'vue';
    import { DateRangePickerPlugin } from '@syncfusion/ej2-vue-calendars';
    Vue.use(DateRangePickerPlugin);
    export default {

```



```

data () {
  return {
    waterMark : 'Select a Range'
  }
},
methods: {
  disableDate: function(args) {
    if (args.date.getDay() === 0 || args.date.getDay() === 6) {
      args.isDisabled = true;
    }
  }
}
}
</script>
<style>
@import '../node_modules/@syncfusion/ej2-base/styles/material.css';
@import '../node_modules/@syncfusion/ej2-buttons/styles/material.css';
@import '../node_modules/@syncfusion/ej2-inputs/styles/material.css';
@import '../node_modules/@syncfusion/ej2-popups/styles/material.css';
@import '../node_modules/@syncfusion/ej2-lists/styles/material.css';
@import "../node_modules/@syncfusion/ej2-vue-calendars/styles/material.css";
.wrapper {
  max-width: 250px;
  margin: 0 auto;
}
</style>

```

{% previewsample "page.domainurl/code-snippet/daterangepicker/getting-started-cs2" %}

### Preset Ranges

DateRangePicker provides an option to set the predefined ranges via [presets](#) property with the corresponding label. This property will accept the values in the order of label, start date (date object), end date (date object), and append these ranges in the component for quick selection. In the following sample, you can easily choose the frequently used range options from the list of ranges.

### APP.VUE

```

<template>
  <div id="app">
    <div class='wrapper'>
      <ejs-daterangepicker :placeholder="waterMark">
        <e-presets>
          <e-preset label="This Week" :start='weekStart'
: end='weekEnd'></e-preset>
          <e-preset label="This Month" :start='monthStart'
: end='monthEnd'></e-preset>
          <e-preset label="Last Month" :start='lastStart'
: end='lastEnd'></e-preset>
          <e-preset label="Last Year" :start='yearStart'
: end='yearEnd'></e-preset>
        </e-presets>
      </ejs-daterangepicker>
    </div>
  </div>
</template>
</script>

```

```

import Vue from 'vue';
import { DateRangePickerPlugin } from '@syncfusion/ej2-vue-calendars';
Vue.use(DateRangePickerPlugin);
export default {
  data () {
    return {
      watermark : 'Selct a Range',
      weekStart : new Date(new Date(new Date().setDate(new
Date().getDate() - (new Date().getDay() + 7) % 7)).toDateString()),
      weekEnd : new Date(new Date(new Date().setDate(new Date(new
Date().setDate((new Date().getDate()
- (new Date().getDay() + 7) % 7))).getDate() +
6)).toDateString()),
      monthStart : new Date(new Date(new
Date().setDate(1)).toDateString()),
      monthEnd : new Date(new Date(new Date(new Date().setMonth(new
Date().getMonth() + 1)).setDate(0)).toDateString()),
      lastStart : new Date(new Date(new Date(new Date().setMonth(new
Date().getMonth() - 1)).setDate(1)).toDateString()),
      lastEnd : new Date(new Date(new
Date().setDate(0)).toDateString()),
      yearStart : new Date(new Date(new Date().getFullYear() - 1, 0,
1).toDateString()),
      yearEnd : new Date(new Date(new Date().getFullYear() - 1, 11,
31).toDateString())
    }
  }
}
</script>
<style>
@import '../node_modules/@syncfusion/ej2-base/styles/material.css';
@import '../node_modules/@syncfusion/ej2-buttons/styles/material.css';
@import '../node_modules/@syncfusion/ej2-inputs/styles/material.css';
@import '../node_modules/@syncfusion/ej2-popups/styles/material.css';
@import '../node_modules/@syncfusion/ej2-lists/styles/material.css';
@import "../node_modules/@syncfusion/ej2-vue-calendars/styles/material.css";
.wrapper {
  max-width: 250px;
  margin: 0 auto;
}
</style>

```

{% previewsample "page.domainurl/code-snippet/daterangepicker/getting-started-cs3" %}

### First day of week

Start day in a week will differ based on the culture, but you can also customize this based on the application needs. For this, you have to make use of [firstDayOfWeek](#) property. By default, first day of a week in en-US is Sunday. In the following example it is customized to Monday with the help of this property.

### APP.VUE

```

<template>
  <div id="app">
    <div class='wrapper'>

```

```

        <ejs-daterangepicker :firstDayOfWeek="week"
        :placeholder="waterMark"></ejs-daterangepicker>
      </div>
    </div>
  </template>
  <script>
    import Vue from 'vue';
    import { DateRangePickerPlugin } from '@syncfusion/ej2-vue-calendars';
    Vue.use(DateRangePickerPlugin);
    export default {
      data () {
        return {
          waterMark : 'Select a Range',
          week : 1
        }
      }
    }
  </script>
  <style>
    @import '../node_modules/@syncfusion/ej2-base/styles/material.css';
    @import '../node_modules/@syncfusion/ej2-buttons/styles/material.css';
    @import '../node_modules/@syncfusion/ej2-inputs/styles/material.css';
    @import '../node_modules/@syncfusion/ej2-popups/styles/material.css';
    @import '../node_modules/@syncfusion/ej2-lists/styles/material.css';
    @import "../node_modules/@syncfusion/ej2-vue-calendars/styles/material.css";
    .wrapper {
      max-width: 250px;
      margin: 0 auto;
    }
  </style>

```

{% previewsample "page.domainurl/code-snippet/daterangepicker/getting-started-cs4" %}

See Also

- [How to customize DateRangePicker using cssClass](#)
- [How to disable DateRangePicker component](#)
- [How to customize the DateRangePicker day header](#)

## Accessibility in Vue Daterangepicker component

The DateRangePicker component followed the accessibility guidelines and standards, including [ADA](#), [Section 508](#), [WCAG 2.2](#) standards, and [WCAG roles](#) that are commonly used to evaluate accessibility.

The accessibility compliance for the DateRangePicker component is outlined below.

| Accessibility Criteria | Compatibility |

| -- | -- |

| [WCAG 2.2 Support](#) |  |

| [Section 508 Support](#) |  |

| [Screen Reader Support](#) |  |

| [Right-To-Left Support](#) |  |

| [Color Contrast](#) |  |

| [Mobile Device Support](#) |  |

| [Keyboard Navigation Support](#) |  |

| [Accessibility Checker Validation](#) |  |

| [Axe-core Accessibility Validation](#) |  |

```
<style>
.post .post-content img {
display: inline-block;
margin: 0.5em 0;
}
</style>

<div> - All
features of the component meet the requirement.</div>

<div> - Some features of the component do not meet the requirement.</div>

<div> - The
component does not meet the requirement.</div>
```

### WAI-ARIA attributes

The web accessibility makes web content and web applications more accessible for people with disabilities. It especially helps in dynamic content change and development of advanced user interface controls with AJAX, HTML, JavaScript, and related technologies. DateRangePicker provides built-in compliance with [WAI-ARIA](#) specifications. WAI-ARIA support is achieved through the attributes like `aria-expanded`, `aria-disabled`, and `aria-activedescendant` applied as an input element.

To know about the accessibility of Calendar, refer to the Calendar's [Accessibility](#) section.

It helps people with disabilities by providing information about the widget for assistive technology in the screen readers. DateRangePicker component contains grid role and grid cell for each day cell.

- **Aria-expanded:** Indicates the currently selected date of the DateRangePicker component.
- **Aria-disabled:** Indicates the disabled state of the DateRangePicker component.

### Keyboard Interaction

Use the below keys to interact with the DateRangePicker. This component implements the keyboard navigation support by following the [WAI-ARIA practices](#).

It supports the following list of shortcut keys:

#### Input Navigation

Before opening the popup, use the following list of keys to control the popup element.

| **Press** | **To do this** |

| --- | --- |

| **Alt + Down Arrow** | Opens the popup. |

| **Alt + Up Arrow** | Closes the popup. |

| **Esc** | Closes the popup. |

#### Calendar Navigation

Use the following list of keys to navigate the currently focused Calendar after the popup has opened.

| **Press** | **To do this** |

| --- | --- |

| **Upper Arrow** | Focuses the same day of the previous week. |

| **Down Arrow** | Focuses the same day of the next week. |

| **Left Arrow** | Focuses the day before. |

| **Right Arrow** | Focuses the next day. |

| **Home** | Focuses the first day of the month. |

| **End** | Focuses the last day of the month. |

| **Page Up** | Focuses the same date of the previous month. |

| **Page Down** | Focuses the same date of the next month. |

| **Enter** | Selects the currently focused date. |

| **Shift + Page Up** | Focuses the same date for the previous year. |

| **Shift + Page Down** | Focuses the same date for the next year. |

| **Control + Home** | Focuses the first date of the current year. |

| **Control + End** | Focuses the last date of the current year. |

To focus the DateRangePicker component, use the **alt+t** keys.

### APP.VUE

```
<template>
  <div id="app">
    <div class='wrapper'>
```

```

    <ejs-daterangepicker placeholder="Select a Range"
    ref="DateInstance"></ejs-daterangepicker>
  </div>
</div>
</template>
<script>
import Vue from 'vue';
import { DateRangePickerPlugin } from '@syncfusion/ej2-vue-calendars';
Vue.use(DateRangePickerPlugin);
export default {
  mounted () {
    let DateObj = this.$refs.DateInstance;
    document.onkeyup = function (e) {
      if (e.altKey && e.keyCode === 84) {
        // press alt+t to focus the control.
        DateObj.$el.focus();
      }
    }
  }
}
</script>
<style>
@import '../node_modules/@syncfusion/ej2-base/styles/material.css';
@import '../node_modules/@syncfusion/ej2-buttons/styles/material.css';
@import '../node_modules/@syncfusion/ej2-inputs/styles/material.css';
@import '../node_modules/@syncfusion/ej2-popups/styles/material.css';
@import '../node_modules/@syncfusion/ej2-lists/styles/material.css';
@import "../node_modules/@syncfusion/ej2-vue-calendars/styles/material.css";
.wrapper {
  max-width: 250px;
  margin: 0 auto;
}
</style>

```

{% previewsample "page.domainurl/code-snippet/daterangepicker/getting-started-cs1" %}

### Ensuring accessibility

The DateRangePicker component's accessibility levels are ensured through an [accessibility-checker](#) and [axe-core](#) software tools during automated testing.

The accessibility compliance of the DateRangePicker component is shown in the following sample. Open the [sample](#) in a new window to evaluate the accessibility of the DateRangePicker component with accessibility tools.

{% previewsample "https://ej2.syncfusion.com/accessibility/date-range-picker.html" %}

See also

- [Accessibility in Syncfusion Vue components](#)

### Two way binding in Vue Daterangepicker component

Two-way binding can be achieved by using the `v-model` directive in Vue. In the following sample, when you change the value in one DateRangePicker component, v-model will automatically update the value in the other DateRangePicker.

The following example demonstrates how to set the **two-way-binding** in the DateRangePicker.

#### APP.VUE

```
<template>
<div id="app">
  <div id="wrapper1">
    <ejs-daterangepicker id="daterangepicker" :placeholder="waterMark"
v-model="date"></ejs-daterangepicker>
  </div>
  <div id="wrapper2">
    <ejs-daterangepicker id="daterangepicker1" :placeholder="waterMark"
v-model="date"></ejs-daterangepicker>
  </div>
</div>
</template>
<script>
import Vue from "vue";
import { DateRangePickerPlugin } from "@syncfusion/ej2-vue-calendars";
Vue.use(DateRangePickerPlugin);
export default {
  data: function() {
    return {
      waterMark: "Select a Range",
      date: null
    };
  }
}
</script>
<style>
  @import "../node_modules/@syncfusion/ej2-base/styles/material.css";
  @import "../node_modules/@syncfusion/ej2-buttons/styles/material.css";
  @import "../node_modules/@syncfusion/ej2-vue-
calendars/styles/material.css";
  #wrapper1{
    min-width: 250px;
    float: left;
    margin-left: 100px;
  }
  #wrapper2{
    min-width: 250px;
    float: right;
    margin-right: 100px;
  }
</style>
```

{% previewsample "page.domainurl/code-snippet/daterangepicker/two-way-cs1" %}

#### Style appearance in Vue Daterangepicker component

The following content provides the exact CSS structure that can be used to modify the control's appearance based on the user preference.

#### Customizing the appearance of DateRangePicker wrapper element

Use the following CSS to customize the appearance like height and font size of the wrapper element.

,

*/ To specify height and font size /*

```
.e-input-group input.e-input, .e-input-group.e-control-wrapper input.e-input {  
font-size: 20px;  
height: 40px;  
}  
,
```

#### Customizing the DateRangePicker icon element

Use the following CSS to customize the DateRangePicker icon element

,

*/ To specify background color and font size /*

```
.e-input-group .e-input-group-icon:last-child, .e-input-group.e-control-wrapper .e-input-group-icon:last-child {  
background-color: darkgray;  
font-size: 14px;  
}  
,
```

#### Customizing the DateRangePicker popup calendar header

Use the following CSS to customize the DateRangePicker popup calendar header

,

*/ To specify background and height /*

```
.e-daterangepicker.e-popup .e-range-header {  
background: beige;  
height: 80px;  
}  
,
```

#### Customizing the DateRangePicker popup calendar header title

Use the following CSS to customize the DateRangePicker popup calendar header title

,

*/ To specify color and font size /*

```
.e-daterangepicker.e-popup .e-range-header .e-start-label, .e-daterangepicker.e-popup .e-range-header .e-end-label {  
color: brown;  
font-size: 30px;  
}  
,
```



### Customizing the DateRangePicker popup calendar content

Use the following CSS to customize the DateRangePicker popup calendar content

,

*/ To specify background color /*

```
.e-daterangepicker.e-popup .e-calendar {  
background-color: brown;  
}
```

,

### Customizing the DateRangePicker popup calendar content title

Use the following CSS to customize the DateRangePicker popup calendar content title

,

*/ To specify color and font size /*

```
.e-daterangepicker.e-popup .e-calendar .e-header .e-title {  
color: beige;  
font-size: 20px;  
}
```

,

### Customizing the DateRangePicker popup calendar previous and next icon

Use the following CSS to customize the DateRangePicker popup calendar previous and next icon

,

*/ To specify font size /*

```
.e-calendar .e-header .e-prev, .e-calendar .e-header .e-next, .e-bigger.e-small .e-calendar .e-header .e-  
prev, .e-bigger.e-small .e-calendar .e-header .e-next {  
font-size: 20px;  
}
```

,

### Customizing the DateRangePicker popup calendar date cell grid on hovering

Use the following CSS to customize the DateRangePicker popup calendar date cell grid on hovering

,

*/ To specify background color and border /*

```
.e-calendar .e-content td:hover span.e-day {  
background-color: beige;  
border: 1px solid black;  
}
```

### Customizing the DateRangePicker popup calendar primary button in footer

Use the following CSS to customize the DateRangePicker popup calendar primary button in footer

*/ To specify background color and border color /*

```
.e-daterangepicker.e-popup .e-footer .e-btn.e-apply.e-flat.e-primary:disabled, .e-daterangepicker.e-popup .e-footer .e-btn.e-apply.e-flat.e-primary:disabled, .e-daterangepicker.e-popup .e-footer .e-css.e-btn.e-apply.e-flat.e-primary:disabled, .e-daterangepicker.e-popup .e-footer .e-css.e-btn.e-apply.e-flat.e-primary:disabled {
```

```
background-color: brown;
```

```
border-color: black;
```

```
}
```

### Customizing the DateRangePicker popup calendar cancel button in footer

Use the following CSS to customize the DateRangePicker popup calendar cancel button in footer

*/ To specify background color, color, and border color /*

```
.e-daterangepicker.e-popup .e-footer .e-btn.e-flat, .e-daterangepicker.e-popup .e-footer .e-css.e-btn.e-flat {
```

```
background-color: beige;
```

```
border-color: black;
```

```
color: maroon;
```

```
}
```

### Customizing the footer element in the DateRangePicker popup calendar

Use the following CSS to customize the DateRangePicker popup calendar footer element

*/ To specify background color, color, and border color /*

```
.e-daterangepicker.e-popup .e-footer {
```

```
background-color: beige;
```

```
height: 50px;
```

```
}
```

### Customizing the selected date cell grid in the DateRangePicker popup calendar

Use the following CSS to customize the selected date cell grid in the DateRangePicker popup calendar

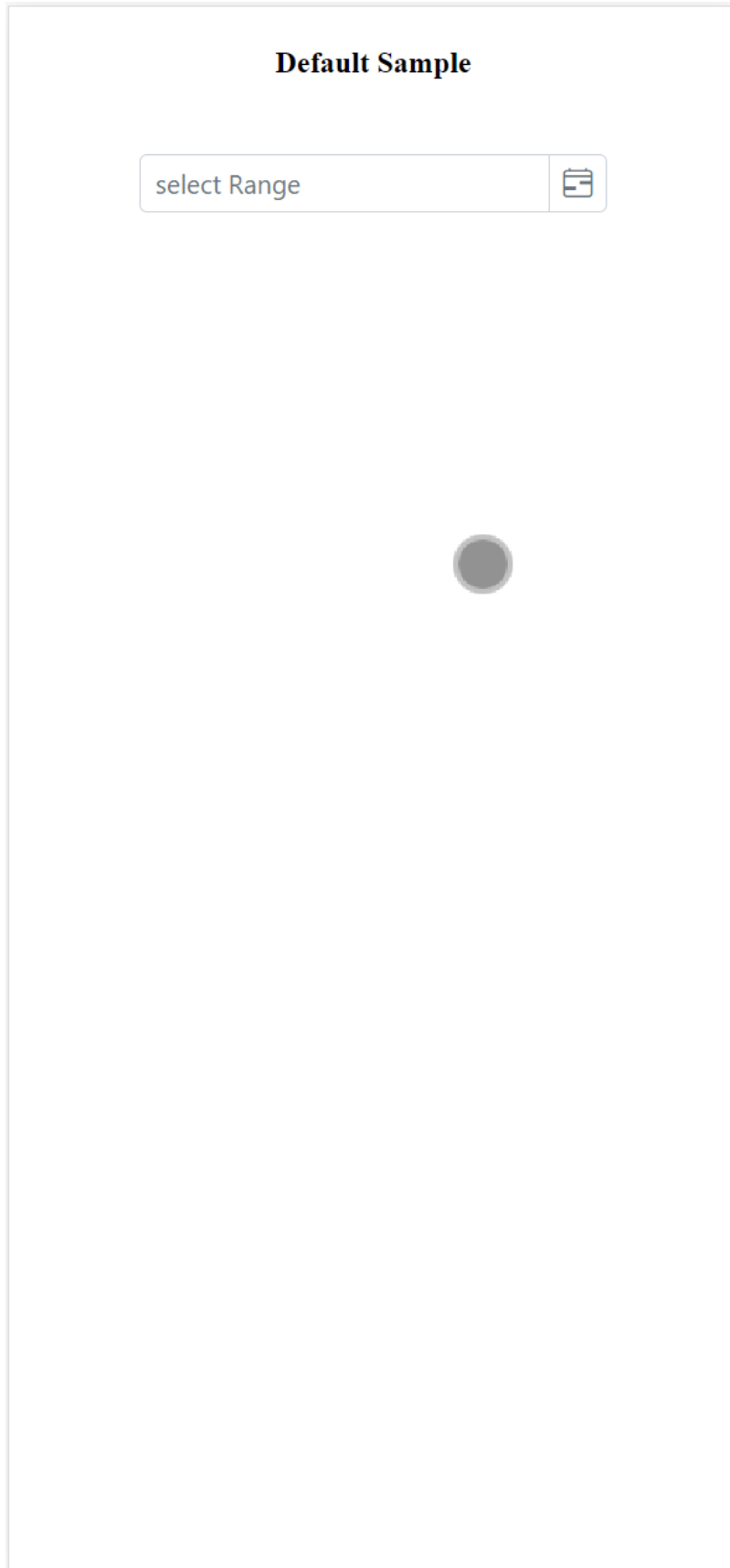
*/ To specify background and border /*

```
.e-calendar .e-content td.e-focused-date.e-today span.e-day {
background: lightgrey;
border: 1px solid black;
}
```

### Full screen mode support in mobiles and tablets

The DateRangePicker component's full-screen mode feature enables users to view the component popup element in full-screen mode on mobile devices with improved visibility and a better user experience. It is important to mention that this feature is exclusively available for mobile and tablet devices in both landscape and portrait orientations. To activate the full screen mode within the DateRangePicker component, simply set the [fullScreenMode](#) API value to **true**. This action will extend the calendar and presets popup element to occupy the entire screen on mobile devices.

```
`html
<template>
<div id="app">
<ejs-daterangepicker :fullScreenMode="mobileMode" ></ejs-daterangepicker>
</div>
</template>
<script>
import Vue from 'vue';
import { DateRangePickerPlugin } from '@syncfusion/ej2-vue-calendars';
Vue.use(DateRangePickerPlugin);
export default {
name: 'app',
data () {
return {
mobileMode : true
}
}
}
</script>
```



![DateRangePickerPresetsFullScreen](../images/DateRangePickerPresetsFullScreen.gif)

## How To

Disable the daterangepicker component in Vue Daterangepicker component

DateRangePicker can be inactivated on a page, by setting [enabled](#) value as false that will disable the component completely from all the user interactions including in form post. The following example demonstrates the disabled component.

### APP.VUE

```
<template>
  <div id="app">
    <div class='wrapper'>
      <ejs-daterangepicker :enabled="enable"
:placeholder="waterMark"></ejs-daterangepicker>
    </div>
  </div>
</template>
<script>
import Vue from 'vue';
import { DateRangePickerPlugin } from '@syncfusion/ej2-vue-calendars';
Vue.use(DateRangePickerPlugin);
export default {
  data () {
    return {
      waterMark: 'Select a Range',
      enable : false
    }
  }
}
</script>
<style>
@import '../node_modules/@syncfusion/ej2-base/styles/material.css';
@import '../node_modules/@syncfusion/ej2-buttons/styles/material.css';
@import '../node_modules/@syncfusion/ej2-inputs/styles/material.css';
@import '../node_modules/@syncfusion/ej2-popups/styles/material.css';
@import '../node_modules/@syncfusion/ej2-lists/styles/material.css';
@import "../node_modules/@syncfusion/ej2-vue-calendars/styles/material.css";
.wrapper {
  max-width: 250px;
  margin: 0 auto;
}
</style>
```

{% previewsample "page.domainurl/code-snippet/daterangepicker/getting-started-cs8" %}

Set the placeholder in Vue Daterangepicker component

The following example demonstrates how to set [placeholder](#) in the DateRangePicker control.

Using **placeholder** you can display a short hint in the input element.

### APP.VUE

```
<template>
  <div id="app">
    <div class='wrapper'>
```

```

        <ejs-daterangepicker :placeholder="waterMark"></ejs-daterangepicker>
      </div>
    </div>
  </template>
  <script>
import Vue from 'vue';
import { DateRangePickerPlugin } from '@syncfusion/ej2-vue-calendars';
Vue.use(DateRangePickerPlugin);
export default {
  data () {
    return {
      waterMark: 'Select a Range',
    }
  }
}
  </script>
  <style>
@import '../node_modules/@syncfusion/ej2-base/styles/material.css';
@import '../node_modules/@syncfusion/ej2-buttons/styles/material.css';
@import '../node_modules/@syncfusion/ej2-inputs/styles/material.css';
@import '../node_modules/@syncfusion/ej2-popups/styles/material.css';
@import '../node_modules/@syncfusion/ej2-lists/styles/material.css';
@import "../node_modules/@syncfusion/ej2-vue-calendars/styles/material.css";
.wrapper {
  max-width: 250px;
  margin: 0 auto;
}
  </style>

```

{% previewsample "page.domainurl/code-snippet/daterangepicker/getting-started-cs9" %}

### Customization using cssclass in Vue Daterangepicker component

To customize UI, you can make use of [cssClass](#) that will be added to the DateRangePicker component as the root CSS class. With this CSS class, you can override existing styles of DateRangePicker.

Following is the list of classes that provides flexible way to customize the DateRangePicker component.

Class Name	Description
---	---
e-date-range-wrapper	Applied to DateRangePicker wrapper.
e-range-icon	Applied to DateRangePicker icon.
e-popup	Applied to DateRangePicker popup wrapper.
e-calendar	Applied to both Calendar element.
e-right-calendar	Applied to right Calendar element.
e-left-calendar	Applied to left Calendar element.
e-start-label	Applied to start label in a popup.
e-end-calendar	Applied to end label in a popup.
e-day-span	Applied to day span details label in a popup.

- | e-footer | Applied to footer elements in a popup. |
- | e-apply | Applied to apply button in footer in a popup. |
- | e-cancel | Applied to cancel button in footer in a popup. |
- | e-header | Applied to Calendar header. |
- | e-title | Applied to Calendar title. |
- | e-icon-container | Applied to Calendar previous and next icon container. |
- | e-prev | Applied to Calendar previous icon. |
- | e-next | Applied to Calendar next icon. |
- | e-weekend | Applied to Calendar weekend dates. |
- | e-other-month | Applied to Calendar other month dates. |
- | e-day | Applied to each day cell of the Calendar. |
- | e-selected | Applied to Calendar selected dates. |
- | e-disabled | Applied to Calendar disabled dates. |

#### **APP.VUE**

```

<template>
  <div id="app">
    <div class='wrapper'>
      <ejs-daterangepicker :placeholder="waterMark"
      :cssClass="classVal"></ejs-daterangepicker>
    </div>
  </div>
</template>
<script>
import Vue from 'vue';
import { DateRangePickerPlugin } from '@syncfusion/ej2-vue-calendars';
Vue.use(DateRangePickerPlugin);
export default {
  data () {
    return {
      waterMark: "Select a Range",
      classVal: "customCSS"
    }
  }
}
</script>
<style>
@import '../node_modules/@syncfusion/ej2-base/styles/material.css';
@import '../node_modules/@syncfusion/ej2-buttons/styles/material.css';
@import '../node_modules/@syncfusion/ej2-inputs/styles/material.css';
@import '../node_modules/@syncfusion/ej2-popups/styles/material.css';
@import '../node_modules/@syncfusion/ej2-lists/styles/material.css';
@import "../node_modules/@syncfusion/ej2-vue-calendars/styles/material.css";
.wrapper {
  max-width: 250px;
  margin: 0 auto;
}

```

```

.customCSS .e-calendar .e-content .e-selected span.e-day, /* csslint allow:
adjoining-classes*/
.customCSS .e-calendar .e-content .e-selected span.e-day:hover, /* csslint
allow: adjoining-classes*/
.customCSS .e-calendar .e-content .e-today.e-selected:hover span.e-day, /*
csslint allow: adjoining-classes*/
.customCSS .e-calendar .e-content .e-today.e-selected span.e-day, /* csslint
allow: adjoining-classes*/
.customCSS .e-calendar .e-content .e-selected:hover span.e-day /* csslint
allow: adjoining-classes*/
{
background-color: #35b86b;
}
.customCSS .e-calendar .e-content .e-today span.e-day, /* csslint allow:
adjoining-classes*/
.customCSS .e-calendar .e-content .e-focused-date.e-today span.e-day { /*
csslint allow: adjoining-classes*/
border: 1px solid #35b86b;
color: #ff3337;
}
.customCSS .e-calendar .e-content .e-weekend span { /* csslint allow:
adjoining-classes*/
color: #ff3337;
}
.customCSS.e-date-range-wrapper .e-input-group-icon.e-icons.e-active, /*
csslint allow: adjoining-classes*/
.customCSS .e-btn.e-flat, /* csslint allow: adjoining-classes*/
.customCSS .e-btn.e-flat:hover { /* csslint allow: adjoining-classes*/
color: #35b86b;
}
</style>

```

{% previewsample "page.domainurl/code-snippet/daterangepicker/getting-started-cs7" %}

[Customize the daterangepicker day header in Vue Daterangepicker component](#)

You can change the format of the day that to be displayed in header using [dayHeaderFormat](#) property. By default, the format is **Short**.

You can find the possible formats on below.

Name	Description
Short	Sets the short format of day name (like Su ) in day header.
Narrow	Sets the single character of day name (like S ) in day header.
Abbreviated	Sets the min format of day name (like Sun ) in day header.
Wide	Sets the long format of day name (like Sunday ) in day header.

#### APP.VUE

```

<template>
  <div id="container">
    <div id='daterangepicker'>

```



```

        <ejs-daterangepicker cssClass="format-wide"
dayHeaderFormat="Short" ref="daterangepickerObj"></ejs-daterangepicker>
    </div>
    <div id="format">
        <label id="custom-input-label">Header Format Types</label>
        <ejs-dropdownlist id="select" :fields = "fields" :index =
"currentIndex" :dataSource = "items" :popupHeight= "popupHeight" :change=
"formatHandler">
        </ejs-dropdownlist>
    </div>
</div>
</template>
<script>
import Vue from "vue";
import { DateRangePickerPlugin } from "@syncfusion/ej2-vue-calendars";
import { DropDownListPlugin } from "@syncfusion/ej2-vue-dropdowns";
Vue.use(DateRangePickerPlugin);
Vue.use(DropDownListPlugin);
export default {
    data () {
        return {
            items: [
                { sizeVal: 'Short' , sizeTxt: 'Short' },
                { sizeVal: 'Narrow' , sizeTxt: 'Narrow' },
                { sizeVal: 'Abbreviated' , sizeTxt: 'Abbreviated' },
                { sizeVal: 'Wide' , sizeTxt: 'Wide' },
            ],
            fields: { text: 'sizeTxt', value: 'sizeVal' },
            popupHeight: 200,
            currentIndex: 0,
            formatHandler: (args) => {
                this.$refs.daterangepickerObj.dayHeaderFormat = args.value;
            }
        }
    }
}
</script>
<style>
    @import "../node_modules/@syncfusion/ej2-base/styles/material.css";
    @import "../node_modules/@syncfusion/ej2-buttons/styles/material.css";
    @import "../node_modules/@syncfusion/ej2-vue-
calendars/styles/material.css";
    @import "../node_modules/@syncfusion/ej2-vue-
dropdowns/styles/material.css";
    #container{
        width: 100%;
    }
    #daterangepicker {
        width: 25%;
        margin-left: 30px;
        margin-top: 30px;
        display: inline-flex;
    }
    .e-daterangepicker.e-daterange-day-header-lg.format-wide .e-calendar.e-
calendar-day-header-lg{
        min-width: 490px;
    }

```

```
.e-daterangepicker.e-daterange-day-header-lg.format-wide {
  max-width: 1000px;
  left: 0px !important;
}
#format {
  width: 180px;
  margin-left: 400px;
  display: inline-block;
}
#select {
  height: 30px;
  width: 150px;
  margin-top: 10px;
}
#custom-input-label{
  font-size: 15px;
  font-weight: bold
}
</style>
```

{% previewsample "page.domainurl/code-snippet/daterangepicker/header-format-cs1" %}

## DateTimePicker

### Getting Started

This section explains how to create a simple DateTimePicker and how to configure the DateTimePicker component.

#### Prerequisites

[System requirements for Syncfusion Vue UI components](#)

#### Dependencies

The list of dependencies required to use the DateTimePicker component in your application is given below:

```
`javascript
|-- @syncfusion/ej2-vue-calendars
|-- @syncfusion/ej2-base
|-- @syncfusion/ej2-data
|-- @syncfusion/ej2-vue-base
|-- @syncfusion/ej2-calendars
|-- @syncfusion/ej2-inputs
|-- @syncfusion/ej2-splitbuttons
|-- @syncfusion/ej2-lists
|-- @syncfusion/ej2-popups
|-- @syncfusion/ej2-buttons
,`
```

### Get Started with Vue CLI

You can use [Vue CLI](#) to setup your vue applications.

To install Vue CLI use the following command.

```
`bash
npm install -g @vue/cli
npm install -g @vue/cli-init
`
```

Start a new project using below Vue CLI command.

```
`bash
vue init webpack-simple quickstart
cd quickstart
npm install
`
```

### Adding Syncfusion packages

All the available Essential JS 2 packages are published in [npmjs.com](https://www.npmjs.com) registry.

You can choose the component that you want to install. For this application, we are going to use DateTimePicker component.

To install DateTimePicker component, use the following command

```
`bash
npm install @syncfusion/ej2-vue-calendars --save
`
```

### Registering Vue Component

For Registering Vue Component two ways are available. They are as follows.

- `Vue.use()`
- `Vue.component()`

#### Using `Vue.use()`

Import the Component Plugin from the EJ2 Vue Package and register the same using `Vue.use()` with Component Plugin as its argument.

Refer the code snippet given below.

```
`ts
import { DateTimePickerPlugin } from '@syncfusion/ej2-vue-calendars';
Vue.use(DateTimePickerPlugin);
`
```

By Registering Component Plugin in Vue, all child directives are also globally registered.

### Using `Vue.component()`

Import the Component and Component Plugin from EJ2 Vue Package, register the same using the `Vue.component()` with name of Component from ComponentPlugin and the EJ2 Vue Component as its arguments.

Refer the code snippet given below.

```
`ts
import { DateTimePickerComponent, DateTimePickerPlugin } from '@syncfusion/ej2-vue-calendars';
Vue.component(DateTimePickerPlugin.name, DateTimePickerComponent);
`
```

By using `Vue.component()`, only the EJ2 Vue Component is registered. Child directives needs to be registered separately.

### Creating Vue Sample

Add the EJ2 Vue DateTimePicker using `<ejs-datetimepicker>` to the `<template>` section of the `App.vue` file in `src` directory, the content attribute of the DateTimePicker component is provided as name in data option in the `<script>` section.

```
`
<template>
<div id="app">
<ejs-datetimepicker :placeholder="waterMark" ></ejs-datetimepicker>
</div>
</template>
<script>
import Vue from 'vue';
import { DateTimePickerPlugin } from '@syncfusion/ej2-vue-calendars';
Vue.use(DateTimePickerPlugin);
export default {
name: 'app',
data () {
return {
waterMark : 'Select a datetime'
}
}
}
</script>
`
```

### Adding CSS Reference

To render the DateTimePicker component, need to import DateTimePicker and its dependent component's styles as given below in `<style>` section of the `App.vue` file.

```
<style>
@import '../node_modules/@syncfusion/ej2-base/styles/material.css';
@import '../node_modules/@syncfusion/ej2-buttons/styles/material.css';
@import '../node_modules/@syncfusion/ej2-inputs/styles/material.css';
@import '../node_modules/@syncfusion/ej2-popups/styles/material.css';
@import '../node_modules/@syncfusion/ej2-lists/styles/material.css';
@import "../node_modules/@syncfusion/ej2-vue-calendars/styles/material.css";
</style>
```

If you want to refer the combined component styles, please make use of our [CRG](#) (Custom Resource Generator) in your application.

### Running the Application

Now run the `npm run dev` command in the console, it will build your application and open in the browser.

### APP.VUE

```
<template>
  <div id="app">
    <div class='wrapper'>
      <ejs-datetimepicker :placeholder="waterMark" ></ejs-datetimepicker>
    </div>
  </div>
</template>
<script>
import Vue from 'vue';
import { DateTimePickerPlugin } from '@syncfusion/ej2-vue-calendars';
Vue.use(DateTimePickerPlugin);
export default {
  data () {
    return {
      waterMark : 'Select a datetime'
    }
  }
}
</script>
<style>
@import '../node_modules/@syncfusion/ej2-base/styles/material.css';
@import '../node_modules/@syncfusion/ej2-buttons/styles/material.css';
@import '../node_modules/@syncfusion/ej2-inputs/styles/material.css';
@import '../node_modules/@syncfusion/ej2-popups/styles/material.css';
@import '../node_modules/@syncfusion/ej2-lists/styles/material.css';
@import "../node_modules/@syncfusion/ej2-vue-calendars/styles/material.css";
```

```
.wrapper {
  max-width: 250px;
  margin: 0 auto;
}
</style>
```

```
{% previewsample "page.domainurl/code-snippet/datetimepicker/getting-started-cs5" %}
```

### Setting the value,min and max

The minimum and maximum date time can be defined with the help of `min` and `max` property. The following example demonstrates to set the `min` and `max` on initializing the DateTimePicker. To know more about range restriction in DateTimePicker, please refer this [page](#).

### APP.VUE

```
<template>
  <div id="app">
    <div class='wrapper'>
      <ejs-datetimepicker :placeholder="waterMark" :min="minDate"
      :max="maxDate" :value="val"></ejs-datetimepicker>
    </div>
  </div>
</template>
<script>
import Vue from 'vue';
import { DateTimePickerPlugin } from '@syncfusion/ej2-vue-calendars';
Vue.use(DateTimePickerPlugin);
export default {
  data () {
    return {
      waterMark : 'Select a datetime',
      minDate : new Date('5/5/2019 2:00 AM'),
      maxDate : new Date('5/25/2019 2:00 AM'),
      val : new Date('5/10/2019 12:00 AM')
    }
  }
}
</script>
<style>
@import '../node_modules/@syncfusion/ej2-base/styles/material.css';
@import '../node_modules/@syncfusion/ej2-buttons/styles/material.css';
@import '../node_modules/@syncfusion/ej2-inputs/styles/material.css';
@import '../node_modules/@syncfusion/ej2-popups/styles/material.css';
@import '../node_modules/@syncfusion/ej2-lists/styles/material.css';
@import "../node_modules/@syncfusion/ej2-vue-calendars/styles/material.css";
.wrapper {
  max-width: 250px;
  margin: 0 auto;
}
</style>
```

```
{% previewsample "page.domainurl/code-snippet/datetimepicker/getting-started-cs6" %}
```

If the value of `min` or `max` properties changed through code behind, then you have to update the `value` property to set within the range.

## See Also

- [Render DateTimePicker with specific culture](#)

## Getting Started with the Vue DateTimePicker Component in Vue 3

This article provides a step-by-step guide for setting up a [Vite](#) project with a JavaScript environment and integrating the Syncfusion Vue DateTimePicker component using the [Composition API](#) / [Options API](#).

The **Composition API** is a new feature introduced in Vue.js 3 that provides an alternative way to organize and reuse component logic. It allows developers to write components as functions that use smaller, reusable functions called composition functions to manage their properties and behavior.

The **Options API** is the traditional way of writing Vue.js components, where the component logic is organized into a series of options that define the component's properties and behavior. These options include data, methods, computed properties, watchers, lifecycle hooks, and more.

## Prerequisites

### [System requirements for Syncfusion Vue UI components](#)

### Set up the Vite project

A recommended approach for beginning with Vue is to scaffold a project using [Vite](#). To create a new Vite project, use one of the commands that are specific to either NPM or Yarn.

```
`bash
```

```
npm create vite@latest
```

```
,
```

```
or
```

```
`bash
```

```
yarn create vite
```

```
,
```

Using one of the above commands will lead you to set up additional configurations for the project as below:

1. Define the project name: We can specify the name of the project directly. Let's specify the name of the project as **my-project** for this article.

```
`bash
```

```
? Project name: » my-project
```

```
,
```

2. Select **Vue** as the framework. It will create a Vue 3 project.

```
`bash
```

```
? Select a framework: » - Use arrow-keys. Return to submit.
```

```
Vanilla
```

```
Vue
```

React

Preact

Lit

Svelte

Others

,

3. Choose **JavaScript** as the framework variant to build this Vite project using JavaScript and Vue.

```
`bash
```

? Select a variant: » - Use arrow-keys. Return to submit.

JavaScript

TypeScript

Customize with create-vue ↗

Nuxt ↗

,

4. Upon completing the aforementioned steps to create the **my-project**, run the following command to install its dependencies:

```
`bash
```

```
cd my-project
```

```
npm install
```

,

or

```
`bash
```

```
cd my-project
```

```
yarn install
```

,

Now that **my-project** is ready to run with default settings, let's add Syncfusion components to the project.

#### Add Syncfusion Vue packages

Syncfusion Vue component packages are available at [npmjs.com](https://www.npmjs.com). To use Syncfusion Vue components in the project, install the corresponding npm package.

This article uses the [Vue DateTimePicker component](#) as an example. To use the Vue DateTimePicker component in the project, the **@syncfusion/ej2-vue-calendars** package needs to be installed using the following command:

```
`bash
```

```
npm install @syncfusion/ej2-vue-calendars --save
```



or

```
`bash
```

```
yarn add @syncfusion/ej2-vue-calendars
```

### Import Syncfusion CSS styles

You can import themes for the Syncfusion Vue component in various ways, such as using CSS or SASS styles from npm packages, CDN, [CRG](#) and [Theme Studio](#). Refer to [themes topic](#) to know more about built-in themes and different ways to refer to themes in a Vue project.

In this article, **Material** theme is applied using CSS styles, which are available in installed packages. The necessary **Material** CSS styles for the DateTimePicker component and its dependents were imported into the `<style>` section of **src/App.vue** file.

#### ~/SRC/APP.VUE

```
<style>
@import '../node_modules/@syncfusion/ej2-base/styles/material.css';
@import '../node_modules/@syncfusion/ej2-buttons/styles/material.css';
@import '../node_modules/@syncfusion/ej2-inputs/styles/material.css';
@import '../node_modules/@syncfusion/ej2-popups/styles/material.css';
@import '../node_modules/@syncfusion/ej2-lists/styles/material.css';
@import "../node_modules/@syncfusion/ej2-vue-calendars/styles/material.css";
</style>
```

The order of importing CSS styles should be in line with its dependency graph.

### Add Syncfusion Vue component

Follow the below steps to add the Vue DateTimePicker component using **Composition API** or **Options API**:

1.First, import and register the DateTimePicker component in the **script** section of the **src/App.vue** file. If you are using the **Composition API**, you should add the **setup** attribute to the **script** tag to indicate that Vue will be using the **Composition API**.

#### COMPOSITION API (~/SRC/APP.VUE)

```
<script setup>
import { DateTimePickerComponent as EjsDatetimepicker } from
"@syncfusion/ej2-vue-calendars";
</script>
```

#### OPTIONS API (~/SRC/APP.VUE)

```
<script>
import { DateTimePickerComponent } from "@syncfusion/ej2-vue-calendars";
//Component registration
export default {
  name: "App",
  components: {
    'ejs-datetimepicker' : DateTimePickerComponent,
```

```
}
}
```

2. In the **template** section, define the DateTimePicker component with the [dataSource](#) property and column definitions.

### ~/SRC/APP.VUE

```
<template>
<div class="control_wrapper">
<ejs-datetimepicker></ejs-datetimepicker>
</div>
</template>
```

Here is the summarized code for the above steps in the **src/App.vue** file:

### COMPOSITION API (~/SRC/APP.VUE)

```
<template>
<div class="control_wrapper">
<ejs-datetimepicker></ejs-datetimepicker>
</div>
</template>
<script setup>
import { DateTimePickerComponent as EjsDatetimepicker } from
"@syncfusion/ej2-vue-calendars";
</script>
<style>
@import '../node_modules/@syncfusion/ej2-base/styles/material.css';
@import '../node_modules/@syncfusion/ej2-buttons/styles/material.css';
@import '../node_modules/@syncfusion/ej2-inputs/styles/material.css';
@import '../node_modules/@syncfusion/ej2-popups/styles/material.css';
@import '../node_modules/@syncfusion/ej2-lists/styles/material.css';
@import "@syncfusion/ej2-vue-calendars/styles/material.css";
.control_wrapper {
max-width: 250px;
margin: 0 auto;
}
</style>
```

### OPTIONS API (~/SRC/APP.VUE)

```
<template>
<div class="control_wrapper">
<ejs-datetimepicker></ejs-datetimepicker>
</div>
</template>
<script>
import { DateTimePickerComponent } from "@syncfusion/ej2-vue-calendars";
//Component registration
export default {
name: 'App',
components: {
"ejs-datetimepicker": DateTimePickerComponent
},
}
```

```

}
</script>
<style>
@import '../node_modules/@syncfusion/ej2-base/styles/material.css';
@import '../node_modules/@syncfusion/ej2-buttons/styles/material.css';
@import '../node_modules/@syncfusion/ej2-inputs/styles/material.css';
@import '../node_modules/@syncfusion/ej2-popups/styles/material.css';
@import '../node_modules/@syncfusion/ej2-lists/styles/material.css';
@import "../node_modules/@syncfusion/ej2-vue-calendars/styles/material.css";
.control_wrapper {
max-width: 250px;
margin: 0 auto;
}
</style>

```

### Run the project

To run the project, use the following command:

```
`bash
```

```
npm run dev
```

```
`
```

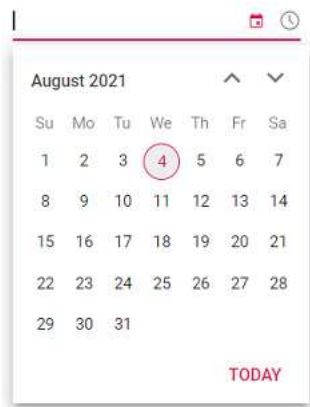
or

```
`bash
```

```
yarn run dev
```

```
`
```

The output will appear as follows:



### Setting the value,min and max

The minimum and maximum date time can be defined with the help of `min` and `max` property. The following example demonstrates to set the `min` and `max` on initializing the DateTimePicker. To know more about range restriction in DateTimePicker, please refer this [page](#).

#### COMPOSITION API (~SRC/APP.VUE)

```

<template>
<div id="app">

```

```

<div class='wrapper'>
  <ejs-datetimepicker :placeholder="data[0].waterMark" :min="data[0].minDate"
    :max="data[0].maxDate" :value="data[0].val"></ejs-datetimepicker>
</div>
</div>
</template>
<script setup>
import { DateTimePickerComponent as EjsDatetimepicker } from
"@syncfusion/ej2-vue-calendars";
const data = [{ waterMark : 'Select a datetime',
minDate : new Date('5/5/2019 2:00 AM'),
maxDate : new Date('5/25/2019 2:00 AM'),
val : new Date('5/10/2019 12:00 AM') }]];
</script>
<style>
@import '../node_modules/@syncfusion/ej2-base/styles/material.css';
@import '../node_modules/@syncfusion/ej2-buttons/styles/material.css';
@import '../node_modules/@syncfusion/ej2-inputs/styles/material.css';
@import '../node_modules/@syncfusion/ej2-popups/styles/material.css';
@import '../node_modules/@syncfusion/ej2-lists/styles/material.css';
@import "@syncfusion/ej2-vue-calendars/styles/material.css";
.wrapper {
max-width: 250px;
margin: 0 auto;
}
</style>

```

### OPTIONS API (~SRC/APP.VUE)

```

<template>
<div id="app">
  <div class='wrapper'>
    <ejs-datetimepicker :placeholder="waterMark" :min="minDate" :max="maxDate"
      :value="val"></ejs-datetimepicker>
  </div>
</div>
</template>
<script>
import { DateTimePickerComponent } from "@syncfusion/ej2-vue-calendars";
//Component registration
export default {
name: 'App',
components: {
"ejs-datetimepicker": DateTimePickerComponent
},
data () {
return {
waterMark : 'Select a datetime',
minDate : new Date('5/5/2019 2:00 AM'),
maxDate : new Date('5/25/2019 2:00 AM'),
val : new Date('5/10/2019 12:00 AM')
}
}
}
</script>
<style>

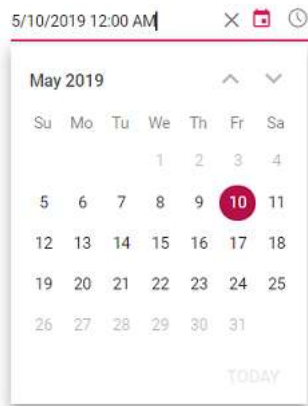
```

```

@import '../node_modules/@syncfusion/ej2-base/styles/material.css';
@import '../node_modules/@syncfusion/ej2-buttons/styles/material.css';
@import '../node_modules/@syncfusion/ej2-inputs/styles/material.css';
@import '../node_modules/@syncfusion/ej2-popups/styles/material.css';
@import '../node_modules/@syncfusion/ej2-lists/styles/material.css';
@import "../node_modules/@syncfusion/ej2-vue-calendars/styles/material.css";
.wrapper {
max-width: 250px;
margin: 0 auto;
}
</style>

```

The output will appear as follows:



If the value of `min` or `max` properties changed through code behind, then you have to update the `value` property to set within the range.

See Also

- [Render DateTimePicker with specific culture](#)

## Globalization in Vue Datetimepicker component

Globalization is the combination of internalization and localization. You can adapt the component to various languages by parsing and formatting the date or number [Internationalization](#), and also add culture specific customization and translation to the text [localization](#).

By default, the date format, week, month, time format and meridian names are specific to the American English culture. It utilizes the

[Essential JavaScript 2 Internationalization](#) package to parse and format the date object based on the culture by using the official [UNICODE CLDR](#) JSON data. It provides the `loadCldr` method to load culture specific CLDR JSON data. To use a different culture other than English, follow the steps below:

- Install the `CLDR-Data` package by using the following command (installs all the CLDR JSON data). To know more about CLDR-Data refer to the [CLDR-Data](#) link.

```
npm install cldr-data --save
```

Once the package is installed, you can find the culture specific JSON data under the location `/node_modules/cldr-data`.

- Import the installed CLDR JSON data into the `app.vue` file.
- Use the [loadCldr](#) method to load the culture specific CLDR JSON data

from the installed location to `app.vue` file.

- DateTimePicker displayed `Sunday` as the first day of week based on default culture ("en-US"). If you want to display the DateTimePicker with loaded culture's first day of week, you need to import `weekdata.json` file from the `cldr-data/supplemental` as given in the code example.

```
`ts
```

```
//Load the loadCldr from ej2-base
```

```
import { loadCldr } from '@syncfusion/ej2-base';
```

```
import * as numberingSystems from 'cldr-data/supplemental/numberingSystems.json';
```

```
import * as gregorian from 'cldr-data/main/de/ca-gregorian.json';
```

```
import * as numbers from 'cldr-data/main/de/numbers.json';
```

```
import * as timeZoneNames from 'cldr-data/main/de/timeZoneNames.json';
```

```
import * as weekData from 'cldr-data/supplemental/weekdata.json'; // To load the culture based first day of week
```

```
loadCldr(numberingSystems, gregorian, numbers, timeZoneNames, weekData);
```

The `Localization` library allows you to localize default text content of the DateTimePicker. The DateTimePicker component has static text for **today** feature that can be changed to other cultures (Arabic, Deutsch, French, etc.) by defining the [locale](#) value and translation object.

Locale keywords | Text

today | Name of the button to choose Today date.

placeholder | Hint to describe expected value in input element.

- Before changing to a culture other than `English`, ensure that locale text for the concerned culture is loaded through `load` method of

[L10n](#) class.

```
`ts
```

```
//Load the L10n, loadCldr from ej2-base
```

```
import { loadCldr, L10n } from '@syncfusion/ej2-base';
```

```
//load the locale object to set the localized placeholder value
```

```
L10n.load({
  'de': {
    'datetimepicker': { placeholder: 'Wählen Sie ein Datum und eine Uhrzeit aus', today: 'heute' }
  }
});
```

- Set the culture by using the [locale](#) property.

The following example demonstrates the DateTimePicker in **German** culture.

#### APP.VUE

```
<template>
  <div id="app">
    <div class='wrapper'>
      <ejs-datetimepicker id="datetime" locale="de" ></ejs-
datetimepicker>
    </div>
  </div>
</template>
<script>
import Vue from 'vue';
import { loadCldr,L10n} from '@syncfusion/ej2-base';
import { DateTimePickerPlugin } from '@syncfusion/ej2-vue-calendars';
// Here we have referred local json files for preview purpose
import * as numberingSystems from './numberingSystems.json';
import * as gregorian from './ca-gregorian.json';
import * as numbers from './numbers.json';
import * as timeZoneNames from './timeZoneNames.json';
Vue.use(DateTimePickerPlugin);
loadCldr(numberingSystems, gregorian, numbers, timeZoneNames);
L10n.load({
  'de': {
    'datetimepicker': { placeholder: "Wählen Sie Datum und Uhrzeit",
      today: "heute"}
  }
});
export default {}
</script>
<style>
@import '../node_modules/@syncfusion/ej2-base/styles/material.css';
@import '../node_modules/@syncfusion/ej2-buttons/styles/material.css';
@import '../node_modules/@syncfusion/ej2-inputs/styles/material.css';
@import '../node_modules/@syncfusion/ej2-popups/styles/material.css';
@import '../node_modules/@syncfusion/ej2-lists/styles/material.css';
@import "../node_modules/@syncfusion/ej2-vue-calendars/styles/material.css";
.wrapper {
  max-width: 250px;
  margin: 0 auto;
}
</style>
```

```
{% previewsample "page.domainurl/code-snippet/datetimepicker/locale-cs1" %}
```

### Right-To-Left

The DateTimePicker supports RTL (right-to-left) functionality for languages like Arabic and Hebrew to displays the text in the right-to-left direction. Use [enableRtl](#) property to set the RTL direction. The following code example initialize the DateTimePicker component in Arabic culture and also explains how to set the localized text to the placeholder using `load` method of [L10n](#) class.

### APP.VUE

```
<template>
  <div id="app">
    <div class='wrapper'>
      <ejs-datetimepicker id="datetime" :locale="locale" :enableRtl="rtl"
    ></ejs-datetimepicker>
    </div>
  </div>
</template>
<script>
import Vue from 'vue';
import { loadCldr,L10n} from '@syncfusion/ej2-base';
import { DateTimePickerPlugin } from '@syncfusion/ej2-vue-calendars';
// Here we have referred local json files for preview purpose
import * as numberingSystems from './numberingSystems.json';
import * as gregorian from './ca-gregorian.json';
import * as numbers from './numbers.json';
import * as timeZoneNames from './timeZoneNames.json';
Vue.use(DateTimePickerPlugin);
loadCldr(numberingSystems, gregorian, numbers, timeZoneNames);
L10n.load({
  'ar': {
    'datetimepicker': { placeholder: 'حدد التاريخ والوقت',
      today: 'اليوم' }
  }
});
export default {
  data () {
    return {
      locale: "ar",
      rtl: true
    }
  }
}
</script>
<style>
@import '../node_modules/@syncfusion/ej2-base/styles/material.css';
@import '../node_modules/@syncfusion/ej2-buttons/styles/material.css';
@import '../node_modules/@syncfusion/ej2-inputs/styles/material.css';
@import '../node_modules/@syncfusion/ej2-popups/styles/material.css';
@import '../node_modules/@syncfusion/ej2-lists/styles/material.css';
@import "../node_modules/@syncfusion/ej2-vue-calendars/styles/material.css";
.wrapper {
  max-width: 250px;
  margin: 0 auto;
}
```



```
</style>
```

```
{% previewsample "page.domainurl/code-snippet/datetimepicker/locale-rtl-cs1" %}
```

## Strict mode in Vue Datetimepicker component

### Enable Strict Mode

The [strictMode](#) is an act, that allows the user to enter only the valid date and time within the specified min/max range in textbox. If the input entered is invalid, then the component will stay with the previous value. Else, if the date and time is out of range, then the component will set the date to the min/max value.

The following example demonstrates the DateTimePicker in `strictMode` with min/max range of 5/5/2019 2:00 AM to 5/25/2019 2:00 AM. Here, it allows to enter only the valid date and time within the specified range. If you are trying to enter the out-of-range value as like 5/28/2019, then the value will set to the `max` value as 5/25/2019 2:00 AM. Since the value 28 is greater than to `max` value of 25. Or else if you are trying to enter the invalid date, then the value will stay with the previous value.

### APP.VUE

```
<template>
  <div id="app">
    <div class='wrapper'>
      <ejs-datetimepicker :min="minDate" :max="maxDate" :value="dateVal"
:strictMode="mode" ></ejs-datetimepicker>
    </div>
  </div>
</template>
<script>
import Vue from 'vue';
import { DateTimePickerPlugin } from '@syncfusion/ej2-vue-calendars';
Vue.use(DateTimePickerPlugin);
export default {
  data () {
    return {
      minDate : new Date('5/5/2019 2:00 AM'),
      maxDate : new Date('5/25/2019 2:00 AM'),
      dateVal : new Date('5/10/2019 2:00 AM'),
      mode : true
    }
  }
}
</script>
<style>
@import '../node_modules/@syncfusion/ej2-base/styles/material.css';
@import '../node_modules/@syncfusion/ej2-buttons/styles/material.css';
@import '../node_modules/@syncfusion/ej2-inputs/styles/material.css';
@import '../node_modules/@syncfusion/ej2-popups/styles/material.css';
@import '../node_modules/@syncfusion/ej2-lists/styles/material.css';
@import "../node_modules/@syncfusion/ej2-vue-calendars/styles/material.css";
.wrapper {
  max-width: 250px;
  margin: 0 auto;
}
</style>
```

```
{% previewsample "page.domainurl/code-snippet/datetimepicker/getting-started-cs9" %}
```

### Disable Strict Mode

By default, the DateTimePicker act in strictMode `false` state, that allows to enter the invalid or out-of-range datetime in textbox.

If the datetime is out-of-range or invalid, then the model value will be set to `out of range` datetime value or `null` respectively with highlighted `error` class to indicates the datetime is out of range or invalid.

The following example demonstrates the `strictMode` as `false`. Here, it allows to enter the valid or invalid value in textbox. If you are entering the out-of-range or invalid datetime value, then the model value will be set to `out of range` datetime value or `null` respectively with highlighted `error` class to indicates the datetime is out of range or invalid.

### APP.VUE

```
<template>
  <div id="app">
    <div class='wrapper'>
      <ejs-datetimepicker :min="minDate" :max="maxDate" :value="dateVal"
:placeholder="waterMark" ></ejs-datetimepicker>
    </div>
  </div>
</template>
<script>
import Vue from 'vue';
import { DateTimePickerPlugin } from '@syncfusion/ej2-vue-calendars';
Vue.use(DateTimePickerPlugin);
export default {
  data () {
    return {
      minDate : new Date('5/5/2017 2:00 PM'),
      maxDate : new Date('5/25/2017 3:00 PM'),
      dateVal : new Date('5/25/2017 4:00 PM'),
      waterMark: "Select a date and time"
    }
  }
}
</script>
<style>
@import '../node_modules/@syncfusion/ej2-base/styles/material.css';
@import '../node_modules/@syncfusion/ej2-buttons/styles/material.css';
@import '../node_modules/@syncfusion/ej2-inputs/styles/material.css';
@import '../node_modules/@syncfusion/ej2-popups/styles/material.css';
@import '../node_modules/@syncfusion/ej2-lists/styles/material.css';
@import "../node_modules/@syncfusion/ej2-vue-calendars/styles/material.css";
.wrapper {
  max-width: 250px;
  margin: 0 auto;
}
</style>
```

```
{% previewsample "page.domainurl/code-snippet/datetimepicker/getting-started-cs10" %}
```

## Date time range in Vue Datetimepicker component

DateTimePicker provides an option to select a date and time value within a specified range by using the [min](#) and [max](#) properties. Always the min value has to be lesser than the max value.

When the min and max properties are configured and the selected datetime value is out-of-range or invalid, then the model value will be set to **out of range** datetime value or **null** respectively with highlighted **error** class to indicates the datetime is out of range or invalid. The value property depends on the min/max with respect to [strictMode](#) property.

The below example allows selecting a date within the range from 7th to 27th day in a month.

### APP.VUE

```
<template>
  <div id="app">
    <div class='wrapper'>
      <ejs-datetimepicker :min="minDate" :max="maxDate"
:placeholder="waterMark" :value="dateVal"></ejs-datetimepicker>
    </div>
  </div>
</template>
<script>
import Vue from 'vue';
import { DateTimePickerPlugin } from '@syncfusion/ej2-vue-calendars';
Vue.use(DateTimePickerPlugin);
export default {
  data () {
    return {
      minDate : new Date(new Date().getFullYear(), new Date().getMonth(),
7, 0, 0, 0),
      maxDate : new Date(new Date().getFullYear(), new Date().getMonth(),
27, new Date().getHours(), new Date().getMinutes(), new
Date().getSeconds()),
      dateVal : new Date(new Date().setDate(14)),
      waterMark : 'Select a datetime'
    }
  }
}
</script>
<style>
@import '../node_modules/@syncfusion/ej2-base/styles/material.css';
@import '../node_modules/@syncfusion/ej2-buttons/styles/material.css';
@import '../node_modules/@syncfusion/ej2-inputs/styles/material.css';
@import '../node_modules/@syncfusion/ej2-popups/styles/material.css';
@import '../node_modules/@syncfusion/ej2-lists/styles/material.css';
@import "../node_modules/@syncfusion/ej2-vue-calendars/styles/material.css";
.wrapper {
  max-width: 250px;
  margin: 0 auto;
}
</style>
```

{% previewsample "page.domainurl/code-snippet/datetimepicker/getting-started-cs4" %}

If the value of `min` or `max` properties changed through code behind, then you have to update the `value` property to set within the range.

### Date time masking in Vue Datetimepicker component

DateTimePicker has `enableMask` property that provides the option to enable the built-in date masking support. Also, you must inject the `MaskedDateTime` module to enable the masking support.

#### APP.VUE

```
<template>
<div id="app">
  <div class='wrapper'>
    <ejs-datetimepicker id="datetimepicker" :enableMask="true"></ejs-
datetimepicker>
  </div>
</div>
</template>
<script>
import Vue from "vue";
import { DateTimePickerPlugin, DateTimePicker, MaskedDateTime } from
"@syncfusion/ej2-vue-calendars";
DateTimePicker.Inject (MaskedDateTime)
Vue.use(DateTimePickerPlugin);
export default Vue.extend({
  data: function() {
    return {
    };
  },
  provide: {
    datetimepicker: [MaskedDateTime]
  }
});
</script>
<style>
@import '../node_modules/@syncfusion/ej2-base/styles/material.css';
@import '../node_modules/@syncfusion/ej2-buttons/styles/material.css';
@import '../node_modules/@syncfusion/ej2-inputs/styles/material.css';
@import '../node_modules/@syncfusion/ej2-popups/styles/material.css';
@import "../node_modules/@syncfusion/ej2-vue-calendars/styles/material.css";
.wrapper {
  max-width: 250px;
  margin: 0 auto;
}
</style>
```

{% previewsample "page.domainurl/code-snippet/datetimepicker/mask-module-cs1" %}

The mask pattern is defined based on the provided date format to the component. If the format is not specified, the mask pattern is formed based on the default format of the current culture.

| **Keys** | **Actions** |

| --- | --- |

| Up / Down arrows | To increment and decrement the selected portion of the date and time. |

| Left / Right arrows and Tab | To navigate the selection from one portion to next portion |

The following example demonstrates default and custom format of DateTimePicker component with mask.

#### APP.VUE

```
<template>
<div id="app">
  <div class='wrapper1'>
    <!-- Specifies the masked datetimepicker without format property. -->
  >
    <ejs-datetimepicker id="datetimepicker" :enableMask="true" ></ejs-
datetimepicker>
  </div>
  <div class='wrapper2'>
    <!-- Specifies the masked datetimepicker with format property. -->
    <ejs-datetimepicker id="datetimepicker" :enableMask="true"
:format='dateFormat'></ejs-datetimepicker>
  </div>
</div>
</template>
<script>
import Vue from "vue";
import { DateTimePickerPlugin, DateTimePicker, MaskedDateTime } from
"@syncfusion/ej2-vue-calendars";
DateTimePicker.Inject(MaskedDateTime)
Vue.use(DateTimePickerPlugin);
export default Vue.extend({
  data: function() {
    return {
      dateFormat: 'M/d/yyyy hh:mm a'
    };
  },
  provide: {
    datetimepicker: [MaskedDateTime]
  }
});
</script>
<style>
@import '../node_modules/@syncfusion/ej2-base/styles/material.css';
@import '../node_modules/@syncfusion/ej2-buttons/styles/material.css';
@import '../node_modules/@syncfusion/ej2-inputs/styles/material.css';
@import '../node_modules/@syncfusion/ej2-popups/styles/material.css';
@import "../node_modules/@syncfusion/ej2-vue-calendars/styles/material.css";
#wrapper1{
  min-width: 250px;
  float: left;
  margin-left: 100px;
}
#wrapper2{
  min-width: 250px;
  float: right;
  margin-right: 100px;
}
</style>
```

```
{% previewsample "page.domainurl/code-snippet/datetimepicker/mask-module-cs2" %}
```

### Configure Mask Placeholder

You can change mask placeholder value through property `maskPlaceholder`. By default , it takes the full name of date and time co-ordinates such as `day`, `month`, `year`, `hour` etc.

While changing to a culture other than `English`, ensure that locale text for the concerned culture is loaded through load method of `L10n` class for mask placeholder values like below.

```
`ts
//Load the L10n from ej2-base
import { L10n } from '@syncfusion/ej2-base';
//load the locale object to set the localized mask placeholder value
L10n.load({
  'de': {
    'datetimepicker': { day: 'Tag' , month: 'Monat', year: 'Jahr', hour: 'Stunde' ,minute: 'Minute',
    second:'Sekunden' }
  }
});
`
```

The following example demonstrates default and customized mask placeholder value.

### APP.VUE

```
<template>
<div id="app">
  <div class='wrapper1'>
    <!-- Specifies the masked datetimepicker without mask placeholder.
-->
    <ejs-datetimepicker id="datetimepicker" :enableMask="true" ></ejs-
datetimepicker>
  </div>
  <div class='wrapper2'>
    <!-- Specifies the masked datetimepicker with mask placeholder. --
>
    <ejs-datetimepicker id="placeholder" :enableMask="true"
:maskPlaceholder='maskPlaceholderValue'></ejs-datetimepicker>
  </div>
</div>
</template>
<script>
import Vue from "vue";
import { DateTimePickerPlugin, DateTimePicker, MaskedDateTime } from
"@syncfusion/ej2-vue-calendars";
DateTimePicker.Inject(MaskedDateTime)
Vue.use(DateTimePickerPlugin);
export default Vue.extend({
  data: function() {
    return {
```

```

        maskPlaceholderValue: {day: 'd', month: 'M', year: 'y', hour: 'h',
minute: 'm', second: 's'}
    };
  },
  provide: {
    datetimepicker: [MaskedDateTime]
  }
});
</script>
<style>
@import '../node_modules/@syncfusion/ej2-base/styles/material.css';
@import '../node_modules/@syncfusion/ej2-buttons/styles/material.css';
@import '../node_modules/@syncfusion/ej2-inputs/styles/material.css';
@import '../node_modules/@syncfusion/ej2-popups/styles/material.css';
@import "../node_modules/@syncfusion/ej2-vue-calendars/styles/material.css";
#wrapper1{
  min-width: 250px;
  float: left;
  margin-left: 100px;
}
#wrapper2{
  min-width: 250px;
  float: right;
  margin-right: 100px;
}
</style>

```

{% previewsample "page.domainurl/code-snippet/datetimepicker/mask-module-cs3" %}

### Customization in Vue Datetimepicker component

The DateTimePicker is available for UI customization that can be achieved by using available properties and events in the component.

#### Day and Time Cell format

The DateTimePicker is available for UI customization based on your application requirements. It can be achieved by using [renderDayCell](#) event that provides an option to customize each day cell on rendering.

The following example disables the weekends of every month by using `renderDayCell` event.

#### APP.VUE

```

<template>
  <div id="app">
    <div class='wrapper'>
      <ejs-datetimepicker :renderDayCell="disableDate"
:placeholder="waterMark"></ejs-datetimepicker>
    </div>
  </div>
</template>
<script>
import Vue from 'vue';
import { DateTimePickerPlugin } from '@syncfusion/ej2-vue-calendars';
Vue.use(DateTimePickerPlugin);
export default {
  data () {
    return {

```

```

        waterMark : 'Select a date and time'
      }
    },
    methods: {
      disableDate: function(args) {
        if (args.date.getDay() === 0 || args.date.getDay() === 6) {
          args.isDisabled = true;
        }
      }
    }
  }
</script>
<style>
@import '../node_modules/@syncfusion/ej2-base/styles/material.css';
@import '../node_modules/@syncfusion/ej2-buttons/styles/material.css';
@import '../node_modules/@syncfusion/ej2-inputs/styles/material.css';
@import '../node_modules/@syncfusion/ej2-popups/styles/material.css';
@import '../node_modules/@syncfusion/ej2-lists/styles/material.css';
@import "../node_modules/@syncfusion/ej2-vue-calendars/styles/material.css";
.wrapper {
  max-width: 250px;
  margin: 0 auto;
}
</style>

```

{% previewsample "page.domainurl/code-snippet/datetimepicker/getting-started-cs2" %}

### Adding mandatory asterisk to placeholder and float label

You can add a mandatory asterisk(\*) to placeholder and float label using **<e-input-group.e-control-wrapper.e-float-input .e-float-text::after</b>** class.

### APP.VUE

```

<template>
  <div id="app">
    <div class='wrapper'>
      <ejs-datetimepicker :placeholder="Select Date"
      :floatLabelType="auto"></ejs-datetimepicker>
    </div>
  </div>
</template>
<script>
import Vue from 'vue';
import { DateTimePickerPlugin } from '@syncfusion/ej2-vue-calendars';
Vue.use(DateTimePickerPlugin);
export default {
  data () {
    return {
      waterMark : 'Select a datetime'
    }
  }
}
</script>
<style>
@import '../node_modules/@syncfusion/ej2-base/styles/material.css';
@import '../node_modules/@syncfusion/ej2-buttons/styles/material.css';

```



```

@import '../node_modules/@syncfusion/ej2-inputs/styles/material.css';
@import '../node_modules/@syncfusion/ej2-popups/styles/material.css';
@import '../node_modules/@syncfusion/ej2-lists/styles/material.css';
@import "../node_modules/@syncfusion/ej2-vue-calendars/styles/material.css";

.wrapper {
  max-width: 250px;
  margin: 0 auto;
}
.e-input-group.e-control-wrapper.e-float-input .e-float-text::after {
  content: '*';
  color: red;
}
</style>

```

{% previewsample "page.domainurl/code-snippet/datetimepicker/getting-started-cs3" %}

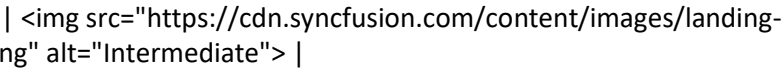
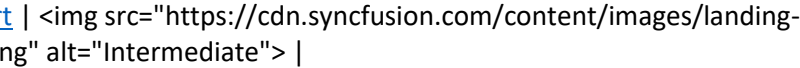
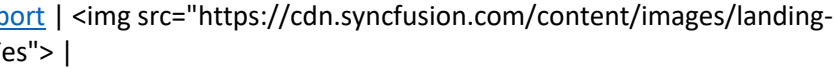
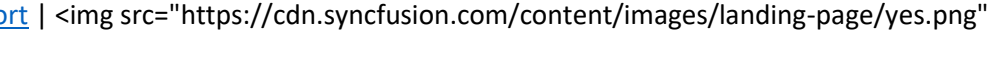
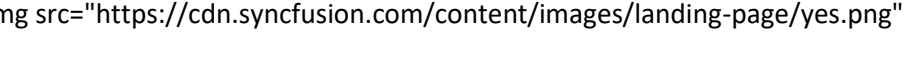
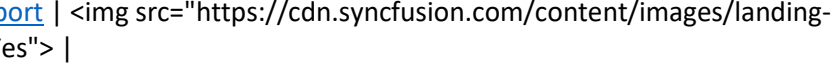
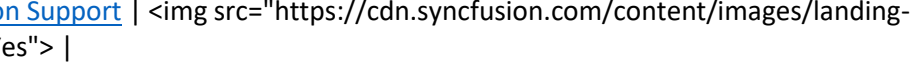
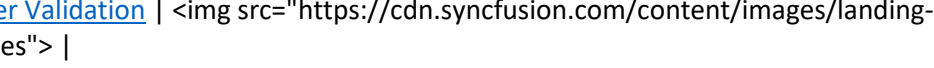
See Also

- [How to disable the DateTimePicker component](#)
- [How to customize the DateTimePicker day header](#)

## Accessibility in Vue Datetimepicker component

The DateTimePicker component followed the accessibility guidelines and standards, including [ADA](#), [Section 508](#), [WCAG 2.2](#) standards, and [WCAG roles](#) that are commonly used to evaluate accessibility.

The accessibility compliance for the DateTimePicker component is outlined below.

Accessibility Criteria   Compatibility
--   --
<a href="#">WCAG 2.2 Support</a>   
<a href="#">Section 508 Support</a>   
<a href="#">Screen Reader Support</a>   
<a href="#">Right-To-Left Support</a>   
<a href="#">Color Contrast</a>   
<a href="#">Mobile Device Support</a>   
<a href="#">Keyboard Navigation Support</a>   
<a href="#">Accessibility Checker Validation</a>   

| [Axe-core Accessibility Validation](#) |  |

<style>

.post .post-content img {

display: inline-block;

margin: 0.5em 0;

}

</style>

<div> - All features of the component meet the requirement.</div>

<div> - Some features of the component do not meet the requirement.</div>

<div> - The component does not meet the requirement.</div>

### WAI-ARIA attributes

The Web accessibility defines a way to make web content and web applications more accessible to disabled people. It especially helps the dynamic content change and advanced user interface controls developed with Ajax, HTML, JavaScript, and related technologies.

DateTimePicker provides built-in compliance with the [WAI-ARIA](#) specifications. WAI-ARIA supports is achieved through the attributes like `aria-expanded`, `aria-disabled`, `aria-activedescendant` applied to the input element.

To know about the accessibility of Calendar refer to the Calendar's [Accessibility](#) section.

It helps to provide information about the widget for assistive technology to the disabled person in screen reader.

- **Aria-expanded:** attributes indicates the state of a collapsible element.
- **Aria-disabled:** attribute indicates the disabled state of this DateTimePicker component.
- **Aria-activedescendent:** attribute helps in managing the current active child of the DateTimePicker component.

### Keyboard Interaction

You can use the following keys to interact with the DateTimePicker. The component implements the keyboard navigation support by following the [WAI-ARIA practices](#).

DateTimePicker supports the below list of shortcut keys.

#### Input Navigation

Before opening the popup, use the below list of keys to `DateTimePicker` control the popup element.

| **Press** | **To do this** |

| --- | --- |

| **Alt + Down Arrow** | **Open the select popup** |

| Alt + Down Arrow + Alt + Down Arrow | Toggle between two popups |

#### Calendar Navigation

Use the below list of keys to interact with the Calendar after the DatePicker popup has opened.

| Press | To do this |

| --- | --- |

| Upper Arrow | Focus the previous week date. |

| Down Arrow | Focus the next week date. |

| Left Arrow | Focus the previous date. |

| Right Arrow | Focus the next date. |

| Home | Focus the first date in the month. |

| End | Focus the last date in the month. |

| Page Up | Focus the same date in the previous month. |

| Page Down | Focus the same date in the next month. |

| Enter | Select the currently focused date. |

| Shift + Page Up | Focus the same date in the previous year. |

| Shift + Page Down | Focus the same date in the next year. |

| Control + Upper Arrow | Moves into the inner level of view like month-year, year-decade |

| Control + Down Arrow | Moves out from the depth level view like decade-year, year-month |

| Control + Home | Focus the starting date in the current year. |

| Control + End | Focus the ending date in the current year. |

Use the below list of shortcut keys to interact with the TimePicker after the TimePicker Popup has opened.

| Press | To do this |

| --- | --- |

| Upper Arrow | Navigate and select the previous item. |

| Down Arrow | Navigate and select the next item. |

| Left Arrow | Move the cursor towards arrow key pressed direction. |

| Right Arrow | Move the cursor towards arrow key pressed direction. |

| Home | Navigate and select the first item. |

| End | Navigate and select the last item. |

| Enter | Select the currently focused item and close the popup. |

| Alt + Upper Arrow | Close the popup. |

| Alt + Down Arrow | Open the popup. |

| Esc | Close the popup. |

To focus the DateTimePicker component use the **alt+t** keys.

#### APP.VUE

```
<template>
  <div id="app">
    <div class='wrap'>
      <ejs-datetimepicker placeholder='Select a datetime'
ref="DateInstance"></ejs-datetimepicker>
    </div>
  </div>
</template>
<script>
import Vue from "vue";
import { DateTimePickerPlugin } from "@syncfusion/ej2-vue-calendars";
Vue.use(DateTimePickerPlugin);
export default {
  mounted () {
    let DateObj = this.$refs.DateInstance;
    document.onkeyup = function (e) {
      if (e.altKey && e.keyCode === 84) {
        // press alt+t to focus the control.
        DateObj.$el.focus();
      }
    }
  }
}
</script>
<style>
  @import '../node_modules/@syncfusion/ej2-base/styles/material.css';
  @import '../node_modules/@syncfusion/ej2-buttons/styles/material.css';
  @import '../node_modules/@syncfusion/ej2-inputs/styles/material.css';
  @import '../node_modules/@syncfusion/ej2-popups/styles/material.css';
  @import '../node_modules/@syncfusion/ej2-lists/styles/material.css';
  @import "../node_modules/@syncfusion/ej2-vue-
calendars/styles/material.css";
  .wrap {
    margin: 35px auto;
    width: 240px;
  }
</style>
```

{% previewsample "page.domainurl/code-snippet/datetimepicker/getting-started-cs1" %}

#### Ensuring accessibility

The DateTimePicker component's accessibility levels are ensured through an [accessibility-checker](#) and [axe-core](#) software tools during automated testing.

The accessibility compliance of the DateTimePicker component is shown in the following sample. Open the [sample](#) in a new window to evaluate the accessibility of the DateTimePicker component with accessibility tools.

{% previewsample "https://ej2.syncfusion.com/accessibility/date-time-picker.html" %}

See also

- [Accessibility in Syncfusion Vue components](#)

## Two way binding in Vue Datetimepicker component

Two-way binding can be achieved by using the `v-model` directive in Vue. In the following sample, when you change the value in one DateTimePicker component, v-model will automatically update the value in the other DateTimePicker.

The following example demonstrates how to set the `two-way-binding` in the DateTimePicker.

### APP.VUE

```
<template>
<div id="app">
  <div id="wrapper1">
    <ejs-datetimepicker id="datetimepicker" v-model="date"></ejs-
datetimepicker>
  </div>
  <div id="wrapper2">
    <ejs-datetimepicker id="datetimepicker1" v-model="date"></ejs-
datetimepicker>
  </div>
</div>
</template>
<script>
import Vue from "vue";
import { DateTimePickerPlugin } from "@syncfusion/ej2-vue-calendars";
Vue.use(DateTimePickerPlugin);
export default {
  data: function() {
    return {
      date: new Date()
    };
  }
}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-buttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-
calendars/styles/material.css";
#wrapper1{
  min-width: 250px;
  float: left;
  margin-left: 100px;
}
#wrapper2{
  min-width: 250px;
  float: right;
  margin-right: 100px;
}
</style>
```

```
{% previewsample "page.domainurl/code-snippet/datetimepicker/two-way-cs1" %}
```

### Style appearance in Vue Datetimepicker component

The following content provides the exact CSS structure that can be used to modify the control's appearance based on the user preference.

#### Customizing the appearance of DateTimePicker wrapper element

Use the following CSS to customize the appearance of wrapper element.

`

*/ To specify height and font size /*

```
.e-input-group input.e-input, .e-input-group.e-control-wrapper input.e-input {  
font-size: 20px;  
height: 40px;  
}
```

`

#### Customizing the DateTimePicker icons element

Use the following CSS to customize the DateTimePicker icons element

`

*/ To specify background color and font size /*

```
.e-datetime-wrapper .e-input-group-icon.e-date-icon, .e-datetime-wrapper .e-input-group-icon.e-time-  
icon {  
font-size: 16px;  
background-color: blanchedalmond;  
}
```

`

#### Customizing the time picker popup in the DateTimePicker

Use the following CSS to customize the time picker popup in the DateTimePicker

`

*/ To specify height /*

```
.e-datetimepicker.e-popup {  
height: 100px;  
}
```

`

#### Customizing the Calendar popup of the DateTimePicker

Please check the below section, to customize the style and appearance of the Calendar component in the DateTimePicker

#### [Customizing Calendar's style and appearance](#)



### Full screen mode support in mobiles and tablets

The DateTimePicker component's full-screen mode feature enables users to view the component popup element in full-screen mode on mobile devices with improved visibility and a better user experience. It is important to mention that this feature is exclusively available for mobile and tablet devices in both landscape and portrait orientations. To activate the full screen mode within the DateTimePicker component, simply set the [fullScreenMode](#) API value to `true`. This action will extend the calendar and time popup element to occupy the entire screen on mobile devices.

```
`html
<template>
<div id="app">
<ejs-datetimepicker :fullScreenMode="mobileMode" ></ejs-datetimepicker>
</div>
</template>
<script>
import Vue from 'vue';
import { DateTimePickerPlugin } from '@syncfusion/ej2-vue-calendars';
Vue.use(DateTimePickerPlugin);
export default {
  name: 'app',
  data () {
    return {
      mobileMode : true
    }
  }
}
</script>
`
```

**Default Sample**

12/15/2017 2:00 PM





## How To

Disable the datetimepicker component in Vue Datetimepicker component

To disable the DateTimePicker, use its [enable](#) property to `false`.

### APP.VUE

```
<template>
  <div id="app">
    <div class='wrapper'>
      <ejs-datetimepicker :enabled="enable" :placeholder="waterMark"
    ></ejs-datetimepicker>
    </div>
  </div>
</template>
<script>
import Vue from 'vue';
import { DateTimePickerPlugin } from '@syncfusion/ej2-vue-calendars';
Vue.use(DateTimePickerPlugin);
export default {
  data () {
    return {
      enable : false,
      waterMark: "Select a date and time"
    }
  }
}
</script>
<style>
@import '../node_modules/@syncfusion/ej2-base/styles/material.css';
@import '../node_modules/@syncfusion/ej2-buttons/styles/material.css';
@import '../node_modules/@syncfusion/ej2-inputs/styles/material.css';
@import '../node_modules/@syncfusion/ej2-popups/styles/material.css';
@import '../node_modules/@syncfusion/ej2-lists/styles/material.css';
@import "../node_modules/@syncfusion/ej2-vue-calendars/styles/material.css";
.wrapper {
  max-width: 250px;
  margin: 0 auto;
}
</style>
```

{% previewsample "page.domainurl/code-snippet/datetimepicker/getting-started-cs7" %}

Set the placeholder in Vue Datetimepicker component

The following example demonstrates how to set [placeholder](#) in the DateTimePicker control.

Using `placeholder` you can display a short hint in the input element.

### APP.VUE

```
<template>
  <div id="app">
    <div class='wrapper'>
      <ejs-datetimepicker :placeholder="waterMark" ></ejs-datetimepicker>
    </div>
  </div>
</template>
```

```

<script>
import Vue from 'vue';
import { DateTimePickerPlugin } from '@syncfusion/ej2-vue-calendars';
Vue.use(DateTimePickerPlugin);
export default {
  data () {
    return {
      watermark: "Select a date and time"
    }
  }
}
</script>
<style>
@import '../node_modules/@syncfusion/ej2-base/styles/material.css';
@import '../node_modules/@syncfusion/ej2-buttons/styles/material.css';
@import '../node_modules/@syncfusion/ej2-inputs/styles/material.css';
@import '../node_modules/@syncfusion/ej2-popups/styles/material.css';
@import '../node_modules/@syncfusion/ej2-lists/styles/material.css';
@import "../node_modules/@syncfusion/ej2-vue-calendars/styles/material.css";
.wrapper {
  max-width: 250px;
  margin: 0 auto;
}
</style>

```

{% previewsample "page.domainurl/code-snippet/datetimetypepicker/getting-started-cs8" %}

Customize the datetimetypepicker day header in Vue Datetimetypepicker component

You can change the format of the day that to be displayed in header using [dayHeaderFormat](#) property. By default, the format is **Short**.

You can find the possible formats on below.

Name	Description
----- -----	
<b>Short</b>	Sets the short format of day name (like Su ) in day header.
<b>Narrow</b>	Sets the single character of day name (like S ) in day header.
<b>Abbreviated</b>	Sets the min format of day name (like Sun ) in day header.
<b>Wide</b>	Sets the long format of day name (like Sunday ) in day header.

## APP.VUE

```

<template>
  <div id="container">
    <div id='datetimepicker'>
      <ejs-datetimepicker dayHeaderFormat="Short"
        ref="datetimepickerObj"></ejs-datetimepicker>
    </div>
    <div id="format">
      <label id="custom-input-label">Header Format Types</label>
      <ejs-dropdownlist id="select" :fields = "fields" :index =
        "currentIndex" :dataSource = "items" :popupHeight= "popupHeight" :change=
        "formatHandler">

```

```

        </ejs-dropdownlist>
    </div>
</div>
</template>
<script>
import Vue from "vue";
import { DateTimePickerPlugin } from "@syncfusion/ej2-vue-calendars";
import { DropDownListPlugin } from "@syncfusion/ej2-vue-dropdowns";
Vue.use(DateTimePickerPlugin);
Vue.use(DropDownListPlugin);
export default {
    data () {
        return {
            items: [
                { sizeVal: 'Short' , sizeTxt: 'Short' },
                { sizeVal: 'Narrow' , sizeTxt: 'Narrow' },
                { sizeVal: 'Abbreviated' , sizeTxt: 'Abbreviated' },
                { sizeVal: 'Wide' , sizeTxt: 'Wide' },
            ],
            fields: { text: 'sizeTxt', value: 'sizeVal' },
            popupHeight: 200,
            currentIndex: 0,
            formatHandler: (args) => {
                this.$refs.datetimepickerObj.dayHeaderFormat = args.value;
            }
        }
    }
}
</script>
<style>
    @import "../node_modules/@syncfusion/ej2-base/styles/material.css";
    @import "../node_modules/@syncfusion/ej2-buttons/styles/material.css";
    @import "../node_modules/@syncfusion/ej2-vue-
calendars/styles/material.css";
    @import "../node_modules/@syncfusion/ej2-vue-
dropdowns/styles/material.css";
    #container{
        width: 100%;
    }
    #datetimepicker {
        width: 25%;
        margin-left: 30px;
        margin-top: 30px;
        display: inline-flex;
    }
    #format {
        width: 180px;
        margin-left: 400px;
        display: inline-block;
    }
    #select {
        height: 30px;
        width: 150px;
        margin-top: 10px;
    }
    #custom-input-label{
        font-size: 15px;

```

```
font-weight: bold
}
</style>
```

{% previewsample "page.domainurl/code-snippet/datetimepicker/header-format-cs1" %}

## Diagram

### Getting Started with the Vue Diagram Component in Vue 2

This article provides a step-by-step guide for setting up a Vue 2 project using [Vue-CLI](#) and integrating the Syncfusion Vue Diagram component

#### Prerequisites

[System requirements for Syncfusion Vue UI components](#)

#### Dependencies

The following list of dependencies are required to use the **Diagram** component in your application.

```
`javascript
|-- @syncfusion/ej2-vue-diagrams
|-- @syncfusion/ej2-base
|-- @syncfusion/ej2-data
|-- @syncfusion/ej2-navigations
|-- @syncfusion/ej2-inputs
|-- @syncfusion/ej2-popups
|-- @syncfusion/ej2-buttons
|-- @syncfusion/ej2-lists
|-- @syncfusion/ej2-splitbuttons
|-- @syncfusion/ej2-diagrams
|-- @syncfusion/ej2-vue-base
`
```

#### Setting up the Vue 2 project

To generate a Vue 2 project using Vue-CLI, use the [vue create](#) command. Follow these steps to install Vue CLI and create a new project:

```
`bash
npm install -g @vue/cli
vue create quickstart
cd quickstart
npm run serve
`
```

or

```
`bash
```

```
yarn global add @vue/cli
```

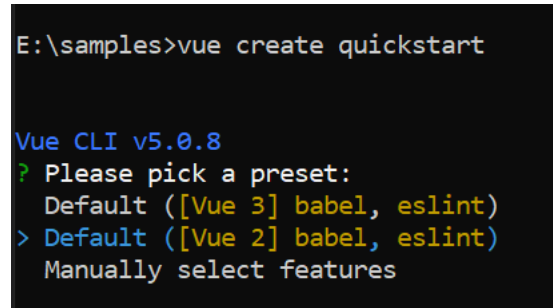
```
vue create quickstart
```

```
cd quickstart
```

```
yarn run serve
```

```
`
```

When creating a new project, choose the option `Default ([Vue 2] babel, eslint)` from the menu.



```
E:\samples>vue create quickstart

Vue CLI v5.0.8
? Please pick a preset:
  Default ([Vue 3] babel, eslint)
> Default ([Vue 2] babel, eslint)
  Manually select features
```

Once the `quickstart` project is set up with default settings, proceed to add Syncfusion components to the project.

#### Add Syncfusion Vue packages

Syncfusion packages are available at [npmjs.com](https://www.npmjs.com). To use Vue components, install the required npm package.

This article uses the [Vue Diagram component](#) as an example. Install the `@syncfusion/ej2-vue-diagrams` package by running the following command:

```
`bash
```

```
npm install @syncfusion/ej2-vue-diagrams --save
```

```
`
```

or

```
`bash
```

```
yarn add @syncfusion/ej2-vue-diagrams
```

```
`
```

#### Import Syncfusion CSS styles

You can import themes for the Syncfusion Vue component in various ways, such as using CSS or SASS styles from npm packages, CDN, [CRG](#) and [Theme Studio](#). Refer to [themes topic](#) to know more about built-in themes and different ways to refer to themes in a Vue project.

In this article, the `Material` theme is applied using CSS styles, which are available in installed packages. The necessary `Material` CSS styles for the Diagram component and its dependents were imported into the `<style>` section of `src/App.vue` file.

**~/SRC/APP.VUE**

```
<style>
@import "../node_modules/@syncfusion/ej2-vue-diagrams/styles/material.css";
</style>
```

**Add Syncfusion Vue component**

Follow the below steps to add the Vue Diagram component:

1\ First, import and register the Diagram component in the **script** section of the **src/App.vue** file.

**~/SRC/APP.VUE**

```
<script>
import { DiagramComponent } from '@syncfusion/ej2-vue-diagrams';
export default {
  components: {
    'ejs-diagram': DiagramComponent
  }
}
</script>
```

2\ In the **template** section, define the Diagram component with [height](#) and [width](#) property.

**~/SRC/APP.VUE**

```
<template>
<div id="app">
<ejs-diagram id="diagram" :width='width' :height='height' ></ejs-diagram>
</div>
</template>
```

3\ Declare the value for the **height** and **width** properties in the **script** section.

**~/SRC/APP.VUE**

```
<script>
data () {
  return {
    width: "100%",
    height: "350px"
  }
}
</script>
```

**Run the project**

To run the project, use the following command:

`bash

npm run serve

,

or

```
`bash
```

```
yarn run serve
```

```
,
```

### APP.VUE

```
<template>
  <div id="app">
    <ejs-diagram id="diagram" :width='width' :height='height' ></ejs-
  diagram>
  </div>
</template>
<script>
import { DiagramComponent } from '@syncfusion/ej2-vue-diagrams';
export default {
  components: {
    'ejs-diagram': DiagramComponent
  },
  name: 'app'
  data () {
    return {
      width: "100%",
      height: "350px"
    }
  }
}
</script>
<style>
  @import "../node_modules/@syncfusion/ej2-vue-
  diagrams/styles/material.css";
</style>
```

```
{% previewsample "page.domainurl/code-snippet/diagram/getting-started/initialize-cs1" %}
```

### Module Injection

Diagram component are segregated into individual feature-wise modules. In order to use a particular feature, you need to inject its feature service in the AppModule. Please find relevant feature service name and description as follows.

- **BpmnDiagrams** - Inject this provider to add built-in BPMN Shapes to diagrams.
- **ConnectorBridging** - Inject this provider to add bridges to connectors.
- **ConnectorEditing** - Inject this provider to edit the segments for connector.
- **ComplexHierarchicalTree** - Inject this provider to complex hierarchical tree like structure.
- **DataBinding** - Inject this provider to populate nodes from given data source.
- **DiagramContextMenu** - Inject this provider to manipulate context menu.
- **HierarchicalTree** - Inject this provider to use hierarchical tree like structure.
- **LayoutAnimation** - Inject this provider animation to layouts.
- **MindMap** - Inject this provider to use mind map.
- **PrintAndExport** - Inject this provider to print or export the objects.
- **RadialTree** - Inject this provider to use Radial tree like structure.
- **Snapping** - Inject this provider to Snap the objects.

- `SymmetricLayout` - Inject this provider to render layout in symmetrical method.
- `UndoRedo` - Inject this provider to revert and restore the changes.

These modules should be imported and injected into the Diagram component using `Diagram.Inject` method as follows.

```
,  
  
<template>  
<div id="app">  
<ejs-diagram id="diagram" :width='width' :height='height' ></ejs-diagram>  
</div>  
</template>  
  
<script>  
import { DiagramComponent, HierarchicalTree, MindMap, RadialTree, ComplexHierarchicalTree,  
  DataBinding, Snapping, PrintAndExport, BpmnDiagrams, SymmetricLayout, ConnectorBridging,  
  UndoRedo, LayoutAnimation, DiagramContextMenu, ConnectorEditing } from '@syncfusion/ej2-vue-  
diagrams';  
export default {  
  components: {  
    'ejs-diagram': DiagramComponent  
  },  
  name: 'app'  
  data () {  
    return {  
      width: "100%",  
      height: "350px"  
    }  
  },  
  provide:{  
    diagram: [BpmnDiagrams, ConnectorBridging, ConnectorEditing, ComplexHierarchicalTree,  
      DataBinding,DiagramContextMenu, HierarchicalTree, LayoutAnimation, MindMap, PrintAndExport,  
      RadialTree, Snapping, SymmetricLayout, UndoRedo]  
  },  
}  
</script>  
,
```



## Flow Diagram

[Create and Add Node](#)

Create and add a **node** (JSON data) with specific position, size, label, and shape.

**APP.VUE**

```
<template>
  <div id="app">
    <ejs-diagram id="diagram" :width='width' :height='height'
:nodes='nodes' ></ejs-diagram>
  </div>
</template>
<script>
import { DiagramComponent } from '@syncfusion/ej2-vue-diagrams';
let nodes = [
  {
    id: "node1",
    height: 60,
    offsetX: 300,
    offsetY: 80,
    annotations: [
      {
        content: "start"
      }
    ]
  }
]
export default {
  components: {
    'ejs-diagram': DiagramComponent
  },
  name: 'app'
  data () {
    return {
      width: "100%",
      height: "350px",
      nodes: nodes,
    }
  }
}
</script>
<style>
  @import "../node_modules/@syncfusion/ej2-vue-
diagrams/styles/material.css";
</style>
```

{% previewsample "page.domainurl/code-snippet/diagram/getting-started/addnode-cs1" %}

[Connect Nodes](#)

Add two node to the diagram as shown in the previous example. Connect these nodes by adding a connector using the **connector** property and refer the source and target end by using the **sourceNode** and **targetNode** properties.

**APP.VUE**

```
<template>
```

```

<div id="app">
  <ejs-diagram id="diagram" :width='width' :height='height'
:nodes='nodes' :connectors='connectors' ></ejs-diagram>
</div>
</template>
<script>
import { DiagramComponent } from '@syncfusion/ej2-vue-diagrams';
let nodes = [
  {
    id: "node1",
    height: 100,
    width: 100,
    offsetX: 200,
    offsetY: 100,
  },
  {
    id: "node2",
    height: 100,
    width: 100,
    offsetX: 200,
    offsetY: 300,
  }
];
let connectors = [
  {
    id: "connector1",
    sourceID: "node1",
    targetID: "node2"
  },
]
export default {
  components: {
    'ejs-diagram': DiagramComponent
  },
  name: 'app'
  data () {
    return {
      width: "100%",
      height: "350px",
      nodes: nodes,
      connectors: connectors
    }
  }
}
</script>
<style>
  @import "../node_modules/@syncfusion/ej2-vue-
diagrams/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/diagram/getting-started/connectnode-cs1" %}

Default values for all `nodes` and `connectors` can be set using the `getNodeDefaults` and `getConnectorDefaults` properties, respectively. For example, if all nodes have the same width and height, such properties can be moved into `getNodeDefaults`.

*Complete Flow Diagram*

Similarly, the required nodes and connectors can be added to form a complete flow diagram.

**APP.VUE**

```

<template>
  <div id="app">
    <ejs-diagram id="diagram" :width='width' :height='height'
      :nodes='nodes' :connectors='connectors' :getNodeDefaults='getNodeDefaults'
      :getConnectorDefaults='getConnectorDefaults' ></ejs-diagram>
  </div>
</template>
<script>
  import { DiagramComponent } from '@syncfusion/ej2-vue-diagrams';
  let nodes = [
    {
      id: "node1",
      offsetY: 50,
      shape: { type: "Flow", shape: "Terminator" },
      annotations: [
        {
          content: "Start"
        }
      ]
    },
    {
      id: "node2",
      offsetY: 140,
      shape: { type: "Flow", shape: "Process" },
      annotations: [
        {
          content: "var i = 0;"
        }
      ]
    },
    {
      id: "node3",
      offsetY: 230,
      shape: { type: "Flow", shape: "Decision" },
      annotations: [
        {
          content: "i < 10?"
        }
      ]
    },
    {
      id: "node4",
      offsetY: 320,
      shape: { type: "Flow", shape: "PreDefinedProcess" },
      annotations: [
        {
          content: 'print("Hello!!");',
          style: { fill: "white" }
        }
      ]
    }
  ],
  {

```

```

        id: "node5",
        offsetY: 410,
        shape: { type: "Flow", shape: "Process" },
        annotations: [
          {
            content: "i++;";
          }
        ]
      },
      {
        id: "node6",
        offsetY: 500,
        shape: { type: "Flow", shape: "Terminator" },
        annotations: [
          {
            content: "End"
          }
        ]
      }
    ];
    let connectors = [
      {
        id: "connector1",
        sourceID: "node1",
        targetID: "node2"
      },
      {
        id: "connector2",
        sourceID: "node2",
        targetID: "node3"
      },
      {
        id: "connector3",
        sourceID: "node3",
        targetID: "node4",
        annotations: [{ text: "Yes" }]
      },
      {
        id: "connector4",
        sourceID: "node3",
        targetID: "node6",
        labels: [{ text: "No" }],
        segments: [
          { length: 30, direction: "Right" },
          { length: 300, direction: "Bottom" }
        ]
      },
      {
        id: "connector5",
        sourceID: "node4",
        targetID: "node5"
      },
      {
        id: "connector6",
        sourceID: "node5",
        targetID: "node3",
        segments: [

```

```

        { length: 30, direction: "Left" },
        { length: 200, direction: "Top" }
      ]
    }
  ];
  export default {
    components: {
      'ejs-diagram': DiagramComponent
    },
    name: 'app',
    data () {
      return {
        width: "100%",
        height: "600px",
        nodes: nodes,
        connectors: connectors,
        getNodeDefaults: (node) => {
          node.height = 60;
          node.width = 100;
          node.offsetX = 300;
          return node;
        },
        getConnectorDefaults: (obj) => {
          obj.type = 'Orthogonal';
          return obj;
        },
      }
    }
  }
</script>
<style>
  @import "../node_modules/@syncfusion/ej2-vue-
  diagrams/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/diagram/getting-started/flowdiagram-cs1" %}

### Automatic Organization Chart

In the 'Flow Diagram' section, how to create a diagram manually was discussed. This section explains how to create and position the diagram automatically.

#### Business object (Employee information)

Define Employee Information as JSON data. The following code example shows an employee array whose, **Name** is used as a unique identifier and **ReportingPerson** is used to identify the person to whom an employee report to, in the organization.

```

`ts
public data: Object[] = [
{
  Name: "Elizabeth",
  Role: "Director"
},

```

```
{
  Name: "Christina",
  ReportingPerson: "Elizabeth",
  Role: "Manager"
},
{
  Name: "Yoshi",
  ReportingPerson: "Christina",
  Role: "Lead"
},
{
  Name: "Philip",
  ReportingPerson: "Christina",
  Role: "Lead"
},
{
  Name: "Yang",
  ReportingPerson: "Elizabeth",
  Role: "Manager"
},
{
  Name: "Roland",
  ReportingPerson: "Yang",
  Role: "Lead"
},
{
  Name: "Yvonne",
  ReportingPerson: "Yang",
  Role: "Lead"
}
];
、
```

### *Map data source*

You can configure the above "Employee Information" with diagram, so that the nodes and connectors are automatically generated using the mapping properties. The following code example show how `dataSourceSettings` is used to map ID and parent with property name identifiers for employee information.

,

```
<template>
<div id="app">
  <ejs-diagram id="diagram" :width='width' :height='height' :dataSourceSettings='dataSourceSettings'
  ></ejs-diagram>
</div>
</template>
<script>
import { DiagramComponent, HierarchicalTree, DataBinding } from '@syncfusion/ej2-vue-diagrams';
import { DataManager } from "@syncfusion/ej2-data";
export let localdata = [
{
  Name: "Elizabeth",
  Role: "Director"
},
{
  Name: "Christina",
  ReportingPerson: "Elizabeth",
  Role: "Manager"
},
{
  Name: "Yoshi",
  ReportingPerson: "Christina",
  Role: "Lead"
},
{
  Name: "Philip",
  ReportingPerson: "Christina",
  Role: "Lead"
},
},
],
```

```
{
  Name: "Yang",
  ReportingPerson: "Elizabeth",
  Role: "Manager"
},
{
  Name: "Roland",
  ReportingPerson: "Yang",
  Role: "Lead"
},
{
  Name: "Yvonne",
  ReportingPerson: "Yang",
  Role: "Lead"
}
];
export default {
  components: {
    'ejs-diagram': DiagramComponent
  },
  name: 'app',
  data () {
    return {
      width: "100%",
      height: "350px",
      dataSourceSettings: {
        id: 'Name', parentId: 'ReportingPerson',
        dataManager: new DataManager(localdata),
        doBinding: (nodeModel, localdata, diagram) => {
          nodeModel.shape = {
            type: "Text",
            content: (localdata).Role,
          }
        }
      }
    }
  }
}
```



```

}
}
}
}
}
</script>
`

```

### Visualize employee

The following code examples indicate how to define the default appearance of nodes and connectors. The `setNodeTemplate` is used to update each node based on employee data.

#### APP.VUE

```

<template>
  <div id="app">
    <ejs-diagram id="diagram" :width='width' :height='height'
      :getNodeDefaults='getNodeDefaults'
      :getConnectorDefaults='getConnectorDefaults' :layout='layout'
      :dataSourceSettings='dataSourceSettings' ></ejs-diagram>
    </div>
  </template>
<script>
  import { DiagramComponent, HierarchicalTree, DataBinding } from
  '@syncfusion/ej2-vue-diagrams';
  import { DataManager } from '@syncfusion/ej2-data';
  export let localdata = [
    {
      Name: "Elizabeth",
      Role: "Director"
    },
    {
      Name: "Christina",
      ReportingPerson: "Elizabeth",
      Role: "Manager"
    },
    {
      Name: "Yoshi",
      ReportingPerson: "Christina",
      Role: "Lead"
    },
    {
      Name: "Philip",
      ReportingPerson: "Christina",
      Role: "Lead"
    },
    {
      Name: "Yang",
      ReportingPerson: "Elizabeth",
      Role: "Manager"
    },
    {
      Name: "Roland",

```

```

        ReportingPerson: "Yang",
        Role: "Lead"
      },
      {
        Name: "Yvonne",
        ReportingPerson: "Yang",
        Role: "Lead"
      }
    ];
    export default {
      components: {
        'ejs-diagram': DiagramComponent
      },
      name: 'app',
      data () {
        return {
          width: "100%",
          height: "350px",
          getNodeDefaults: (node) => {
            node.height = 60;
            node.width = 100;
            return node;
          },
          getConnectorDefaults: (obj) => {
            obj.type = 'Orthogonal';
            return obj;
          },
          layout: {
            type: "OrganizationalChart",
          },
          dataSourceSettings: {
            id: 'Name', parentId: 'ReportingPerson',
            dataManager: new DataManager(localdata),
            doBinding: (nodeModel, localdata, diagram) => {
              nodeModel.shape = {
                type: "Text",
                content: (localdata).Role,
              }
            }
          }
        }
      },
      provide: {diagram: [DataBinding, HierarchicalTree]},
    }
  }
</script>
<style>
  @import "../node_modules/@syncfusion/ej2-vue-
  diagrams/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/diagram/getting-started/orgchart-cs1" %}

You can refer to our [Vue Diagram](#) feature tour page for its groundbreaking feature representations. You can also explore our [Vue Diagram example](#) that shows how to render the Diagram in Vue.

## Getting Started with the Vue Diagram Component in Vue 3

This article provides a step-by-step guide for setting up a [Vite](#) project with a JavaScript environment and integrating the Syncfusion Vue Diagram component using the [Composition API](#) / [Options API](#).

The **Composition API** is a new feature introduced in Vue.js 3 that provides an alternative way to organize and reuse component logic. It allows developers to write components as functions that use smaller, reusable functions called composition functions to manage their properties and behavior.

The **Options API** is the traditional way of writing Vue.js components, where the component logic is organized into a series of options that define the component's properties and behavior. These options include data, methods, computed properties, watchers, lifecycle hooks, and more.

### Prerequisites

#### [System requirements for Syncfusion Vue UI components](#)

#### Set up the Vite project

A recommended approach for beginning with Vue is to scaffold a project using [Vite](#). To create a new Vite project, use one of the commands that are specific to either NPM or Yarn.

```
`bash
```

```
npm create vite@latest
```

```
`
```

or

```
`bash
```

```
yarn create vite
```

```
`
```

Using one of the above commands will lead you to set up additional configurations for the project as below:

1. Define the project name: We can specify the name of the project directly. Let's specify the name of the project as **my-project** for this article.

```
`bash
```

```
? Project name: » my-project
```

```
`
```

2. Select **Vue** as the framework. It will create a Vue 3 project.

```
`bash
```

```
? Select a framework: » - Use arrow-keys. Return to submit.
```

```
Vanilla
```

```
Vue
```

```
React
```

```
Preact
```

```
Lit
```

Svelte

Others

,

3. Choose **JavaScript** as the framework variant to build this Vite project using JavaScript and Vue.

```
`bash
```

? Select a variant: » - Use arrow-keys. Return to submit.

JavaScript

TypeScript

Customize with create-vue ↗

Nuxt ↗

,

4. Upon completing the aforementioned steps to create the **my-project**, run the following command to install its dependencies:

```
`bash
```

```
cd my-project
```

```
npm install
```

,

or

```
`bash
```

```
cd my-project
```

```
yarn install
```

,

Now that **my-project** is ready to run with default settings, let's add Syncfusion components to the project.

#### Add Syncfusion Vue packages

Syncfusion Vue component packages are available at [npmjs.com](https://www.npmjs.com). To use Syncfusion Vue components in the project, install the corresponding npm package.

This article uses the [Vue Diagram component](#) as an example. To use the Vue Diagram component in the project, the **@syncfusion/ej2-vue-diagrams** package needs to be installed using the following command:

```
`bash
```

```
npm install @syncfusion/ej2-vue-diagrams --save
```

,

or

```
`bash
```

```
yarn add @syncfusion/ej2-vue-diagrams
```

### Import Syncfusion CSS styles

You can import themes for the Syncfusion Vue component in various ways, such as using CSS or SASS styles from npm packages, CDN, [CRG](#) and [Theme Studio](#). Refer to [themes topic](#) to know more about built-in themes and different ways to refer to themes in a Vue project.

In this article, **Material** theme is applied using CSS styles, which are available in installed packages. The necessary **Material** CSS styles for the Diagram component and its dependents were imported into the `<style>` section of **src/App.vue** file.

#### ~/SRC/APP.VUE

```
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-navigations/styles/material.css";
@import "../node_modules/@syncfusion/ej2-buttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-inputs/styles/material.css";
@import "../node_modules/@syncfusion/ej2-popups/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-diagrams/styles/material.css";
</style>
```

The order of importing CSS styles should be in line with its dependency graph.

### Add Syncfusion Vue component

Follow the below steps to add the Vue Diagram component using **Composition API** or **Options API**:

1.First, import and register the Diagram component and its child directives in the **script** section of the **src/App.vue** file. If you are using the **Composition API**, you should add the **setup** attribute to the **script** tag to indicate that Vue will be using the **Composition API**.

#### COMPOSITION API (~/SRC/APP.VUE)

```
<script setup>
import { DiagramComponent as EjsDiagram, NodesDirective as ENodes,
NodeDirective as ENode, ConnectorsDirective as EConnectors,
ConnectorDirective as EConnector,NodeAnnotationsDirective as
ENodeAnnotations, NodeAnnotationDirective as ENodeAnnotation,
ConnectorAnnotationsDirective as
EConnectorAnnotations,ConnectorAnnotationDirective as EConnectorAnnotation}
from '@syncfusion/ej2-vue-diagrams';
</script>
```

#### OPTIONS API (~/SRC/APP.VUE)

```
<script>
import { DiagramComponent, NodesDirective, NodeDirective,
ConnectorsDirective, ConnectorDirective,NodeAnnotationsDirective,
NodeAnnotationDirective,
ConnectorAnnotationsDirective,ConnectorAnnotationDirective } from
'@syncfusion/ej2-vue-diagrams';
//Component registration
export default {
```

```

name: "App",
components: {
  "ejs-diagram": DiagramComponent,
  "e-nodes": NodesDirective,
  "e-node": NodeDirective,
  "e-connectors": ConnectorsDirective,
  "e-connector": ConnectorDirective,
  "e-node-annotations": NodeAnnotationsDirective,
  "e-node-annotation": NodeAnnotationDirective,
  "e-connector-annotations": ConnectorAnnotationsDirective,
  "e-connector-annotation": ConnectorAnnotationDirective
}
}
</script>

```

2. In the **template** section, define the Diagram component with the [dataSource](#) property and column definitions.

#### ~/SRC/APP.VUE

```

<template>
  <ejs-diagram
    id="diagram"
    ref="diagramInstance"
    :width="width"
    :height="height"
    :layout="layout"
    :dataSourceSettings="dataSourceSettings"
    :getNodeDefaults="getNodeDefaults"
    :getConnectorDefaults="getConnectorDefaults"
  >
  </ejs-diagram>
</template>

```

3. Declare the values for the **dataSource** property in the **script** section.

#### COMPOSITION API (~/SRC/APP.VUE)

```

<script setup>
const localdata = [
  {
    Name: "Elizabeth",
    Role: "Editor",
  },
  {
    Name: "Christina",
    ReportingPerson: "Elizabeth",
    Role: "Managing Editor",
  },
  {
    Name: "Yoshi",
    ReportingPerson: "Christina",
    Role: "Assistant Editor",
  },
  {
    Name: "Philip",
  },
]

```

```
ReportingPerson: "Christina",
Role: "Copy Editor",
},
{
Name: "Yang",
ReportingPerson: "Elizabeth",
Role: "Bussiness Editor",
},
{
Name: "Roland",
ReportingPerson: "Yang",
Role: "Assistant Editor",
},
{
Name: "Yvonne",
ReportingPerson: "Yang",
Role: "Editorial Assistant",
},
];
</script>
```

### **OPTIONS API (~SRC/APP.VUE)**

```
<script>
data() {
  return {
    localdata:[
      {
        Name: "Elizabeth",
        Role: "Editor",
      },
      {
        Name: "Christina",
        ReportingPerson: "Elizabeth",
        Role: "Managing Editor",
      },
      {
        Name: "Yoshi",
        ReportingPerson: "Christina",
        Role: "Assistant Editor",
      },
      {
        Name: "Philip",
        ReportingPerson: "Christina",
        Role: "Copy Editor",
      },
      {
        Name: "Yang",
        ReportingPerson: "Elizabeth",
        Role: "Bussiness Editor",
      },
      {
        Name: "Roland",
        ReportingPerson: "Yang",
        Role: "Assistant Editor",
      },
    ],
  };
}
```

```
{
  Name: "Yvonne",
  ReportingPerson: "Yang",
  Role: "Editorial Assistant",
},
],
};
}
```

Here is the summarized code for the above steps in the **src/App.vue** file:

#### **COMPOSITION API (~ /SRC /APP.VUE)**

```
<template>
  <ejs-diagram
    id="diagram"
    :width="width"
    :height="height"
    :layout="layout"
    :dataSourceSettings="dataSourceSettings"
    :getNodeDefaults="getNodeDefaults"
    :getConnectorDefaults="getConnectorDefaults"
  >
  </ejs-diagram>
</template>
<script setup>
import {DiagramComponent as EjsDiagram, DataBinding, HierarchicalTree,} from
"@syncfusion/ej2-vue-diagrams";
import { provide } from "vue";
import { DataManager } from "@syncfusion/ej2-data";
const width="1300px";
const height = "800px";
const localdata = [
{
  Name: "Elizabeth",
  Role: "Editor",
},
{
  Name: "Christina",
  ReportingPerson: "Elizabeth",
  Role: "Managing Editor",
},
{
  Name: "Yoshi",
  ReportingPerson: "Christina",
  Role: "Assistant Editor",
},
{
  Name: "Philip",
  ReportingPerson: "Christina",
  Role: "Copy Editor",
},
{
  Name: "Yang",
  ReportingPerson: "Elizabeth",
```



```
Role: "Bussiness Editor",
},
{
Name: "Roland",
ReportingPerson: "Yang",
Role: "Assistant Editor",
},
{
Name: "Yvonne",
ReportingPerson: "Yang",
Role: "Editorial Assistant",
},
];
const getNodeDefaults= (node) => {
node.height = 60;
node.width = 150;
return node;
};
const getConnectorDefaults= (obj) => {
obj.type = "Orthogonal";
obj.style = {
strokeColor: "#6BA5D7",
fill: "#6BA5D7",
strokeWidth: 2,
};
obj.targetDecorator = {
style: {
fill: "#6BA5D7",
strokeColor: "#6BA5D7",
},
};
return obj;
};
const layout={
type: "OrganizationalChart",
};
const dataSourceSettings= {
id: "Name",
parentId: "ReportingPerson",
dataManager: new DataManager(localdata),
doBinding: (nodeModel, localdata) => {
nodeModel.annotations = [
{
content: localdata.Name,
offset: { x: 0.5, y: 0.2 },
style: { color: "white" },
},
{
content: localdata.Role,
offset: { x: 0.5, y: 0.7 },
style: { color: "white" },
},
];
nodeModel.style = { fill: "#6BA5D7", strokeWidth: 0 };
};
const diagram = [DataBinding, HierarchicalTree];
```

```

provide('diagram', diagram);
</script>
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-navigations/styles/material.css";
@import "../node_modules/@syncfusion/ej2-buttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-inputs/styles/material.css";
@import "../node_modules/@syncfusion/ej2-popups/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-diagrams/styles/material.css";
</style>

```

### OPTIONS API (~SRC/APP.VUE)

```

<template>
<ejs-diagram
id="diagram"
:width="width"
:height="height"
:getNodeDefaults="getNodeDefaults"
:getConnectorDefaults="getConnectorDefaults"
:layout="layout"
:dataSourceSettings="dataSourceSettings"
>
</ejs-diagram>
</template>
<script>
import {DiagramComponent, DataBinding, HierarchicalTree,} from
"@syncfusion/ej2-vue-diagrams";
import { DataManager } from "@syncfusion/ej2-data";
// Component registration
export default {
name: "App",
// Declaring component and its directives
components: {
'ejs-diagram': DiagramComponent,
},
// Bound properties declarations
data() {
return {
data:[
{
Name: "Elizabeth",
Role: "Editor",
},
{
Name: "Christina",
ReportingPerson: "Elizabeth",
Role: "Managing Editor",
},
{
Name: "Yoshi",
ReportingPerson: "Christina",
Role: "Assistant Editor",
},
{
Name: "Philip",

```

```

ReportingPerson: "Christina",
Role: "Copy Editor",
},
{
  Name: "Yang",
  ReportingPerson: "Elizabeth",
  Role: "Bussiness Editor",
},
{
  Name: "Roland",
  ReportingPerson: "Yang",
  Role: "Assistant Editor",
},
{
  Name: "Yvonne",
  ReportingPerson: "Yang",
  Role: "Editorial Assistant",
},
],
width: "1300px",
height: "800px",
getNodeDefaults: (node) => {
  node.height = 60;
  node.width = 150;
  return node;
},
getConnectorDefaults: (obj) => {
  obj.type = "Orthogonal";
  obj.style = {
    strokeColor: "#6BA5D7",
    fill: "#6BA5D7",
    strokeWidth: 2,
  };
  obj.targetDecorator = {
    style: {
      fill: "#6BA5D7",
      strokeColor: "#6BA5D7",
    },
  };
  return obj;
},
layout: {
  type: "OrganizationalChart",
},
dataSourceSettings: {
  id: "Name",
  parentId: "ReportingPerson",
  dataManager: new DataManager(localdata),
  doBinding: (nodeModel, localdata) => {
    nodeModel.annotations = [
      {
        content: localdata.Name,
        offset: { x: 0.5, y: 0.2 },
        style: { color: "white" },
      },
      {
        content: localdata.Role,

```

```
offset: { x: 0.5, y: 0.7 },
style: { color: "white" },
},
];
nodeModel.style = { fill: "#6BA5D7", strokeWidth: 0 };
},
},
};
}
};
</script>
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-inputs/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-diagrams/styles/material.css";
@import "../node_modules/@syncfusion/ej2-navigations/styles/material.css";
</style>
```

### Run the project

To run the project, use the following command:

```
`bash
```

```
npm run dev
```

```
`
```

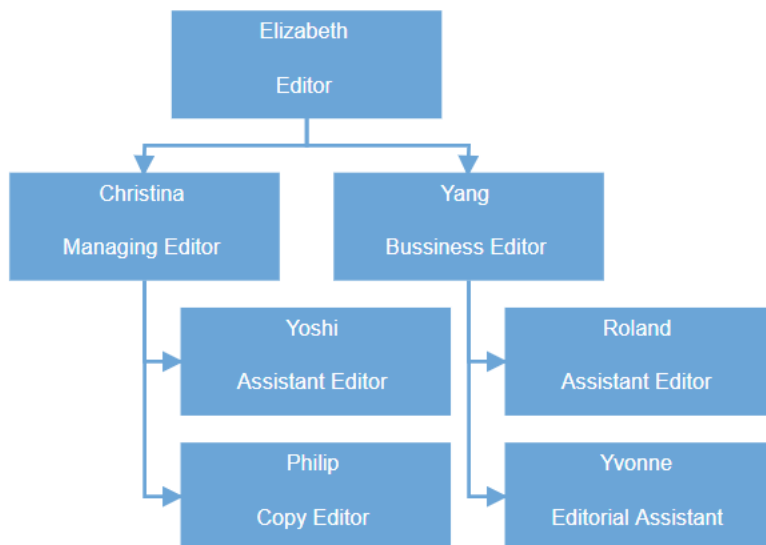
or

```
`bash
```

```
yarn run dev
```

```
`
```

The output will appear as follows:



**Sample:** [vue-3-diagram-getting-started](#).

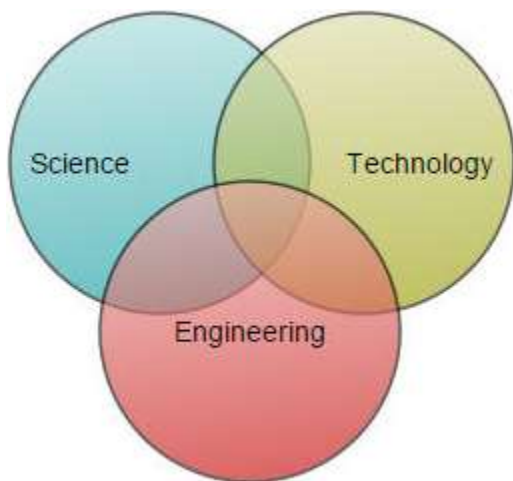
For migrating from Vue 2 to Vue 3, refer to the [migration](#) documentation.

See also

- [Getting Started with Vue UI Components using Composition API and TypeScript](#)
- [Getting Started with Vue UI Components using Options API and TypeScript](#)

### Nodes in Vue Diagram component

Nodes are graphical objects used to visually represent the geometrical information, process flow, internal business procedure, entity, or any other kind of data.



<!-- markdownlint-disable MD033 -->

### Create node

A node can be created and added to the diagram, either programmatically or interactively. Nodes are stacked on the diagram area from bottom to top in the order they are added.

### Add node through nodes collection

To create a node, define the [node](#) object and add that to nodes collection of the diagram model. The following code example illustrates how to add a node to the diagram.

#### APP.VUE

```
<template>
  <div id="app">
    <ejs-diagram id="diagram" :width='width' :height='height'
:nodes='nodes' ></ejs-diagram>
  </div>
</template>
<script>
  import Vue from 'vue';
  import { DiagramPlugin } from '@syncfusion/ej2-vue-diagrams';
  Vue.use(DiagramPlugin);
  let nodes = [{
    // Position of the node
    offsetX: 250,
    offsetY: 250,
    // Size of the node
    width: 100,
    height: 100,
    style: {
      fill: '#6BA5D7',
      strokeColor: 'white'
    },
    // Text(label) added to the node
  }]
  export default {
    name: 'app'
    data() {
      return {
        width: "100%",
        height: "350px",
        nodes: nodes,
      }
    }
  }
</script>
<style>
  @import "../../node_modules/@syncfusion/ej2-vue-
diagrams/styles/material.css";
</style>
```

{% previewsample "page.domainurl/code-snippet/diagram/nodes/Node-cs1" %}

### Add/Remove node at runtime

- Nodes can be added at runtime by using public method, add and can be removed at runtime by using public method,

remove. On adding node at runtime, the nodes collection is changed and the [collectionChange](#) event will trigger.

- The node's ID property is used to define the name of the node and its further used to find the node at runtime and do any customization.

The following code illustrates how to add a node.

#### APP.VUE

```
<template>
  <div id="app">
    <ejs-diagram id="diagram" :width='width' :height='height' ></ejs-
  diagram>
  </div>
</template>
<script>
  import Vue from 'vue';
  import { DiagramPlugin } from '@syncfusion/ej2-vue-diagrams';
  Vue.use(DiagramPlugin);
  let nodes = {
    // Position of the node
    offsetX: 250,
    offsetY: 250,
    // Size of the node
    width: 100,
    height: 100,
    style: {
      fill: '#6BA5D7',
      strokeColor: 'white'
    },
    // Text(label) added to the node
  }
  export default {
    name: 'app'
    data() {
      return {
        width: "100%",
        height: "350px",
      }
    }
    mounted: function() {
      let diagramInstance: Diagram;
      let diagramObj: any = document.getElementById("diagram");
      diagramInstance = diagramObj.ej2_instances[0];
      // Adds to the Diagram
      diagramInstance.add(nodes)
    }
  }
</script>
<style>
  @import "../../node_modules/@syncfusion/ej2-vue-
  diagrams/styles/material.css";
</style>
```

```
{% previewsample "page.domainurl/code-snippet/diagram/nodes/run-cs1" %}
```

### Add node from palette

Nodes can be predefined and added to the palette, and can be dropped into the diagram when needed. For more information

about adding nodes from symbol palette, refer to [Symbol Palette](#).

- Once you drag a node/connector from the palette to the diagram, the following events can be used to do your customization.
- When a symbol is dragged into diagram from symbol palette, the [dragEnter](#) event gets triggered.
- When a symbol is dragged over diagram, the [dragOver](#) event gets triggered.
- When a symbol is dragged and dropped from symbol palette to diagram area, the [drop](#) event gets triggered.
- When a symbol is dragged outside of the diagram, the [dragLeave](#) event gets triggered.

### Create node through data source

Nodes can be generated automatically with the information provided through data source. The default properties for

these nodes are fetched from default settings. For more information about data source, refer to Data Binding.

### Draw nodes

Nodes can be interactively drawn by clicking and dragging the diagram surface by using `NodeDrawingTool`. For more

information about drawing nodes, refer to Draw Nodes.

### Position

- Position of a node is controlled by using its [offsetX](#) and [offsetY](#) properties. By default, these offset properties represent the distance between the origin of the diagram's page and node's center point.
- You may expect this offset values to represent the distance between page origin and node's top-left corner instead of center. The Pivot property helps to solve this problem. Default value of node's [pivot](#) point is (0.5, 0.5), that means center of the node.
- The size of the node can be controlled by using its [width](#) and

[height](#) properties.

- Rotation of a node is controlled by using its [rotateAngle](#) property.

The following table illustrates how pivot relates offset values with node boundaries.

Pivot	Offset
(0.5,0.5)	offsetX and offsetY values are considered as the node's center point.
(0,0)	offsetX and offsetY values are considered as the top-left corner of the node.



| (1,1) | offsetX and offsetY values are considered as the bottom-right corner of the node. |

The following code illustrates how to change the **pivot** value.

### APP.VUE

```
<template>
  <div id="app">
    <ejs-diagram id="diagram" :width='width' :height='height'
:nodes='nodes' ></ejs-diagram>
  </div>
</template>
<script>
  import Vue from 'vue';
  import { DiagramPlugin } from '@syncfusion/ej2-vue-diagrams';
  Vue.use(DiagramPlugin);
  let nodes = [{
    // Position of the node
    offsetX: 250,
    offsetY: 250,
    // Size of the node
    width: 100,
    height: 100,
    pivot: {
      x: 0,
      y: 0
    }
    style: {
      fill: '#6BA5D7',
      strokeColor: 'white'
    },
    // Text(label) added to the node
  }]
  export default {
    name: 'app'
    data() {
      return {
        width: "100%",
        height: "350px",
        nodes: nodes,
      }
    }
    mounted: function() {
      let diagramInstance: Diagram;
      let diagramObj: any = document.getElementById("diagram");
      diagramInstance = diagramObj.ej2_instances[0];
      diagramInstance.select([diagramInstance.nodes[0]]);
    }
  }
</script>
<style>
  @import "../node_modules/@syncfusion/ej2-vue-
diagrams/styles/material.css";
</style>
```

{% previewsample "page.domainurl/code-snippet/diagram/nodes/position-cs1" %}

## Flip

The diagram Provides support to flip the node. [flip](#) is performed to give the mirrored image of the original element.

The flip types are as follows:

- HorizontalFlip

[Horizontal](#) is used to change the element in horizontal direction.

- VerticalFlip

[Vertical](#) is used to change the element in vertical direction

- Both

[Both](#) which involves both vertical and horizontal changes of the element.

The following code illustrates how to provide the mirror image of the original element.

### APP.VUE

```
<template>
  <div id="app">
    <ejs-diagram id="diagram" :width='width' :height='height'
:nodes='nodes' ></ejs-diagram>
  </div>
</template>
<script>
  import Vue from 'vue';
  import { DiagramPlugin } from '@syncfusion/ej2-vue-diagrams';
  Vue.use(DiagramPlugin);
  let nodes = [{
    // Position of the node
    offsetX: 250,
    offsetY: 250,
    // Size of the node
    width: 100,
    height: 100,
    // Flip the node in Horizontal Direction
    flip: 'Horizontal',
    shape: {
      type: 'Basic',
      shape: 'RightTriangle',
    },
    style: {
      fill: '#6BA5D7',
      strokeColor: 'white'
    },
  },
]
export default {
  name: 'app'
  data() {
    return {
```

```

        width: "100%",
        height: "350px",
        nodes: nodes,
      }
    }
    mounted: function() {
      let diagramInstance: Diagram;
      let diagramObj: any = document.getElementById("diagram");
      diagramInstance = diagramObj.ej2_instances[0];
      diagramInstance.select([diagramInstance.nodes[0]]);
    }
  }
</script>
<style>
  @import "../node_modules/@syncfusion/ej2-vue-
  diagrams/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/diagram/nodes/position-cs2" %}

Note: The flip is also applicable for group and BPMN shapes.

### Appearance

- The appearance of a node can be customized by changing its [fill](#) color, [borderColor](#), [borderWidth](#), [strokeDashArray](#),

[opacity](#), and [shadow](#).

- The [visible](#) property of the node enables or disables the visibility of the node.

The following code illustrates how to customize the appearance of the shape.

### APP.VUE

```

<template>
  <div id="app">
    <ejs-diagram id="diagram" :width='width' :height='height'
    :nodes='nodes' ></ejs-diagram>
  </div>
</template>
<script>
  import Vue from 'vue';
  import { DiagramPlugin } from '@syncfusion/ej2-vue-diagrams';
  Vue.use(DiagramPlugin);
  let nodes = [{
    // Position of the node
    offsetX: 250,
    offsetY: 250,
    // Size of the node
    width: 100,
    height: 100,
    style: {
      fill: '#6BA5D7',
      strokeDashArray: '5,5'
    }
  }];

```

```

    },
    borderWidth: 2,
    borderColor: 'red',
    // Text(label) added to the node
  ]
}
export default {
  name: 'app'
  data() {
    return {
      width: "100%",
      height: "350px",
      nodes: nodes,
    }
  }
}
</script>
<style>
  @import "../node_modules/@syncfusion/ej2-vue-
  diagrams/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/diagram/nodes/Appear-cs1" %}

## Gradient

The [gradient](#) property of the node allows you to define and apply the gradient effect to that node.

The gradient stop property defines the color and a position, where the previous color transition ends and a new color transition starts.

The gradient stop's opacity property defines the transparency level of the region.

There are two types of gradients as follows:

- Linear gradient
- Radial gradient

## Linear gradient

- [LinearGradient](#) defines a smooth transition between a set of colors (so-called stops) on a line.
- A linear gradient's x1, y1, x2, y2 properties are used to define the position (relative to the node) of the rectangular region that needs to be painted.

## APP.VUE

```

<template>
  <div id="app">
    <ejs-diagram id="diagram" :width='width' :height='height'
    :nodes='nodes' ></ejs-diagram>
  </div>
</template>
<script>
  import Vue from 'vue';

```

```

import {
DiagramPlugin,GradientModel,LinearGradientModel,RadialGradientModel } from
'@syncfusion/ej2-vue-diagrams';
Vue.use(DiagramPlugin);
let linearGradient: GradientModel | LinearGradientModel |
RadialGradientModel;
linearGradient = {
  //Start point of linear gradient
  x1: 0,
  y1: 0,
  //End point of linear gradient
  x2: 50,
  y2: 50,
  //Sets an array of stop objects
  stops: [{
    color: 'white',
    offset: 0
  },
  {
    color: '#6BA5D7',
    offset: 100
  }
  ],
  type: 'Linear'
};
let nodes = [{
  // Position of the node
  offsetX: 250,
  offsetY: 250,
  // Size of the node
  width: 100,
  height: 100,
  style: {
    gradient: linearGradient
  }
  // Text(label) added to the node
}]
export default {
  name: 'app'
  data() {
    return {
      width: "100%",
      height: "350px",
      nodes: nodes,
    }
  }
}
</script>
<style>
  @import "../../node_modules/@syncfusion/ej2-vue-
diagrams/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/diagram/nodes/Gradient-cs1" %}

## Radial gradient

- [RadialGradient](#) defines a smooth transition between stops on a circle.
- A radial gradient's cx, cy, fx, fy properties are used to define the position (relative to the node) of the outermost or the innermost circle of the radial gradient.

**APP.VUE**

```
<template>
  <div id="app">
    <ejs-diagram id="diagram" :width='width' :height='height'
:nodes='nodes' ></ejs-diagram>
  </div>
</template>
<script>
  import Vue from 'vue';
  import {
DiagramPlugin, GradientModel, LinearGradientModel, RadialGradientModel } from
'@syncfusion/ej2-vue-diagrams';
  Vue.use(DiagramPlugin);
  let radialGradient: GradientModel | LinearGradientModel |
RadialGradientModel;
  radialGradient = {
    //Center point of outer circle
    cx: 50,
    cy: 50,
    //Center point of inner circle
    fx: 25,
    fy: 25,
    //Radius of a radial gradient
    r: 50,
    //Sets an array of stop objects
    stops: [{
      color: 'white',
      offset: 0
    },
    {
      color: '#6BA5D7',
      offset: 100
    }
  ],
    type: 'Radial'
  };
  let nodes = [{
    // Position of the node
    offsetX: 250,
    offsetY: 250,
    // Size of the node
    width: 100,
    height: 100,
    style: {
      gradient: radialGradient
    }
    // Text(label) added to the node
  }]
  export default {
```

```

    name: 'app'
    data() {
      return {
        width: "100%",
        height: "350px",
        nodes: nodes,
      }
    }
  }
</script>
<style>
  @import "../node_modules/@syncfusion/ej2-vue-
  diagrams/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/diagram/nodes/Gradient-cs2" %}

### Shadow

Diagram provides support to add [shadow](#) effect to a node that is disabled, by default. It can be enabled with the constraints property of the node. The following code illustrates how to drop shadow.

### APP.VUE

```

<template>
  <div id="app">
    <ejs-diagram id="diagram" :width='width' :height='height'
    :nodes='nodes' ></ejs-diagram>
  </div>
</template>
<script>
  import Vue from 'vue';
  import { DiagramPlugin, NodeConstraints } from '@syncfusion/ej2-vue-
  diagrams';
  Vue.use(DiagramPlugin);
  let nodes = [{
    // Position of the node
    offsetX: 250,
    offsetY: 250,
    // Size of the node
    width: 100,
    height: 100,
    style: {
      fill: '#6BA5D7',
      strokeColor: 'white'
    },
    constraints: NodeConstraints.Default | NodeConstraints.Shadow,
    // Text(label) added to the node
  }]
  export default {
    name: 'app'
    data() {
      return {
        width: "100%",
        height: "350px",
        nodes: nodes,
      }
    }
  }

```

```

    }
  }
</script>
<style>
  @import "../node_modules/@syncfusion/ej2-vue-
diagrams/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/diagram/nodes/shadow-cs1" %}

### Customizing shadow

The angle, distance, and opacity of the shadow can be customized with the shadow property of the node. The following code example illustrates how to customize shadow.

### APP.VUE

```

<template>
  <div id="app">
    <ejs-diagram id="diagram" :width='width' :height='height'
:nodes='nodes' ></ejs-diagram>
  </div>
</template>
<script>
  import Vue from 'vue';
  import { DiagramPlugin, NodeConstraints } from '@syncfusion/ej2-vue-
diagrams';
  Vue.use(DiagramPlugin);
  let nodes = [{
    // Position of the node
    offsetX: 250,
    offsetY: 250,
    // Size of the node
    width: 100,
    height: 100,
    style: {
      fill: '#6BA5D7',
      strokeColor: 'white'
    },
    constraints: NodeConstraints.Default | NodeConstraints.Shadow,
    shadow: {
      angle: 50,
      opacity: 0.8,
      distance: 9
    }
  }]
  export default {
    name: 'app'
    data() {
      return {
        width: "100%",
        height: "350px",
        nodes: nodes,
      }
    }
  }
</script>

```



```
<style>
  @import "../../node_modules/@syncfusion/ej2-vue-
  diagrams/styles/material.css";
</style>
```

```
{% previewsample "page.domainurl/code-snippet/diagram/nodes/shadow2-cs1" %}
```

### Icon

Diagram provides support to describe the state of the node. i.e., the node is expanded or collapsed state.

Note: Icon can be created only when the node has outEdges.

- To explore the properties of expand and collapse icon, refer to [expandIcon](#) and [collapselIcon](#).
- The expandIcon's and collapseIcon's shape properties allow to define the shape of the icon.

The following code example illustrates how to create an icon of various shapes.

### APP.VUE

```
<template>
  <div id="app">
    <ejs-diagram id="diagram" :width='width' :height='height'
    :nodes='nodes' :connectors='connectors' ></ejs-diagram>
  </div>
</template>
<script>
  import Vue from 'vue';
  import { DiagramPlugin } from '@syncfusion/ej2-vue-diagrams';
  Vue.use(DiagramPlugin);
  let nodes = [{
    id: 'Start',
    width: 140,
    height: 50,
    offsetX: 300,
    offsetY: 50,
    annotations: [{
      content: 'Node1'
    }],
    style: {
      fill: '#6BA5D7',
      strokeColor: 'white'
    },
    expandIcon: {
      shape: 'ArrowDown',
      width: 10,
      height: 10
    },
    collapseIcon: {
      shape: 'ArrowUp',
      width: 10,
      height: 10
    }
  },
  {
    {
```

```

        id: 'Init',
        width: 140,
        height: 50,
        offsetX: 300,
        offsetY: 140,
        style: {
            fill: '#6BA5D7',
            strokeColor: 'white'
        },
        annotations: [{
            content: 'Node2'
        }],
    },
];
let connectors = [{
    // Unique name for the connector
    id: "connector1",
    // Source and Target node's name to which connector needs to be
    // connected.
    sourceID: "Start",
    targetID: "Init",
    type: 'Orthogonal'
}]
export default {
    name: 'app'
    data() {
        return {
            width: "100%",
            height: "350px",
            nodes: nodes,
            connectors: connectors
        }
    }
}
</script>
<style>
    @import "../node_modules/@syncfusion/ej2-vue-
    diagrams/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/diagram/nodes/icon-cs1" %}

### Customizing expand icon

- Set the borderColor, borderWidth, and background color for an expandIcon using borderColor, borderWidth, and fill properties.
- Set a size for an expandIcon by using width and height properties.
- The expand icon can be aligned relative to the node boundaries. It has margin, offset, horizontalAlignment, and verticalAlignment settings. It is quite tricky, when all four alignments are used together but gives you more control over alignment.
- The [iconColor](#) property can be used to set the strokeColor of the Icon.

### Customizing collapse icon

- Set the [borderColor](#),

[borderWidth](#), background color for an collapseIcon using [borderColor](#), [borderWidth](#), and [fill](#) properties.

- Set a size for collapseIcon by using [width](#) and

[height](#) properties.

- Like expand icon, collapse icon also can be aligned relative to the node boundaries. It has margin, offset, horizontalAlignment, and verticalAlignment settings. It is quite tricky, when all four alignments are used together but gives you more control over alignment.
- The [iconColor](#) property can be used to set the strokeColor of the Icon.

### Interaction

Diagram provides support to drag, resize, or rotate the node interactively. For more information about editing a node at runtime, refer to [Edit Nodes](#).

### Constraints

The constraints property of the node allows you to enable/disable certain features. For more information about node constraints, refer to [Node Constraints](#).

### Custom properties

The [addInfo](#) property of the node allows to maintain additional information to the node.

### Stack order

The nodes z-order property specifies the stack order of the node. A node with greater stack order is always in front of a node with a lower stack order.

### Data flow

Node has the InEdges and OutEdges read-only property. In this property, you can find what are all the connectors that are connected to the node, and then you can find these connectors by using the [getObject](#) method in the diagram.

```

\
<template>
<div id="app">
<ejs-diagram id="diagram" :width='width' :height='height' :nodes='nodes' :connectors='connectors'
></ejs-diagram>
</div>
</template>
<script>
import Vue from 'vue';
import { DiagramPlugin } from '@syncfusion/ej2-vue-diagrams';
Vue.use(DiagramPlugin);

```

```
let nodes = [{
  id: 'node1',
  // Position of the node
  offsetX: 450,
  offsetY: 100,
  // Size of the node
  width: 80,
  height: 50,
  style: {
    fill: '#6BA5D7',
    strokeColor: 'white'
  },
},
{
  id: 'node2',
  // Position of the node
  offsetX: 350,
  offsetY: 200,
  // Size of the node
  width: 80,
  height: 50,
  style: {
    fill: '#6BA5D7',
    strokeColor: 'white'
  },
},
{
  id: 'node3',
  // Position of the node
  offsetX: 450,
  offsetY: 200,
  // Size of the node
  width: 80,
```

```
height: 50,
style: {
  fill: '#6BA5D7',
  strokeColor: 'white'
},
},
{
  id: 'node4',
  // Position of the node
  offsetX: 550,
  offsetY: 200,
  // Size of the node
  width: 80,
  height: 50,
  style: {
    fill: '#6BA5D7',
    strokeColor: 'white'
  },
}
];
let connectors = [{
  id: 'connector1',
  sourceID: 'node1',
  targetID: 'node2',
  type: 'Orthogonal'
},
{
  id: 'connector2',
  sourceID: 'node1',
  targetID: 'node3',
  type: 'Orthogonal'
},
{
```

```
id: 'connector3',
sourceID: 'node1',
targetID: 'node4',
type: 'Orthogonal'
}]
export default {
name: 'app'
data() {
return {
width: "100%",
height: "350px",
nodes: nodes,
connectors: connectors
}
}
mounted: function() {
let diagramInstance: Diagram;
let diagramObj: any = document.getElementById("diagram");
diagramInstance = diagramObj.ej2_instances[0];
diagramInstance.getObject('connector1');
}
}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-vue-diagrams/styles/material.css";
</style>
`
```

#### See Also

- [How to add annotations to the node](#)
- [How to add ports to the node](#)
- [How to enable/disable the behavior of the node](#)
- [How to add nodes to the symbol palette](#)
- [How to edit the node visual interface](#)
- [How to create diagram nodes using drawing tools](#)

## Shapes in Vue Diagram component

Diagram provides support to add different kind of nodes. They are as follows:

- Text node
- Image node
- HTML node
- Native node
- Basic shapes
- Flow shapes

<!-- markdownlint-disable MD033 -->

<!-- markdownlint-disable MD010 -->

### Text

Texts can be added to the diagram as [text](#) nodes. The shape property of the node allows you to set the type of node and for text nodes, it should be set as **text**. In addition, define the content object that is used to define the text to be added and style is used to customize the appearance of that text. The following code illustrates how to create a text node.

#### APP.VUE

```
<template>
  <div id="app">
    <ejs-diagram id="diagram" :width='width' :height='height'
:nodes='nodes' ></ejs-diagram>
  </div>
</template>
<script>
  import Vue from 'vue';
  import { DiagramPlugin } from '@syncfusion/ej2-vue-diagrams';
  Vue.use(DiagramPlugin);
  let nodes = [{
    // Position of the node
    offsetX: 250,
    offsetY: 250,
    // Size of the node
    width: 100,
    height: 100,
    //Sets type of the shape as text
    shape: {
      type: 'Text',
      content: 'Text Element'
    },
    //Customizes the appearances such as text, font, fill, and stroke.
    style: {
      strokeColor: 'none',
      fill: 'none',
      color: 'black',
      textAlign: 'Center'
    }
  }]
  export default {
    name: 'app'
    data() {
```

```

        return {
            width: "100%",
            height: "350px",
            nodes: nodes,
        }
    }
}
</script>
<style>
    @import "../node_modules/@syncfusion/ej2-vue-
    diagrams/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/diagram/shapes/Text-cs1" %}

### Image

Diagram allows to add images as [image](#) nodes. The shape property of node allows you to set the type of node and for image nodes, it should be set as **image**. In addition, the source property of shape enables you to set the image source.

The following code illustrates how an image node is created.

### APP.VUE

```

<template>
    <div id="app">
        <ejs-diagram id="diagram" :width='width' :height='height'
        :nodes='nodes' ></ejs-diagram>
    </div>
</template>
<script>
    import Vue from 'vue';
    import { DiagramPlugin } from '@syncfusion/ej2-vue-diagrams';
    Vue.use(DiagramPlugin);
    let nodes = [{
        // Position of the node
        offsetX: 250,
        offsetY: 250,
        // Size of the node
        width: 100,
        height: 100,
        // sets the type of the shape as image
        shape: {
            type: 'Image',
            source:
                'https://ej2.syncfusion.com/demos/src/diagram/employees/image16.png'
        },
        //Customizes the appearances such as text, font, fill, and stroke.
        style: {
            fill: 'none'
        }
    }]
    export default {
        name: 'app',
        data() {
            return {

```



```

        width: "100%",
        height: "350px",
        nodes: nodes,
      }
    }
  }
</script>
<style>
  @import "../..//node_modules/@syncfusion/ej2-vue-
  diagrams/styles/material.css";
</style>

```

```
{% previewsample "page.domainurl/code-snippet/diagram/shapes/Image-cs1" %}
```

### Base64 Encoded Image Into The Image Node:

The following code illustrates how add Base64 image into image node.

**APP.VUE**

```
<template>
    <div id="app">
        <ejs-diagram id="diagram" :width='width' :height='height'
:nodes='nodes' ></ejs-diagram>
    </div>
</template>
<script>
import Vue from 'vue';
import { DiagramPlugin } from '@syncfusion/ej2-vue-diagrams';
Vue.use(DiagramPlugin);
let nodes = [{
    // Position of the node
    offsetX: 250,
    offsetY: 250,
    // Size of the node
    width: 100,
    height: 100,
    // sets the type of the shape as image
    shape: {
        type: 'Image',
        source:
'data:image/gif;base64,R0lGODlhPQBEAPeoAJosM//AwO/AwHVYZ/z595kzAP/s7P+goOXMv
8+fhw/v739/f+8PD98fH/8mJl+fn/9ZWb8/PzWlwv///6wWGbImAPgTEMImIN9gUFCEm/gDALULD
N8PAD6atYdCTX9gUNKlj8wZAKUsAOzz+UMAOsJAP/Z2ccMDA8PD/95eX5NWvsJCOVNQPtfX/8zM
8+QePLl38MGBr8JCP+zs9myn/8GBqwpAP/GxgwJCPny78lzYLgjAJ8vAP9fX/+mjMuCAN8zM/9wc
M8ZGcATEL+QePdZWf/29uc/P9cmJu9MTDImIN+/r7+/vz8/P8VnQGNgV8AAf9fX8swMNgtAF1DO
ICAgPNSUnNWSMQ5MBAQEJE3QPIGAM9AQmqGcG9vb6MhJsEdGM8vLx8fH98AANIWAMuqeL8fABkTE
PPQ0OM5OSydGF15jo+Pj/+pqcsTE78wmFNGQLyMid4dGPvd3UBAQJmTkP+8vH9QUK+vr8ZWSHpzc
JMmILdwcLOGcHRQUHxwcK9PT9DQOO/v70w5MLypoG8wkOuwsP/g4P/QOIcwKEswKMl8aJ9fx2xjd
OtGRs/Pz+Dg4GIIP8gIH0sKEAWKKmTiKZ8aB/f39Wsl+LFt8dgUE9PT5x5aHBwcP+AgP+WltdgY
MyZfyfyz78AAAAAAD///8AAP9mZv///wAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
CH5BAEAAKgALAAAAA9AEQAAaj/AFEJHEiwoMGDCBMqXMiwocAbBww4nEhxoykUpzJGrMixogkfG
UNqlNixJEIDB0SgHGmyJSojMlbkZOmyop0gm30e2liTISKMOoPy7GnwY9CjIYcSRYM0aVKSLmE6n
```

```

fq05QycVLPUhDrxBlCtYJUqNAq2bNWEbj6ZXRUyxZyDRTqwnXvkhACDV+euTeJm1Ki7A73qNWtFi
F+/gA95Gly2CJLDhwEHMOUAAuOpLYDEgBxZ4GRTlC1fDnpkM+fOqD6DDj1aZpITp0dtGCDhr+fVu
Cu3zlg49ijaokTZTo27uG7Gjn2P+hI8+PDPERoUB318bWbfAJ5sUNFcUGRTYUqV/3ogfXp1rWlMc
6awJjiAAAd2fm4ogXjz56aypOoIde4OE5u/F9x199dlXnnGiHZWEYbGpsAEA3QXYnHwEFliKAgsWG
J8LPeiUXGwedCAKABACCN+EAlpYIIYaFlcDhytd51sGAJbo3onOpajiih1092KHGaUXGwWjUBChj
SPiWJuOO/LYIm4v1tXfE6J4gCSJEZ7YgRYUNrkji9P55sF/ogxw5ZkSqIDaZBV6aSGYq/lGZplnd
kckZ98xoICbTcIJGQAZcNmduC210hs35nCyJ58fgmIKX5RQGOZowxaZwYA+JaoKQswGijBV4C6
SiTUmpphMspJx9unX4KaimjDv9aaXOEBteBqmuuxgEHOlX6Kqx+yXqqBANsgCtit4FWQAEkrNbpq
7HSOmtwag5w57GrmlJBASEU18ADjUYb3ADTinIttsgSBloJFFa63bduimuqKB1keqWUhoCSK374w
bujvOSu4QG6UvxBrydcpKsav++Ca6G8A6Pr1x2kVMYHwsVxUALDq/krnrhPSOzXG11UTIoffqGR7
GoI2MAxbv6O2kEG56I7CSlRsEFKfVYovDJoIRTg7sugNRDGqCJzJgcKE0ywc0ELm6KBCCJo8DIPF
eCWNGcyqNFE06ToAfV0HBRgxsvLThHn1oddQMrXj5DyAQgjEHSAJMWZwS3HPxT/QMbabi/iBCliM
LEJkX2EEkomBAUCxRi42VDADxyTYDVogV+wSCHqmKxEKCDAYFDFj4OmwbY7bDGdBhtrntQYOigeC
hUmc1K3QTnAUfEgGFgAWt88hKA6aCRiXhxnQ1yg3BCayK44EWdkUQcBBYEQChFXfCB776aQsG0BI
lQgQgE8qO26X1h8cEUep8ngRBnOy74E9QgRgEAC8SvOfQkh7FDBDmS43PmGoIiKUUEGkMEC/PJHg
xw0xH74yx/3XnaYRJgMB8obxQW6kL9QYEJ0FIFgByfIL7/IQAlvQwEpnAC7DtLNJCKUoO/w45c44
GwCXiAFB/OXAATQryUxdN4LfFiwgjCNYg+kYMIEfkCKDs6PKAIJouyGWMS1FSKJOMRB/BoIxYJIU
XFUxNwoIkEKPAgCBZSQHQ1A2EWDfDEUVLyADj5AchSIQW6gu10bE/JG2VnZCGfo4R4d0sdQoBAHh
PjhIB94v/wRoRkQWGRHgrhGSQJxCS+0pCZbEhAAOw== '
    },
    //Customizes the appearances such as text, font, fill, and stroke.
    style: {
        fill: 'none'
    }
  }
}
export default {
  name: 'app'
  data() {
    return {
      width: "100%",
      height: "350px",
      nodes: nodes,
    }
  }
}
</script>
<style>
  @import "../node_modules/@syncfusion/ej2-vue-
  diagrams/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/diagram/shapes/Image-cs2" %}

Note: Deploy your HTML file in the web application and export the diagram (image node) or else the image node will not be exported in the Chrome and Firefox due to security issues. Refer to the following link.

Link 1: <http://asked.online/draw-images-on-canvas-locally-using-chrome/2546077/>

Link 2: <http://stackoverflow.com/questions/4761711/local-image-in-canvas-in-chrome>

### Image alignment

Stretch and align the image content anywhere but within the node boundary.

The scale property of the node allows to stretch the image as you desired (either to maintain proportion or to stretch). By default, the [scale](#) property of the node is set as **meet**.

The following code illustrates how to scale or stretch the content of the image node.

### APP.VUE

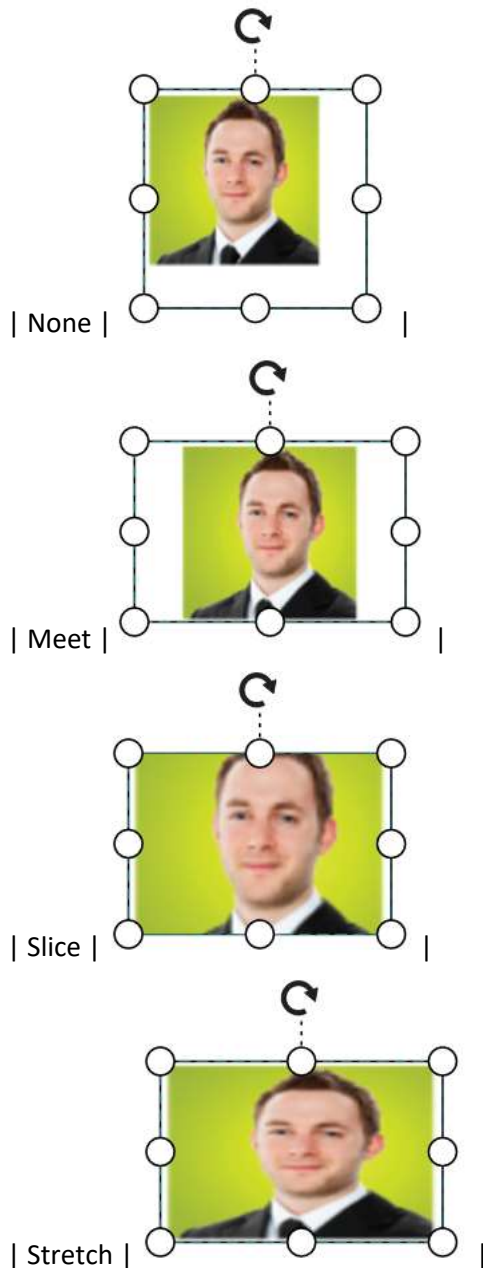
```
<template>
  <div id="app">
    <ejs-diagram id="diagram" :width='width' :height='height'
:nodes='nodes' ></ejs-diagram>
  </div>
</template>
<script>
  import Vue from 'vue';
  import { DiagramPlugin } from '@syncfusion/ej2-vue-diagrams';
  Vue.use(DiagramPlugin);
  let nodes = [{
    // Position of the node
    offsetX: 250,
    offsetY: 250,
    // Size of the node
    width: 100,
    height: 100,
    //sets the type of the shape as Image
    shape: {
      type: 'Image',
      source:
'https://ej2.syncfusion.com/demos/src/diagram/employees/image16.png',
      scale: 'None'
    },
    //Customizes the appearances such as text, font, fill, and stroke.
    style: {
      fill: 'none'
    }
  }
]
export default {
  name: 'app'
  data() {
    return {
      width: "100%",
      height: "350px",
      nodes: nodes,
    }
  }
}
</script>
<style>
  @import "../node_modules/@syncfusion/ej2-vue-
diagrams/styles/material.css";
</style>
```

{% previewsample "page.domainurl/code-snippet/diagram/shapes/HTML-cs1" %}

The following table illustrates all the possible scale options for the image node.

| Values | Images |

|-----| -----|



## HTML

Html elements can be embedded in the diagram through [Html](#) type node. The shape property of node allows you to set the type of node and to create a HTML node it should be set as **HTML**. The following code illustrates how an Html node is created.

## APP.VUE

```
<template>
  <div id="app">
    <ejs-diagram id="diagram" :width='width' :height='height'
:nodes='nodes' ></ejs-diagram>
  </div>
</template>
<script>
```

```

import Vue from 'vue';
import { DiagramPlugin, NodeModel } from '@syncfusion/ej2-vue-diagrams';
Vue.use(DiagramPlugin);
let nodes = [{
  // Position of the node
  offsetX: 250,
  offsetY: 250,
  // Size of the node
  width: 100,
  height: 100,
  style: {
    fill: '#6BA5D7',
    strokeColor: 'white'
  },
  //sets the type of the shape as HTML
  shape: {
    type: 'HTML',
    content: '<div
style="background:#6BA5D7;height:100%;width:100%;"><button type="button"
style="width:100px"> Button</button></div>'
  }
  as NodeModel
}]
export default {
  name: 'app'
  data() {
    return {
      width: "100%",
      height: "350px",
      nodes: nodes,
    }
  }
}
</script>
<style>
  @import "../node_modules/@syncfusion/ej2-vue-diagrams/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/diagram/shapes/HTML-cs2" %}

Note: HTML node cannot be exported to image format, like JPEG, PNG, and BMP. It is by design, while exporting the diagram is drawn in a canvas. Further, this canvas is exported into image formats. Currently, drawing in a canvas equivalent from all possible HTML is not feasible. Hence, this limitation.

### HTML Node With Template

Html elements can be embedded in the diagram through [Html](#) type node. The shape property of node allows you to set the type of node. The following code illustrates how an Html node is created with template.

#### APP.VUE

```

<template>
  <div id="app">
    <ejs-diagram
      ref="diagram"
    >

```

```

        id="diagram"
        :width='width'
        :height='height'
        :nodes = "nodes"
        :nodeTemplate="nodeTemplate"
    >

    </ejs-diagram>
</div>
</template>
<script>
import Vue from "vue";
import { DiagramPlugin, AnnotationConstraints } from "@syncfusion/ej2-vue-
diagrams";
Vue.use(DiagramPlugin);
let itemVue = Vue.component("nodeTemplate", {
    template: `<div style="background:#6BA5D7;height:100%;width:100%;"><button
type="button" style="width:100px"> Button</button></div> `,
    data() {
        return {};
    }
});
let nodes = [
    {
        //Id of the node
        id: "Node",
        //Position of the node
        offsetX: 250,
        offsetY: 250,
        //Size of the node
        width: 100,
        height: 100,
        //sets the type of the shape as HTML
        shape: {
            type: "HTML",
        },
    },
];
export default {
    name: "app",
    data() {
        return {
            width: "100%",
            height: "350px",
            nodes: nodes,
            nodeTemplate: function () {
                return { template: itemVue };
            },
        };
    },
    methods: {
    },
};
</script>

```

{% previewsample "page.domainurl/code-snippet/diagram/shapes/HTML-cs3" %}

## Native

Diagram provides support to embed SVG element into a node. The shape property of node allows you to set the type of node. To create a [native](#) node, it should be set as **native**. The following code illustrates how a native node is created.

### APP.VUE

```
<template>
  <div id="app">
    <ejs-diagram id="diagram" :width='width' :height='height'
:nodes='nodes' ></ejs-diagram>
  </div>
</template>
<script>
  import Vue from 'vue';
  import { DiagramPlugin } from '@syncfusion/ej2-vue-diagrams';
  Vue.use(DiagramPlugin);
  let nodes = [{
    // Position of the node
    offsetX: 250,
    offsetY: 250,
    // Size of the node
    width: 100,
    height: 100,
    //sets the type of the shape as Native
    shape: {
      type: 'Native',
      content: '<g xmlns="http://www.w3.org/2000/svg"> <g
transform="translate(1 1)"><g> <path style="fill:#61443C;"
d="M61.979,435.057c2.645-0.512,5.291-0.853,7.936-1.109c-2.01,1.33-4.472,
1.791-6.827,1.28 C62.726,435.13,62.354,435.072,61.979,435.057z"/><path
style="fill:#61443C;"d="M502.469,502.471h-25.6c0.163-30.757-20.173-57.861-
49.749-66.304 c-5.784-1.581-11.753-2.385-17.749-2.389c-2.425-0.028-
4.849,0.114-7.253,0.427c1.831-7.63,2.747-15.45,2.731-23.296 c0.377-47.729-
34.52-88.418-81.749-95.317c4.274-0.545,8.577-0.83,12.885-
0.853c25.285,0.211,49.448,10.466,67.167,28.504
c17.719,18.039,27.539,42.382,27.297,67.666c0.017,7.846-0.9,15.666-
2.731,23.296c2.405-0.312,4.829-0.455,7.253-0.427
C472.572,434.123,502.783,464.869,502.469,502.471z"/> </g> <path
style="fill:#8B685A;" d="M476.869,502.471h7.536c-0.191-32.558,22.574-
60.747,54.443-67.413
c0.375,0.015,0.747,0.072,1.109,0.171c2.355,0.511,4.817,0.05,6.827-
1.28c1.707-0.085,3.413-0.171,5.12-0.171
c4.59,0,9.166,0.486,13.653,1.451c2.324,0.559,4.775,0.147,6.787-1.141c2.013-
1.288,3.414-3.341,3.879-5.685 c7.68-39.706,39.605-70.228,79.616-
76.117c4.325-0.616,8.687-0.929,13.056-0.939c13.281-
0.016,26.409,2.837,38.485,8.363
c3.917,1.823,7.708,3.904,11.349,6.229c2.039,1.304,4.527,1.705,6.872,1.106c2.
345-0.598,4.337-2.142,5.502-4.264 c14.373-25.502,39.733-42.923,68.693-
47.189h0.171c47.229,6.899,82.127,47.588,81.749,95.317c0.017,7.846-
0.9,15.666-2.731,23.296 c2.405-0.312,4.829-0.455,7.253-
0.427c5.996,0.005,11.965,0.808,17.749,2.389C456.696,444.61,477.033,471.713,4
76.869,502.471 L476.869,502.471z"/> <path style="fill:#66993E;"
d="M502.469,7.537c0,0-6.997,264.96-192.512,252.245c-20.217-1.549-40.166-
5.59-59.392-12.032 c-1.365-0.341-2.731-0.853-4.096-1.28c0,0-0.597-2.219-
1.451-6.144c-6.656-34.048-25.088-198.997,231.765-230.144
C485.061,9.159,493.595,8.22,502.469,7.537z"/> </path

```

```

style="fill:#9ACA5C;" d="M476.784,10.183c-1.28,26.197-16.213,238.165-
166.827,249.6      c-20.217-1.549-40.166-5.59-59.392-12.032c-1.365-0.341-
2.731-0.853-4.096-1.28c0,0-0.597-2.219-1.451-6.144
C238.363,206.279,219.931,41.329,476.784,10.183z"/>      <path
style="fill:#66993E;" d="M206.192,246.727c-0.768,3.925-1.365,6.144-
1.365,6.144c-1.365,0.427-2.731,0.939-4.096,1.28      c-21.505,7.427-
44.293,10.417-66.987,8.789C21.104,252.103,8.816,94.236,7.621,71.452c-0.085-
1.792-0.085-2.731-0.085-2.731
C222.747,86.129,211.653,216.689,206.192,246.727z"/>      <path
style="fill:#9ACA5C;" d="M180.336,246.727c-0.768,3.925-1.365,6.144-
1.365,6.144c-1.365,0.427-2.731,0.939-4.096,1.28      c-13.351,4.412-
27.142,7.359-41.131,8.789C21.104,252.103,8.816,94.236,7.621,71.452
C195.952,96.881,185.541,217.969,180.336,246.727z"/>    </g>    <g>
    <path d="M162.136,426.671c3.451-0.001,6.562-2.08,7.882-5.268s0.591-
6.858-1.849-9.298l-8.533-8.533      c-3.341-3.281-8.701-3.256-12.012,0.054c-
3.311,3.311-3.335,8.671-0.054,12.012l8.533,8.533
C157.701,425.773,159.872,426.673,162.136,426.671z"/>
    <path d="M292.636,398.57c3.341,3.281,8.701,3.256,12.012-0.054c3.311-
3.311,3.335-8.671,0.054-12.012l-8.533-8.533      c-3.341-3.281-8.701-3.256-
12.012,0.054s-3.335,8.671-0.054,12.012L292.636,398.57z"/>      <path
d="M296.169,454.771c-3.341-3.281-8.701-3.256-12.012,0.054c-3.311,3.311-
3.335,8.671-0.054,12.012l8.533,8.533      c3.341,3.281,8.701,3.256,12.012-
0.054c3.311-3.311,3.335-8.671,0.054-12.012L296.169,454.771z"/>
    <path d="M386.503,475.37c3.341,3.281,8.701,3.256,12.012-0.054c3.311-
3.311,3.335-8.671,0.054-12.012l-8.533-8.533      c-3.341-3.281-8.701-3.256-
12.012,0.054c-3.311,3.311-3.335,8.671-0.054,12.012L386.503,475.37z"/>
    <path d="M204.803,409.604c2.264,0.003,4.435-0.897,6.033-
2.518.533-8.533c3.281-3.341,3.256-8.701-0.054-12.012      c-3.311-3.311-8.671-
3.335-12.012-0.054l-8.533,8.533c-2.44,2.44-3.169,6.11-1.849,9.298
C198.241,407.524,201.352,409.603,204.803,409.604z"/>    <path
d="M332.803,443.737c2.264,0.003,4.435-0.897,6.033-2.518.533-8.533c3.281-
3.341,3.256-8.701-0.054-12.012      c-3.311-3.311-8.671-3.335-12.012-0.054l-
8.533,8.533c-2.44,2.44-3.169,6.11-1.849,9.298
C326.241,441.658,329.352,443.737,332.803,443.737z"/>    <path
d="M341.336,366.937c2.264,0.003,4.435-0.897,6.033-2.518.533-8.533c3.281-
3.341,3.256-8.701-0.054-12.012      c-3.311-3.311-8.671-3.335-12.012-0.054l-
8.533,8.533c-2.44,2.44-3.169,6.11-1.849,9.298
C334.774,364.858,337.885,366.937,341.336,366.937z"/>    <path
d="M164.636,454.771l-8.533,8.533c-2.188,2.149-3.055,5.307-
2.27,8.271c0.785,2.965,3.1,5.28,6.065,6.065      c2.965,0.785,6.122-
0.082,8.271-2.27l8.533-8.533c3.281-3.341,3.256-8.701-0.054-12.012
C173.337,451.515,167.977,451.49,164.636,454.771z"/>
    <path d="M232.903,429.171l-8.533,8.533c-2.188,2.149-3.055,5.307-
2.27,8.271c0.785,2.965,3.1,5.28,6.065,6.065      c2.965,0.785,6.122-
0.082,8.271-2.27l8.533-8.533c3.281-3.341,3.256-8.701-0.054-12.012
C241.604,425.915,236.243,425.89,232.903,429.171z"/>
    <path d="M384.003,409.604c2.264,0.003,4.435-0.897,6.033-2.518.533-
8.533c3.281-3.341,3.256-8.701-0.054-12.012      c-3.311-3.311-8.671-3.335-
12.012-0.054l-8.533,8.533c-2.44,2.44-3.169,6.11-1.849,9.298
C377.441,407.524,380.552,409.603,384.003,409.604z"/>    <path
d="M70.77,463.304l-8.533,8.533c-2.188,2.149-3.055,5.307-
2.27,8.271s3.1,5.28,6.065,6.065      c2.965,0.785,6.122-0.082,8.271-
2.27l8.533-8.533c3.281-3.341,3.256-8.701-0.054-12.012
C79.47,460.048,74.11,460.024,70.77,463.304z"/>      <path
d="M121.97,446.238l-8.533,8.533c-2.188,2.149-3.055,5.307-
2.27,8.271c0.785,2.965,3.1,5.28,6.065,6.065      c2.965,0.785,6.122-
0.082,8.271-2.27l8.533-8.533c3.281-3.341,3.256-8.701-0.054-12.012

```



```
C130.67,442.981,125.31,442.957,121.97,446.238L121.97,446.238z"/>
  <path d="M202.302,420.638c-1.6-1.601-3.77-2.5-6.033-2.5c-2.263,0-
4.433,0.899-6.033,2.51-8.533,8.533    c-2.178,2.151-3.037,5.304-
2.251,8.262c0.786,2.958,3.097,5.269,6.055,6.055c2.958,0.786,6.111-
0.073,8.262-2.25118.533-8.533    c1.601-1.6,2.5-3.77,2.5-
6.033C204.802,424.408,203.903,422.237,202.302,420.638L202.302,420.638z"/>
  <path d="M210.836,463.304c-3.341-3.281-8.701-3.256-
12.012,0.054c-3.311,3.311-3.335,8.671-0.054,12.01218.533,8.533
c2.149,2.188,5.307,3.055,8.271,2.27c2.965-0.785,5.28-3.1,6.065-6.065c0.785-
2.965-0.082-6.122-2.27-8.271L210.836,463.304z"/>    <path
d="M343.836,454.7711-8.533,8.533c-2.188,2.149-3.055,5.307-
2.27,8.271c0.785,2.965,3.1,5.28,6.065,6.065    c2.965,0.785,6.122-
0.082,8.271-2.2718.533-8.533c3.281-3.341,3.256-8.701-0.054-12.012
C352.537,451.515,347.177,451.49,343.836,454.771L343.836,454.771z"/>
  <path d="M429.17,483.904c3.341,3.281,8.701,3.256,12.012-0.054s3.335-
8.671,0.054-12.0121-8.533-8.533    c-3.341-3.281-8.701-3.256-12.012,0.054c-
3.311,3.311-3.335,8.671-0.054,12.012L429.17,483.904z"/>    <path
d="M341.336,401.071c2.264,0.003,4.435-0.897,6.033-2.518.533-8.533c3.281-
3.341,3.256-8.701-0.054-12.012    s-8.671-3.335-12.012-0.0541-8.533,8.533c-
2.44,2.441-3.169,6.11-
1.849,9.298C334.774,398.991,337.885,401.07,341.336,401.071z"/>
  <path d="M273.069,435.204c2.264,0.003,4.435-0.897,6.033-2.518.533-
8.533c3.281-3.341,3.256-8.701-0.054-12.012    s-8.671-3.335-12.012-0.0541-
8.533,8.533c-2.44,2.44-3.169,6.11-
1.849,9.298C266.508,433.124,269.618,435.203,273.069,435.204z"/>
  <path
d="M253.318,258.138c22.738,7.382,46.448,11.338,70.351,11.737c31.602,0.543,62
.581-8.828,88.583-26.796    c94.225-65.725,99.567-227.462,99.75-
234.317c0.059-2.421-0.91-4.754-2.667-6.421c-1.751-1.679-4.141-2.52-6.558-
2.308    C387.311,9.396,307.586,44.542,265.819,104.5c-28.443,42.151-
38.198,94.184-26.956,143.776c-3.411,8.366-6.04,17.03-7.852,25.881    c-
4.581-7.691-9.996-14.854-16.147-21.358c8.023-38.158,0.241-77.939-21.57-
110.261C160.753,95.829,98.828,68.458,9.228,61.196    c-2.417-0.214-
4.808,0.628-6.558,2.308c-1.757,1.667-2.726,4-
2.667,6.421c0.142,5.321,4.292,130.929,77.717,182.142
c20.358,14.081,44.617,21.428,69.367,21.008c18.624-0.309,37.097-3.388,54.814-
9.138c11.69,12.508,20.523,27.407,25.889,43.665
c0.149,15.133,2.158,30.19,5.982,44.832c-12.842-5.666-26.723-8.595-40.759-
8.6c-49.449,0.497-91.788,35.567-101.483,84.058    c-5.094-1.093-10.29-1.641-
15.5-1.638c-42.295,0.38-76.303,34.921-76.025,77.217c-
0.001,2.263,0.898,4.434,2.499,6.035
c1.6,1.6,3.771,2.499,6.035,2.499h494.933c2.263,0.001,4.434-0.898,6.035-
2.499c1.6-1.6,2.499-3.771,2.499-6.035    c0.249-41.103-31.914-75.112-72.967-
77.154c0.65-4.78,0.975-9.598,0.975-14.421c0.914-45.674-28.469-86.455-72.083-
100.045    c-43.615-13.59-90.962,3.282-
116.154,41.391C242.252,322.17,242.793,288.884,253.318,258.138L253.318,258.13
8z M87.519,238.092    c-55.35-38.567-67.358-129.25-69.833-
158.996c78.8,7.921,133.092,32.454,161.458,72.992
c15.333,22.503,22.859,49.414,21.423,76.606c-23.253-35.362-77.83-105.726-
162.473-140.577c-2.82-1.165-6.048-0.736-8.466,1.125    s-3.658,4.873-
3.252,7.897c0.406,3.024,2.395,5.602,5.218,6.761c89.261,36.751,144.772,117.77
6,161.392,144.874    C150.795,260.908,115.29,257.451,87.519,238.092z
M279.969,114.046c37.6-53.788,109.708-86.113,214.408-96.138    c-2.65,35.375-
17.158,159.05-91.892,211.175c-37.438,26.116-85.311,30.57-142.305,13.433
c19.284-32.09,92.484-142.574,212.405-191.954c2.819-1.161,4.805-3.738,5.209-
6.76c0.404-3.022-0.835-6.031-3.25-7.892    c-2.415-1.861-5.64-2.292-8.459-
1.131C351.388,82.01,279.465,179.805,252.231,222.711
```

```

C248.573,184.367,258.381,145.945,279.969,114.046L279.969,114.046z
M262.694,368.017c15.097-26.883,43.468-43.587,74.3-43.746
c47.906,0.521,86.353,39.717,85.95,87.625c-0.001,7.188-0.857,14.351-
2.55,21.337c-0.67,2.763,0.08,5.677,1.999,7.774
c1.919,2.097,4.757,3.1,7.568,2.676c1.994-0.272,4.005-0.393,6.017-
0.362c29.59,0.283,54.467,22.284,58.367,51.617H17.661      c3.899-
29.333,28.777-51.334,58.367-51.617c4-
0.004,7.989,0.416,11.9,1.254c4.622,0.985,9.447,0.098,13.417-2.467      c3.858-
2.519,6.531-6.493,7.408-11.017c7.793-40.473,43.043-69.838,84.258-
70.192c16.045-0.002,31.757,4.582,45.283,13.212
c4.01,2.561,8.897,3.358,13.512,2.205C256.422,375.165,260.36,372.163,262.694,
368.017L262.694,368.017z"/>      </g></g>'
    },
  ]
}
export default {
  name: 'app'
  data() {
    return {
      width: "100%",
      height: "350px",
      nodes: nodes,
    }
  }
}
</script>
<style>
  @import "../node_modules/@syncfusion/ej2-vue-
  diagrams/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/diagram/shapes/Native-cs1" %}

Note: Like HTML node, the native node also cannot be exported to image format. Fill color of native node can be overridden by the inline style or fill of the SVG element specified in the template..

### SVG content alignment

Stretch and align the svg content anywhere but within the node boundary.

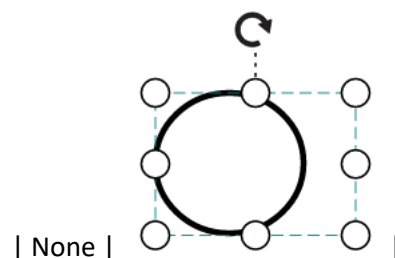
The scale property of the node allows to stretch the svg content as you desired (either to maintain proportion or to stretch). By default, the **scale** property of node is set as **meet**.

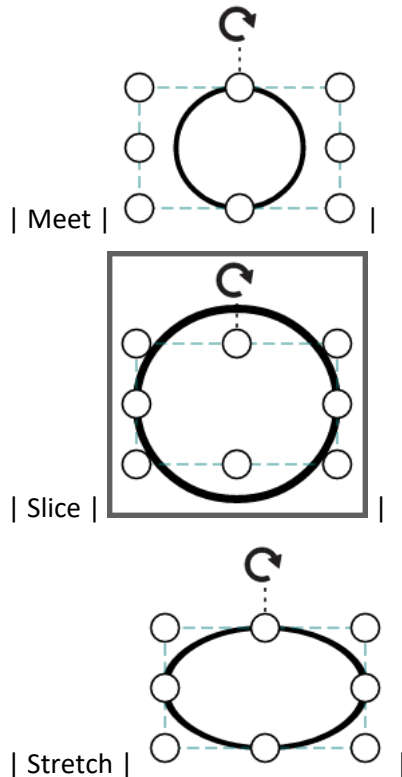
The following code illustrates how to scale or stretch the content of the node.

The following tables illustrates all the possible scale options for the node.

| Values | Images |

|-----| -----|





### Basic shapes

- The [Basic](#) shapes are common shapes that are used to represent the geometrical information visually. To create basic shapes, the type of the shape should be set as **basic**. Its shape property can be set with any one of the built-in shape.
- To render a rounded rectangle, you need to set the type as basic and shape as rectangle. Set the [cornerRadius](#) property to specify the radius of rounded rectangle.

The following code example illustrates how to create a basic shape.

### APP.VUE

```
<template>
  <div id="app">
    <ejs-diagram id="diagram" :width='width' :height='height'
:nodes='nodes' ></ejs-diagram>
  </div>
</template>
<script>
  import Vue from 'vue';
  import { DiagramPlugin } from '@syncfusion/ej2-vue-diagrams';
  Vue.use(DiagramPlugin);
  let nodes = [{
    // Position of the node
    offsetX: 250,
    offsetY: 250,
    // Size of the node
    width: 100,
    height: 100,
```

```

//sets the type of the shape as Basic
shape: {
  type: 'Basic',
  shape: 'Rectangle',
  cornerRadius: 10
},
style: {
  fill: '#6BA5D7',
  strokeColor: 'white'
},
// Text(label) added to the node
}]
export default {
  name: 'app'
  data() {
    return {
      width: "100%",
      height: "350px",
      nodes: nodes,
    }
  }
}
</script>
<style>
  @import "../node_modules/@syncfusion/ej2-vue-
  diagrams/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/diagram/shapes/Basic-cs1" %}

Note: By default, the **shape** property of the node is set as **basic**.

Default property for shape is Rectangle.

Note: When the **shape** is not set for a basic shape, it is considered as a **rectangle**.

The list of basic shapes are as follows.



## Path

The [Path](#) node is a commonly used basic shape that allows visually to represent the geometrical information. To create a path node, specify the shape as **path**. The path property of node allows you to define the path to be drawn. The following code illustrates how a path node is created.

### APP.VUE

```

<template>
  <div id="app">

```

```

    <ejs-diagram id="diagram" :width='width' :height='height'
:nodes='nodes' ></ejs-diagram>
  </div>
</template>
<script>
  import Vue from 'vue';
  import { DiagramPlugin } from '@syncfusion/ej2-vue-diagrams';
  Vue.use(DiagramPlugin);
  let nodes = [{
    // Position of the node
    offsetX: 250,
    offsetY: 250,
    // Size of the node
    width: 100,
    height: 100,
    //sets the type of the shape as Path
    shape: {
      type: 'Path',
      data: 'M35.2441,25 L22.7161,49.9937 L22.7161,0.00657536
L35.2441,25 z M22.7167,25 L-0.00131226,25 M35.2441,49.6337 L35.2441,0.368951
M35.2441,25 L49.9981,25'
    },
  ]
}
export default {
  name: 'app'
  data() {
    return {
      width: "100%",
      height: "350px",
      nodes: nodes,
    }
  }
}
</script>
<style>
  @import "../node_modules/@syncfusion/ej2-vue-
diagrams/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/diagram/shapes/Path-cs1" %}

### Flow Shapes

The [flow](#) shapes are used to represent the process flow. It is used for analyzing, designing, and managing for documentation process. To create a flow shape, specify the shape type as **flow**. Flow shapes and by default, it is considered as **process**. The following code example illustrates how to create a flow shape.

#### APP.VUE

```

<template>
  <div id="app">
    <ejs-diagram id="diagram" :width='width' :height='height'
:nodes='nodes' ></ejs-diagram>
  </div>
</template>
<script>
  import Vue from 'vue';

```

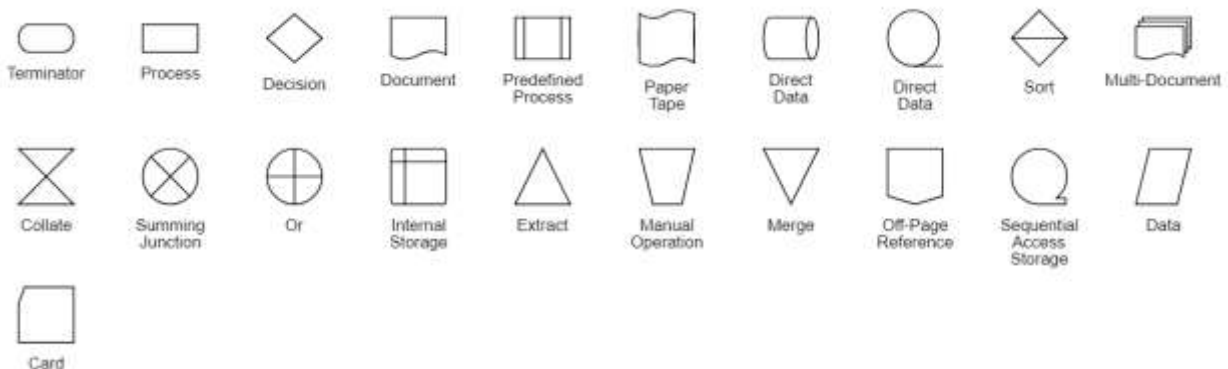
```

import { DiagramPlugin } from '@syncfusion/ej2-vue-diagrams';
Vue.use(DiagramPlugin);
let nodes = [{
  // Position of the node
  offsetX: 250,
  offsetY: 250,
  // Size of the node
  width: 100,
  height: 100,
  //sets the type of the shape as Flow
  shape: {
    type: 'Flow',
    shape: 'Document'
  },
  style: {
    fill: '#6BA5D7',
    strokeColor: 'white'
  },
},
]
export default {
  name: 'app'
  data() {
    return {
      width: "100%",
      height: "350px",
      nodes: nodes,
    }
  }
}
</script>
<style>
  @import "../../node_modules/@syncfusion/ej2-vue-diagrams/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/diagram/shapes/Flow-cs1" %}

The list of flow shapes are as follows.



### Bpmn shapes in Vue Diagram component

BPMN shapes are used to represent the internal business procedure in a graphical notation and enable you to communicate the procedures in a standard manner. To create a BPMN shape, in the node

property shape, type should be set as “bpmn” and its shape should be set as any one of the built-in shapes. The following code example illustrates how to create a simple business process.

Note: If you want to use BPMN shapes in diagram, you need to inject BpmnDiagrams in the diagram.

### APP.VUE

```
<template>
  <div id="app">
    <ejs-diagram id="diagram" :width='width' :height='height'
:nodes='nodes' ></ejs-diagram>
  </div>
</template>
<script>
  import Vue from 'vue';
  import { DiagramPlugin,BpmnDiagrams,Diagram} from '@syncfusion/ej2-vue-
diagrams';
  Diagram.Inject(BpmnDiagrams);
  Vue.use(DiagramPlugin);
  let nodes = [{
    // Position of the node
    offsetX: 250,
    offsetY: 250,
    // Size of the node
    width: 100,
    height: 100,
    style: {
      fill: '#6BA5D7',
      strokeColor: 'white'
    },
    // Text(label) added to the node
  }]
  export default {
    name: 'app'
    data() {
      return {
        width: "100%",
        height: "350px",
        nodes: nodes,
      }
    }
  }
</script>
<style>
  @import "../node_modules/@syncfusion/ej2-vue-
diagrams/styles/material.css";
</style>
```

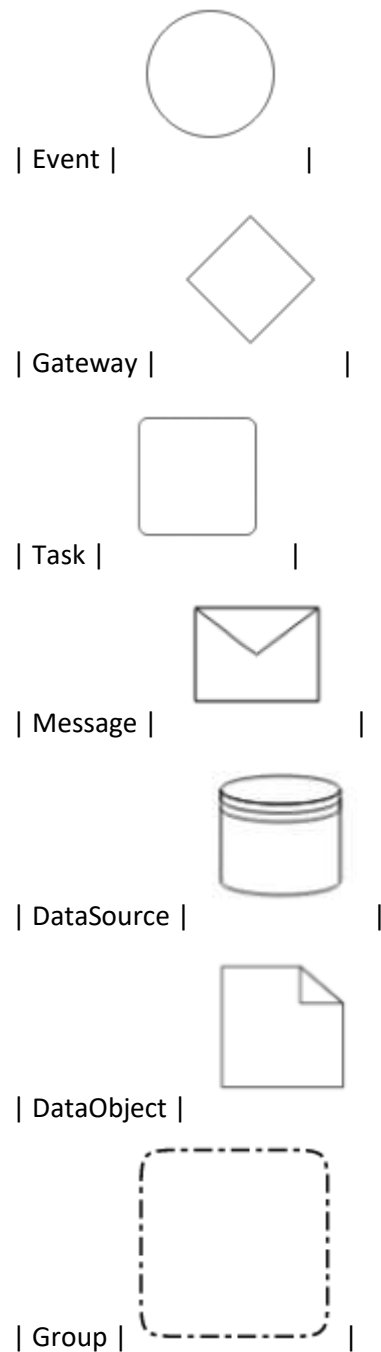
{% previewsample "page.domainurl/code-snippet/diagram/bpmn-shapes/bpmn-cs1" %}

Note : The default value for the property **shape** is “event”.

The list of BPMN shapes are as follows:

| Shape | Image |

| ----- | ----- |



The BPMN shapes and its types are explained as follows.

#### Event

An [event](#) is notated with a circle and it represents an event in a business process. The type of events are as follows:

- Start
- End
- Intermediate



The [event](#) property of the node allows you to define the type of the event. The default value of the event is **start**. The following code example illustrates how to create a BPMN event.

#### APP.VUE

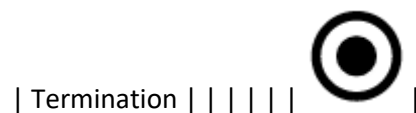
```
<template>
  <div id="app">
    <ejs-diagram id="diagram" :width='width' :height='height'
:nodes='nodes' ></ejs-diagram>
  </div>
</template>
<script>
  import Vue from 'vue';
  import { DiagramPlugin,BpmnDiagrams,Diagram } from '@syncfusion/ej2-vue-
diagrams';
  Diagram.Inject(BpmnDiagrams);
  Vue.use(DiagramPlugin);
  let nodes = [{
    // Position of the node
    offsetX: 250,
    offsetY: 250,
    // Size of the node
    width: 100,
    height: 100,
    //Sets type as Bpmn and shape as event
    shape: {
      type: 'Bpmn',
      shape: 'Event',
      // Sets event as End and trigger as None
      event: {
        event: 'End',
        trigger: 'None'
      }
    },
  },
]
export default {
  name: 'app'
  data() {
    return {
      width: "100%",
      height: "350px",
      nodes: nodes,
    }
  }
}
</script>
<style>
  @import "../node_modules/@syncfusion/ej2-vue-
diagrams/styles/material.css";
</style>
```

{% previewsample "page.domainurl/code-snippet/diagram/bpmn-shapes/Event-cs1" %}

Event triggers are notated as icons inside the circle and they represent the specific details of the process. The [trigger](#) property of the node allows you to set the type of trigger and by default, it is set as **none**. The following table illustrates the type of event triggers.

| Triggers | Start | Non-Interrupting Start | Intermediate | Non-Interrupting Intermediate | Throwing Intermediate | End |

| ----- | ----- | ----- | ----- | ----- | ----- | ----- |





### Gateway

Gateway is used to control the flow of a process and it is represented as a diamond shape. To create a gateway, the shape property of the node should be set as [gateway](#) and the gateway property can be set with any of the appropriate gateways. The following code example illustrates how to create a BPMN Gateway.

### APP.VUE

```
<template>
  <div id="app">
    <ejs-diagram id="diagram" :width='width' :height='height'
      :nodes='nodes' ></ejs-diagram>
    </div>
  </template>
<script>
  import Vue from 'vue';
  import { DiagramPlugin, BpmnDiagrams, Diagram, BpmnGatewayModel } from
    '@syncfusion/ej2-vue-diagrams';
  Diagram.Inject(BpmnDiagrams);
  Vue.use(DiagramPlugin);
  let nodes = [{
    // Position of the node
    offsetX: 250,
    offsetY: 250,
    // Size of the node
    width: 100,
    height: 100,
    //Sets type as Bpmn and shape as Gateway
    shape: {
      type: 'Bpmn',
      shape: 'Gateway',
      //Sets type of the gateway as None
      gateway: {
        type: 'None'
      }
    },
    as BpmnGatewayModel
  ],
  export default {
    name: 'app'
    data() {
      return {
        width: "100%",
        height: "350px",
        nodes: nodes,
      }
    }
  }
</script>
<style>
  @import "../node_modules/@syncfusion/ej2-vue-
    diagrams/styles/material.css";
```

```
</style>
```

{% previewsample "page.domainurl/code-snippet/diagram/bpmn-shapes/Gateway-cs1" %}

Note: By default, the **gateway** will be set as **none**.

There are several types of gateways as tabulated:

Shape	Image
-------	-------

-----	-----
-------	-------

Exclusive		
-----------	---	--


Parallel		
----------	---	--

Inclusive		
-----------	--	--

Complex		
---------	---	--

EventBased		
------------	---	--

ExclusiveEventBased		
---------------------	---	--

ParallelEventBased		
--------------------	---	--

### Activity

The [activity](#) is the task that is performed in a business process. It is represented by a rounded rectangle.

There are two types of activities. They are listed as follows:

- Task: Occurs within a process and it is not broken down to a finer level of detail.
- Subprocess: Occurs within a process and it is broken down to a finer level of detail.

To create a BPMN activity, set the shape as **activity**. You also need to set the type of the BPMN activity by using the activity property of the node. By default, the type of the activity is set as **task**. The following code example illustrates how to create an activity.

#### APP.VUE

```
<template>
  <div id="app">
    <ejs-diagram id="diagram" :width='width' :height='height'
:nodes='nodes' ></ejs-diagram>
  </div>
</template>
<script>
  import Vue from 'vue';
  import { DiagramPlugin,BpmnDiagrams,Diagram,BpmnGatewayModel } from
'@syncfusion/ej2-vue-diagrams';
  Diagram.Inject(BpmnDiagrams);
  Vue.use(DiagramPlugin);
  let nodes = [{
    offsetX: 250,
    offsetY: 250,
    // Size of the node
    width: 100,
    height: 100,
    //Sets type as Bpmn and shape as Activity
    shape: {
      type: 'Bpmn',
      shape: 'Activity',
      //Sets activity as Task
      activity: {
        activity: 'Task'
      },
    },
  ],
}]
export default {
  name: 'app'
  data() {
    return {
      width: "100%",
      height: "350px",
      nodes: nodes,
    }
  }
}
</script>
<style>
  @import "../node_modules/@syncfusion/ej2-vue-
diagrams/styles/material.css";
</style>
```

```
{% previewsample "page.domainurl/code-snippet/diagram/bpmn-shapes/Activity-cs1" %}
```

The different activities of BPMN process are listed as follows.

### Tasks

The [task](#) property of the node allows you to define the type of task such as sending, receiving, user based task, etc. By

default, the type property of task is set as **none**. The following code illustrates how to create different types of

BPMN tasks.

The [type](#) property of tasks allows to represent these results as an event attached to the task.

### APP.VUE

```
<template>
  <div id="app">
    <ejs-diagram id="diagram" :width='width' :height='height'
      :nodes='nodes' ></ejs-diagram>
    </div>
  </template>
<script>
  import Vue from 'vue';
  import { DiagramPlugin, BpmnDiagrams, Diagram, BpmnGatewayModel } from
  '@syncfusion/ej2-vue-diagrams';
  Diagram.Inject(BpmnDiagrams);
  Vue.use(DiagramPlugin);
  let nodes = [{
    // Position of the node
    offsetX: 250,
    offsetY: 250,
    // Size of the node
    width: 100,
    height: 100,
    //Sets type as Bpmn and shape as Activity
    shape: {
      type: 'Bpmn',
      shape: 'Activity',
      //Sets activity as Task
      activity: {
        activity: 'Task',
        //Sets the type of the task as Send
        task: {
          type: 'Send'
        }
      }
    },
  ],
  export default {
    name: 'app'
    data() {
      return {
        width: "100%",
        height: "350px",
        nodes: nodes,
      }
    }
  }
```

```
    }  
  }  
</script>  
<style>  
  @import ".../node_modules/@syncfusion/ej2-vue-  
diagrams/styles/material.css";  
</style>
```

{% previewsample "page.domainurl/code-snippet/diagram/bpmn-shapes/Task-cs1" %}

The various types of BPMN tasks are tabulated as follows.

| Shape | Image |

| ----- | ----- |



| Service | |



| Send | |



| Receive | |



| Instantiating Receive | |



| Manual |



| Business Rule |



| User |



| Script |

### Subprocess

A [sub-process](#) is a group of tasks, which is used to hide or reveal details of additional levels using the [collapsed](#) property.

### APP.VUE

```
<template>
  <div id="app">
    <ejs-diagram id="diagram" :width='width' :height='height'
:nodes='nodes' ></ejs-diagram>
  </div>
</template>
<script>
  import Vue from 'vue';
  import {
DiagramPlugin,BpmnDiagrams,Diagram,BpmnGatewayModel,BpmnSubProcessModel }
from '@syncfusion/ej2-vue-diagrams';
  Diagram.Inject(BpmnDiagrams);
  Vue.use(DiagramPlugin);
  let nodes = [{
    // Position of the node
    offsetX: 250,
```



```

        offsetY: 250,
        // Size of the node
        width: 100,
        height: 100,
        //Sets type as Bpmn and shape as Activity
        shape: {
            type: 'Bpmn',
            shape: 'Activity',
            //Sets activity as SubProcess and collapsed of subprocess as
true
            activity: {
                activity: 'SubProcess',
                subprocess: {
                    collapsed: true
                }
            }
        },
    ],
    export default {
        name: 'app'
        data() {
            return {
                width: "100%",
                height: "350px",
                nodes: nodes,
            }
        }
    }
</script>
<style>
    @import "../node_modules/@syncfusion/ej2-vue-
diagrams/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/diagram/bpmn-shapes/Subprocess-cs1" %}

The different types of subprocess are as follows:

- Event subprocess
- Transaction

### Event Subprocess

A subprocess is defined as an event subprocess, when it is triggered by an event. An event subprocess is placed within another subprocess which is not part of the normal flow of its parent process. You can set event to a subprocess with the [event](#) and [trigger](#) property of the subprocess. The [type](#) property of subprocess allows you to define the type of subprocess whether it should be event subprocess or transaction subprocess.

### APP.VUE

```

<template>
    <div id="app">
        <ejs-diagram id="diagram" :width='width' :height='height'
:nodes='nodes' ></ejs-diagram>
    </div>
</template>

```

```

    </div>
  </template>
  <script>
    import Vue from 'vue';
    import {
      DiagramPlugin, BpmnDiagrams, Diagram, BpmnGatewayModel, BpmnSubProcessModel
    } from '@syncfusion/ej2-vue-diagrams';
    Diagram.Inject(BpmnDiagrams);
    Vue.use(DiagramPlugin);
    let nodes = [{
      // Position of the node
      offsetX: 250,
      offsetY: 250,
      // Size of the node
      width: 100,
      height: 100,
      //Sets type as Bpmn and shape as activity
      shape: {
        type: 'Bpmn',
        shape: 'Activity',
        //Sets activity as SubProcess
        activity: {
          activity: 'SubProcess',
          //Sets the collapsed as true and type as Event
          subprocess: {
            collapsed: true,
            type: 'Event',
            //Sets event as Start and trigger as Message
            event: {
              event: 'Start',
              trigger: 'Message'
            }
          }
        }
      },
      as: BpmnSubProcessModel
    },
    ],
  ];
  export default {
    name: 'app',
    data() {
      return {
        width: "100%",
        height: "350px",
        nodes: nodes,
      }
    }
  }
  </script>
  <style>
    @import "../../node_modules/@syncfusion/ej2-vue-diagrams/styles/material.css";
  </style>

```

{% previewsample "page.domainurl/code-snippet/diagram/bpmn-shapes/EventSub-cs1" %}

## Transaction Subprocess

- [transaction](#) is a set of activities that logically belong together, in which all contained activities must complete their parts of the transaction; otherwise the process is undone. The execution result of a transaction is one of Successful Completion, Unsuccessful Completion (Cancel), and Hazard (Exception). The [events](#) property of subprocess allows to represent these results as an event attached to the subprocess.
- The event object allows you to define the type of event by which the subprocess will be triggered. The name of the event can be defined to identify the event at runtime.
- The event's offset property is used to set the fraction/ratio (relative to parent) that defines the position of the event shape.
- The trigger property defines the type of the event trigger.
- You can also use define ports and labels to subprocess events by using event's ports and labels properties.

### APP.VUE

```
<template>
  <div id="app">
    <ejs-diagram id="diagram" :width='width' :height='height'
:nodes='nodes' ></ejs-diagram>
  </div>
</template>
<script>
  import Vue from 'vue';
  import {
DiagramPlugin, BpmnDiagrams, Diagram, BpmnGatewayModel, BpmnSubProcessModel }
from '@syncfusion/ej2-vue-diagrams';
  Diagram.Inject(BpmnDiagrams);
  Vue.use(DiagramPlugin);
  let nodes = [{
    // Position of the node
    offsetX: 250,
    offsetY: 250,
    // Size of the node
    width: 100,
    height: 100,
    //Sets type as Bpmn and shape as activity
    shape: {
      type: 'Bpmn',
      shape: 'Activity',
      //Sets activity as SubProcess
      activity: {
        activity: 'SubProcess',
        //Sets collapsed as true and type as Transition
        subprocess: {
          collapsed: true,
          type: 'Transaction',
          //Sets event as Intermediate and trigger as Cancel
          event: [{
            event: 'Intermediate',
            trigger: 'Cancel',
            offset: {
              x: 0.25,
```

```

        y: 1
      }
    },
    {
      event: 'Intermediate',
      trigger: 'Error',
      offset: {
        x: 0.25,
        y: 1
      }
    },
  ],
  as BpmnSubProcessModel
},
],
export default {
  name: 'app'
  data() {
    return {
      width: "100%",
      height: "350px",
      nodes: nodes,
    }
  }
}
</script>
<style>
  @import "../node_modules/@syncfusion/ej2-vue-
  diagrams/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/diagram/bpmn-shapes/TransitionSub-cs1" %}

## Process

Processes is an array collection that defines the children values for BPMN subprocess.

## Loop

[Loop](#) is a task that is internally being looped. The loop property of task allows you to define the type of loop. The default value for loop is **none**.

You can define the loop property in subprocess BPMN shape as shown in the following code.

## APP.VUE

```

<template>
  <div id="app">
    <ejs-diagram id="diagram" :width='width' :height='height'
    :nodes='nodes' ></ejs-diagram>
  </div>
</template>
<script>
  import Vue from 'vue';

```

```

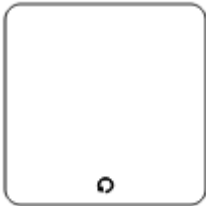
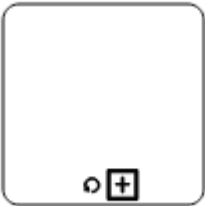



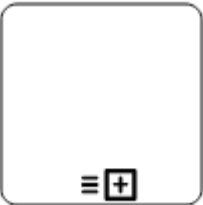
import {
DiagramPlugin, BpmnDiagrams, Diagram, BpmnGatewayModel, BpmnSubProcessModel }
from '@syncfusion/ej2-vue-diagrams';
Diagram.Inject(BpmnDiagrams);
Vue.use(DiagramPlugin);
let nodes = [{
  // Position of the node
  offsetX: 100,
  offsetY: 100,
  // Size of the node
  width: 100,
  height: 100,
  //Sets type as Bpmn and shape as Activity
  shape: {
    type: 'Bpmn',
    shape: 'Activity',
    //Sets activity as Task
    activity: {
      activity: 'Task',
      //Sets loop of the task as Standard
      task: {
        loop: 'Standard'
      }
    }
  },
},
{
  // Position of the node
  offsetX: 300,
  offsetY: 100,
  // Size of the node
  width: 100,
  height: 100,
  //Sets type as Bpmn and shape as activity
  shape: {
    type: 'Bpmn',
    shape: 'Activity',
    //Sets Activity as SubProcess
    activity: {
      activity: 'SubProcess',
      //Sets collapsed as true and loop as Standard
      subProcess: {
        collapsed: true,
        loop: 'Standard'
      }
    }
  }
  as BpmnSubProcessModel
},
],
}]
export default {
  name: 'app'
  data() {
    return {
      width: "100%",
      height: "350px",
      nodes: nodes,
    }
  }
}

```

```
    }  
  }  
</script>  
<style>  
  @import "../node_modules/@syncfusion/ej2-vue-  
diagrams/styles/material.css";  
</style>
```

{% previewsample "page.domainurl/code-snippet/diagram/bpmn-shapes/Loop-cs1" %}

The following table contains various types of BPMN loops.

Loops	Task	Subprocess
-----	-----	-----
<div>Standard</div>	<div></div>	<div></div>
<div>SequenceMultiInstance</div>	<div></div>	<div></div>
<div>ParallelMultiInstance</div>	<div></div>	<div></div>

[Compensation](#)  
[Compensation](#) is triggered, when operation is partially failed and enabled it with the compensation property of the task and the subprocess.

APP.VUE

```
<template>  
  <div id="app">  
    <ejs-diagram id="diagram" :width='width' :height='height'  
:nodes='nodes' ></ejs-diagram>  
  </div>  
</template>  
<script>  
  import Vue from 'vue';
```

```

import {
DiagramPlugin, BpmnDiagrams, Diagram, BpmnGatewayModel, BpmnSubProcessModel }
from '@syncfusion/ej2-vue-diagrams';
Diagram.Inject(BpmnDiagrams);
Vue.use(DiagramPlugin);
let nodes = [{
  // Position of the node
  offsetX: 100,
  offsetY: 100,
  // Size of the node
  width: 100,
  height: 100,
  //Sets type as Bpmn and shape as Activity
  shape: {
    type: 'Bpmn',
    shape: 'Activity',
    //Sets activity as Task
    activity: {
      activity: 'Task',
      //Sets compensation of the task as true
      task: {
        compensation: true
      }
    }
  },
},
{
  // Position of the node
  offsetX: 300,
  offsetY: 100,
  // Size of the node
  width: 100,
  height: 100,
  //Sets type as Bpmn and shape as Activity
  shape: {
    type: 'Bpmn',
    shape: 'Activity',
    //Sets activity as SubProcess
    activity: {
      activity: 'SubProcess',
      //Set the collapsed as true and compensation as true
      subProcess: {
        collapsed: true,
        compensation: true
      }
    }
  }
  as BpmnSubProcessModel
},
],
}]
export default {
  name: 'app'
  data() {
    return {
      width: "100%",
      height: "350px",
      nodes: nodes,
    }
  }
}

```

```

    }
  }
</script>
<style>
  @import "../node_modules/@syncfusion/ej2-vue-
diagrams/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/diagram/bpmn-shapes/Compensation-cs1" %}

## Call

A [call](#) activity is a global subprocess that is reused at various points of the business flow and set it with the call property of the task.

## APP.VUE

```

<template>
  <div id="app">
    <ejs-diagram id="diagram" :width='width' :height='height'
:nodes='nodes' ></ejs-diagram>
  </div>
</template>
<script>
  import Vue from 'vue';
  import { DiagramPlugin, BpmnDiagrams, Diagram, BpmnGatewayModel } from
'@syncfusion/ej2-vue-diagrams';
  Diagram.Inject(BpmnDiagrams);
  Vue.use(DiagramPlugin);
  let nodes = [{
    // Position of the node
    offsetX: 250,
    offsetY: 250,
    // Size of the node
    width: 100,
    height: 100,
    //Sets type as Bpmn and shape as Activity
    shape: {
      type: 'Bpmn',
      shape: 'Activity',
      //Sets the activity as task
      activity: {
        activity: 'Task',
        //Sets the call of the task as true
        task: {
          call: true
        }
      }
    },
  ],
  },
  ]
  export default {
    name: 'app'
    data() {
      return {
        width: "100%",
        height: "350px",
        nodes: nodes,

```



```

    }
  }
}
</script>
<style>
  @import "../node_modules/@syncfusion/ej2-vue-
diagrams/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/diagram/bpmn-shapes/Call-cs1" %}

### Ad-Hoc

An adhoc subprocess is a group of tasks that are executed in any order or skipped in order to fulfill the end condition and set it with the [adhoc](#) property of subprocess.

### APP.VUE

```

<template>
  <div id="app">
    <ejs-diagram id="diagram" :width='width' :height='height'
:nodes='nodes' ></ejs-diagram>
  </div>
</template>
<script>
  import Vue from 'vue';
  import {
DiagramPlugin, BpmnDiagrams, Diagram, BpmnGatewayModel, BpmnSubProcessModel }
from '@syncfusion/ej2-vue-diagrams';
  Diagram.Inject(BpmnDiagrams);
  Vue.use(DiagramPlugin);
  let nodes = [{
    // Position of the node
    offsetX: 250,
    offsetY: 250,
    // Size of the node
    width: 100,
    height: 100,
    //Sets type as Bpmn and shape as Activity
    shape: {
      type: 'Bpmn',
      shape: 'Activity',
      //Sets activity as SubProcess
      activity: {
        activity: 'SubProcess',
        //Sets collapsed as true and adhoc as true
        subprocess: {
          collapsed: true,
          adhoc: true
        }
      }
    },
    as BpmnSubProcessModel
  },
  ],
  export default {
    name: 'app'
    data() {

```

```

        return {
            width: "100%",
            height: "350px",
            nodes: nodes,
        }
    }
}
</script>
<style>
    @import "../node_modules/@syncfusion/ej2-vue-
    diagrams/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/diagram/bpmn-shapes/Adhoc-cs1" %}

### Boundary

Boundary represents the type of task that is being processed. The [boundary](#) property of subprocess allows you to define the type of boundary. By default, it is set as **default**.

### APP.VUE

```

<template>
    <div id="app">
        <ejs-diagram id="diagram" :width='width' :height='height'
        :nodes='nodes' ></ejs-diagram>
    </div>
</template>
<script>
    import Vue from 'vue';
    import {
    DiagramPlugin, BpmnDiagrams, Diagram, BpmnGatewayModel, BpmnSubProcessModel }
    from '@syncfusion/ej2-vue-diagrams';
    Diagram.Inject(BpmnDiagrams);
    Vue.use(DiagramPlugin);
    let nodes = [{
        // Position of the node
        offsetX: 250,
        offsetY: 250,
        // Size of the node
        width: 100,
        height: 100,
        //Sets type as Bpmn and shape as Activity
        shape: {
            type: 'Bpmn',
            shape: 'Activity',
            //Sets activity as SubProcess
            activity: {
                activity: 'SubProcess',
                //Sets collapsed as true and boundary as Call
                subprocess: {
                    collapsed: true,
                    boundary: 'Call'
                }
            }
        } as BpmnSubProcessModel
    },
    ],

```

```

  ]]
  export default {
    name: 'app'
    data() {
      return {
        width: "100%",
        height: "350px",
        nodes: nodes,
      }
    }
  }
</script>
<style>
  @import "../node_modules/@syncfusion/ej2-vue-
  diagrams/styles/material.css";
</style>

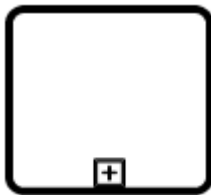
```

{% previewsample "page.domainurl/code-snippet/diagram/bpmn-shapes/Boundary-cs1" %}

The following table contains various types of BPMN boundaries.

| Boundary | Image |

| ----- | ----- |



| Call |



| Event |



| Default |

### Data

A data object represents information flowing through the process, such as data placed into the process, data resulting from the process, data that needs to be collected, or data that must be stored. To define a

[data object](#), set the shape as **DataObject** and the type property defines whether data is an input or an output. You can create multiple instances of data object with the collection property of data.

### APP.VUE

```
<template>
  <div id="app">
    <ejs-diagram id="diagram" :width='width' :height='height'
:nodes='nodes' ></ejs-diagram>
  </div>
</template>
<script>
  import Vue from 'vue';
  import {
DiagramPlugin, BpmnDiagrams, Diagram, BpmnGatewayModel, BpmnShapeModel } from
'@syncfusion/ej2-vue-diagrams';
  Diagram.Inject(BpmnDiagrams);
  Vue.use(DiagramPlugin);
  let nodes = [{
    // Position of the node
    offsetX: 250,
    offsetY: 250,
    // Size of the node
    width: 100,
    height: 100,
    //Sets type as Bpmn and shape as DataObject
    shape: {
      type: 'Bpmn',
      shape: 'DataObject',
      //Sets collection as true and type as Input
      dataObject: {
        collection: true,
        type: 'Input'
      }
    }
  ]
  as BpmnShapeModel,
}]
export default {
  name: 'app'
  data() {
    return {
      width: "100%",
      height: "350px",
      nodes: nodes,
    }
  }
}
</script>
<style>
  @import "../node_modules/@syncfusion/ej2-vue-
diagrams/styles/material.css";
</style>
```

{% previewsample "page.domainurl/code-snippet/diagram/bpmn-shapes/Data-cs1" %}

The following table contains various representation of BPMN data object.

| Boundary | Image |

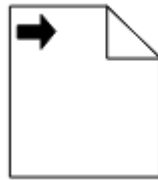
| ----- | ----- |



| Collection Data Object |



| Data Input |



| Data Output |

### Datasource

Datasource is used to store or access data associated with a business process. To create a datasource, set the shape as **datasource**. The following code example illustrates how to create a datasource.

### APP.VUE

```
<template>
  <div id="app">
    <ejs-diagram id="diagram" :width='width' :height='height'
:nodes='nodes' ></ejs-diagram>
  </div>
</template>
<script>
  import Vue from 'vue';
  import { DiagramPlugin,BpmnDiagrams,Diagram,BpmnGatewayModel } from
'@syncfusion/ej2-vue-diagrams';
  Diagram.Inject(BpmnDiagrams);
  Vue.use(DiagramPlugin);
  let nodes = [{
    // Position of the node
    offsetX: 250,
    offsetY: 250,
    // Size of the node
    width: 100,
    height: 100,
    //Sets type as Bpmn and shape as DataSource
    shape: {
      type: 'Bpmn',
      shape: 'DataSource',
    }
  }]
  export default {
    name: 'app'
```

```

    data() {
      return {
        width: "100%",
        height: "350px",
        nodes: nodes,
      }
    }
  }
</script>
<style>
  @import "../node_modules/@syncfusion/ej2-vue-
  diagrams/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/diagram/bpmn-shapes/Datasource-cs1" %}

### Artifact

Artifact is used to show additional information about a process in order to make it easier to understand. There are two types of artifacts in BPMN.

- Text annotation
- Group

### Text Annotation

- A BPMN object can be associated with a text annotation which does not affect the flow but gives details about objects within a flow. The annotation property of the node is used to connect an annotation element to the BPMN node.
- The annotation element can be displaced into a different position interactively by dragging the annotation to a particular position.
- The annotation element can be switched from a BPMN node to another BPMN node simply by dragging the source end of the annotation connector into the other BPMN node.
- The annotation angle property is used to set the angle between the BPMN shape and the annotation.
- The annotation direction property is used to set the direction of the text annotation.
- To set the size for text annotation, use width and height properties.
- The annotation length property is used to set the distance between the BPMN shape and the annotation.
- The annotation text property defines the additional information about the flow object in a BPMN process.

### APP.VUE

```

<template>
  <div id="app">
    <ejs-diagram id="diagram" :width='width' :height='height'
    :nodes='nodes' ></ejs-diagram>
  </div>
</template>
<script>
  import Vue from 'vue';

```

```

import {
DiagramPlugin, BpmnDiagrams, Diagram, BpmnGatewayModel, BpmnShapeModel } from
'@syncfusion/ej2-vue-diagrams';
Diagram.Inject(BpmnDiagrams);
Vue.use(DiagramPlugin);
let nodes = [{
// Position of the node
offsetX: 100,
offsetY: 100,
// Size of the node
width: 100,
height: 100,
//Sets type as Bpmn and shape as DataObject
shape: {
type: 'Bpmn',
shape: 'DataObject',
//Sets collection as true and type as Input
dataObject: {
collection: true,
type: 'Input'
},
//Sets the id, angle, length and text for the annotation
annotations: [{
id: 'left',
angle: 45,
length: 150,
text: 'Left',
}]
}
as BpmnShapeModel,
}]
export default {
name: 'app'
data() {
return {
width: "100%",
height: "350px",
nodes: nodes,
}
}
}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-vue-
diagrams/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/diagram/bpmn-shapes/Text-cs1" %}

### Group

A group is used to frame a part of the diagram, shows that elements included in it are logically belong together and don't have any other semantics other than organizing elements. To create a Group, the shape property of node should be set as "group". The following code example illustrates how to create a BPMN Group.

**APP.VUE**

```

<template>
  <div id="app">
    <ejs-diagram id="diagram" :width='width' :height='height'
:nodes='nodes' ></ejs-diagram>
  </div>
</template>
<script>
  import Vue from 'vue';
  import { DiagramPlugin,BpmnDiagrams,Diagram,BpmnGatewayModel } from
'@syncfusion/ej2-vue-diagrams';
  Diagram.Inject(BpmnDiagrams);
  Vue.use(DiagramPlugin);
  let nodes = [{
    // Position of the node
    offsetX: 250,
    offsetY: 250,
    // Size of the node
    width: 100,
    height: 100,
    //Sets type as Bpmn and shape as Group
    shape: {
      type: 'Bpmn',
      shape: 'Group',
    }
  }]
  export default {
    name: 'app'
    data() {
      return {
        width: "100%",
        height: "350px",
        nodes: nodes,
      }
    }
  }
</script>
<style>
  @import "../node_modules/@syncfusion/ej2-vue-
diagrams/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/diagram/bpmn-shapes/Group-cs1" %}

**BPMN Flows**

**BPMN Flows** are lines that connects BPMN flow objects.

**Association**

**BPMN Association** flow is used to link flow objects with its corresponding text or artifact. An association is represented as a dotted graphical line with opened arrow. The type of association are as follows.

- Directional
- BiDirectional
- Default



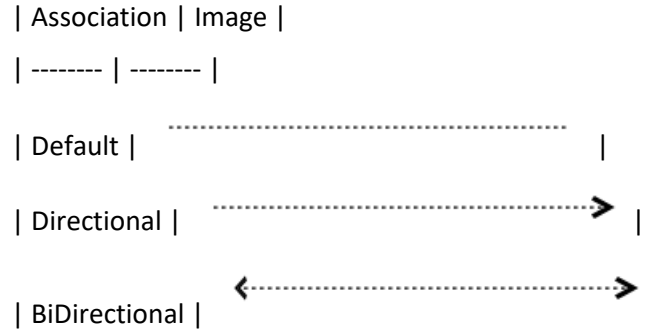
The `association` property allows you to define the type of association. The following code example illustrates how to create an association.

#### APP.VUE

```
<template>
  <div id="app">
    <ejs-diagram id="diagram" :width='width' :height='height'
      :connectors='connectors' ></ejs-diagram>
    </div>
  </template>
<script>
  import Vue from 'vue';
  import { DiagramPlugin, BpmnDiagrams, Diagram } from '@syncfusion/ej2-vue-
diagrams';
  Diagram.Inject(BpmnDiagrams);
  Vue.use(DiagramPlugin);
  let connectors = [{
    // Position of the node
    sourcePoint: {
      x: 100,
      y: 200
    },
    targetPoint: {
      x: 300,
      y: 200
    },
    //Sets type of the connector as Orthogonal
    type: 'Orthogonal',
    //Sets type as Bpmn, shflowape as Association and association as
    BiDirectional
    shape: {
      type: 'Bpmn',
      flow: 'Association',
      association: 'BiDirectional'
    },
  },
}]
export default {
  name: 'app'
  data() {
    return {
      width: "100%",
      height: "350px",
      connectors: connectors
    }
  }
}
</script>
<style>
  @import "../node_modules/@syncfusion/ej2-vue-
diagrams/styles/material.css";
</style>
```

{% previewsample "page.domainurl/code-snippet/diagram/bpmn-shapes/Association-cs1" %}

The following table demonstrates the visual representation of association flows.



Note : The default value for the property `association` is “default”.

### Sequence

A [Sequence](#) flow shows the order in which the activities are performed in a BPMN Process and is represented with a solid graphical line. The type of sequence are as follows.

- Normal
- Conditional
- Default

The sequence property allows you to define the type of sequence. The following code example illustrates how to create a sequence flow.

### APP.VUE

```
<template>
  <div id="app">
    <ejs-diagram id="diagram" :width='width' :height='height'
:connectors='connectors' ></ejs-diagram>
  </div>
</template>
<script>
  import Vue from 'vue';
  import { DiagramPlugin,BpmnDiagrams,Diagram } from '@syncfusion/ej2-vue-
diagrams';
  Diagram.Inject(BpmnDiagrams);
  Vue.use(DiagramPlugin);
  let connectors = [{
    // Position of the node
    sourcePoint: {
      x: 100,
      y: 200
    },
    targetPoint: {
      x: 300,
      y: 200
    },
    type: 'Orthogonal',
    //Sets type as Bpmn, flow as Sequence and sequence as Conditional
    shape: {
      type: 'Bpmn',
      flow: 'Sequence',
      sequence: 'Conditional'
    },
  },
```

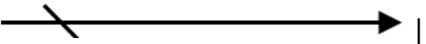
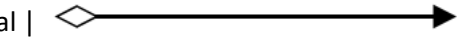
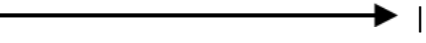
```

  ]]
  export default {
    name: 'app'
    data() {
      return {
        width: "100%",
        height: "350px",
        connectors: connectors
      }
    }
  }
</script>
<style>
  @import "../node_modules/@syncfusion/ej2-vue-
  diagrams/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/diagram/bpmn-shapes/Sequence-cs1" %}

The following table contains various representation of sequence flows.

Sequence	Image
Default	
Conditional	
Normal	

Note : The default value for the property `sequence` is “normal”.

### Message

A [message](#) flow shows the flow of messages between two Participants. A message flow is represented by dashed line. The type of message are as follows.

- InitiatingMessage
- NonInitiatingMessage
- Default

The message property allows you to define the type of message. The following code example illustrates how to define a message flow.

### APP.VUE

```

<template>
  <div id="app">
    <ejs-diagram id="diagram" :width='width' :height='height'
    :connectors='connectors' ></ejs-diagram>
  </div>
</template>
<script>
  import Vue from 'vue';
  import { DiagramPlugin,BpmnDiagrams,Diagram } from '@syncfusion/ej2-vue-
  diagrams';

```

```

Diagram.Inject(BpmnDiagrams);
Vue.use(DiagramPlugin);
let connectors = [{
  // Position of the node
  sourcePoint: {
    x: 100,
    y: 200
  },
  targetPoint: {
    x: 300,
    y: 200
  },
  type: 'Orthogonal',
  //Sets type as Bpmn, flow as Message and message as InitiatingMessage
  shape: {
    type: 'Bpmn',
    flow: 'Message',
    message: 'InitiatingMessage'
  },
},
]
export default {
  name: 'app'
  data() {
    return {
      width: "100%",
      height: "350px",
      connectors: connectors
    }
  }
}
</script>
<style>
  @import "../node_modules/@syncfusion/ej2-vue-
  diagrams/styles/material.css";
</style>


```


{% previewsample "page.domainurl/code-snippet/diagram/bpmn-shapes/Message-cs1" %}


The following table contains various representation of message flows.

| Message | Image |

| ----- | ----- |

| Default |  |

| InitiatingMessage |  |

| NonInitiatingMessage |  |

Note: The default value for the property `message` is “default”.

## UML diagram in Vue Diagram component

### UML Class Diagram

A class diagram visually depicts the static structure of an application and is extensively employed in modeling object-oriented systems. It holds a unique position in UML diagrams, as it directly aligns with object-oriented languages. The diagram also facilitates automatic generation of class diagram shapes based on business logic, streamlining the translation from conceptual models to practical implementation.

### Uml Class Diagram Shapes

The UML class diagram shapes are explained as follows.

#### Class

- A class defines a group of objects that share common specifications, features, constraints, and semantics. To create a class object, the classifier should be defined using the [class](#) notation. This notation serves as a foundational element in object-oriented programming, encapsulating the essential characteristics and behavior that objects belonging to the class will exhibit.
- Also, define the [name](#), [attributes](#), and [methods](#) of the class using the class property of node.
- The attribute's [name](#), [type](#), and [scope](#) properties allow you to define the name, data type, and visibility of the attribute.
- The method's [name](#), [parameters](#), [type](#), and [scope](#) properties allow you to define the name, parameter, return type, and visibility of the methods.
- The method parameters object properties allow you to define the name and type of the parameter.
- The following code example illustrates how to create a class.

### APP.VUE

```
<template>
  <div id="app">
    <ejs-diagram id="diagram" :width='width' :height='height'
:nodes='nodes' ></ejs-diagram>
  </div>
</template>
<script>
  import Vue from 'vue';
  import { DiagramPlugin,UmlClassifierShapeModel } from '@syncfusion/ej2-
vue-diagrams';
  Vue.use(DiagramPlugin);
  let nodes = [{
    id: "Patient",
    //Position of the node
    offsetX: 200,
    offsetY: 200,
    style: {
      fill: '#26A0DA',
    },
    shape: {
      type: "UmlClassifier",
      //Define class object
      classShape: {
        name: "Patient",
        //Define class attributes
```

```

        attributes: [{ name: "accepted", type: "Date" }],
        //Define class methods
        methods: [{ name: "getHistory", type: "getHistory" }]
    },
    classifier: "Class"
  } as UmlClassifierShapeModel
}]
export default {
  name: 'app'
  data() {
    return {
      width: "100%",
      height: "350px",
      nodes: nodes,
    }
  }
}
</script>
<style>
  @import "../node_modules/@syncfusion/ej2-vue-
  diagrams/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/diagram/umldiagramshapes/class-cs1" %}

### Interface

- An interface is a specific type of classifier that signifies a declaration of a cohesive set of public features and obligations. When creating an interface, it involves defining the classifier property using the [interface](#) notation. This essential concept in object-oriented programming outlines a contract for classes to adhere to, specifying the required methods and behaviors without delving into the implementation details.
- Also, define the [name](#), [attributes](#), and [methods](#) of the interface using the interface property of the node.
- The attribute's name, type, and scope properties allow you to define the name, data type, and visibility of the attribute.
- The method's name, parameter, type, and scope properties allow you to define the name, parameter, return type, and visibility of the methods.
- The method parameter object properties of name and type allows you to define the name and type of the parameter.
- The following code example illustrates how to create an interface.

### APP.VUE

```

<template>
  <div id="app">
    <ejs-diagram id="diagram" :width='width' :height='height'
    :nodes='nodes' ></ejs-diagram>
  </div>
</template>
<script>
  import Vue from 'vue';

```

```

import { DiagramPlugin,UmlClassifierShapeModel } from '@syncfusion/ej2-
vue-diagrams';
Vue.use(DiagramPlugin);
let nodes = [{
  id: "Patient",
  //Position of the node
  offsetX: 200,
  offsetY: 200,
  style: {
    fill: '#26A0DA',
  },
  shape: {
    type: "UmlClassifier",
    //Define interface object
    interfaceShape: {
      name: "Patient",
      //Define interface attributes
      attributes: [{ name: "owner", type: "String[*]" }],
      //Define interface methods
      methods: [
        {
          name: "deposit",
          parameters: [
            {
              name: "amount",
              type: "Dollars"
            }
          ]
        }
      ]
    },
    classifier: "Interface"
  } as UmlClassifierShapeModel
}]
export default {
  name: 'app'
  data() {
    return {
      width: "100%",
      height: "350px",
      nodes: nodes,
    }
  }
}
</script>
<style>
  @import "../node_modules/@syncfusion/ej2-vue-
diagrams/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/diagram/umldiagramshapes/interface-cs1" %}

### Enumeration

- To establish an enumeration, designate the classifier property of the node as [enumeration](#). Additionally, define the name and enumerate the members of the enumeration using the appropriate enumeration property of the node. This process encapsulates a set of distinct values within the enumeration, allowing for a clear representation of specific, named constants within a system.
- You can set a name for the enumeration members collection using the name property of members collection.
- The following code example illustrates how to create an enumeration.

### APP.VUE

```
<template>
  <div id="app">
    <ejs-diagram id="diagram" :width='width' :height='height'
:nodes='nodes' ></ejs-diagram>
  </div>
</template>
<script>
  import Vue from 'vue';
  import { DiagramPlugin,UmlClassifierShapeModel } from '@syncfusion/ej2-
vue-diagrams';
  Vue.use(DiagramPlugin);
  let nodes = [{
    id: "Patient",
    offsetX: 200,
    offsetY: 200,
    style: {
      fill: '#26A0DA',
    },
    shape: {
      type: "UmlClassifier",
      //Define enumeration object
      enumerationShape: {
        name: "AccountType",
        //set the members of enumeration
        members: [
          {
            name: "Checking Account",
          },
          {
            name: "Savings Account"
          },
          {
            name: "Credit Account"
          }
        ]
      },
      classifier: "Enumeration"
    } as UmlClassifierShapeModel
  ]
}
export default {
  name: 'app'
  data() {
```



```

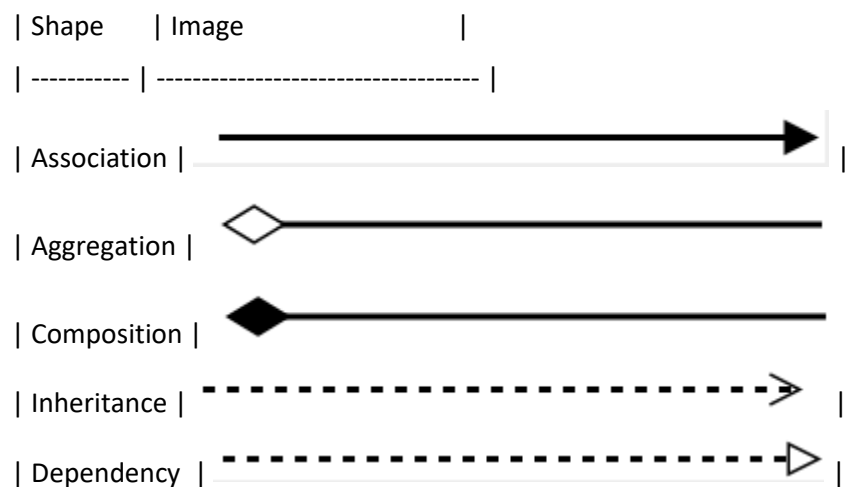
    return {
      width: "100%",
      height: "350px",
      nodes: nodes,
    }
  }
}
</script>
<style>
  @import "../node_modules/@syncfusion/ej2-vue-
  diagrams/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/diagram/umldiagramshapes/enumeration-cs1" %}

## UML Class Relationships

- A class may be involved in one or more relationships with other classes. A relationship can be one of the following types:



### Association

Association is basically a set of links that connects elements of an UML model. The type of association are as follows.

1. Directional
2. BiDirectional

The association property allows you to define the type of association. The default value of association is “Directional”. The following code example illustrates how to create an association.

### APP.VUE

```

<template>
  <div id="app">
    <ejs-diagram id="diagram" :width='width' :height='height'
    :connectors='connectors' ></ejs-diagram>
  </div>

```

```

</template>
<script>
  import Vue from 'vue';
  import { DiagramPlugin, UmlClassifierShapeModel } from '@syncfusion/ej2-vue-diagrams';
  Vue.use(DiagramPlugin);
  let connectors = [{
    id: "connector",
    //Define connector start and end points
    sourcePoint: { x: 100, y: 100 },
    targetPoint: { x: 300, y: 300 },
    type: "Straight",
    shape: {
      type: "UmlClassifier",
      relationship: "Association",
      //Define type of association
      association: "BiDirectional"
    }
  }]
  export default {
    name: 'app'
    data() {
      return {
        width: "100%",
        height: "350px",
        connectors: connectors,
      }
    }
  }
</script>
<style>
  @import "../../node_modules/@syncfusion/ej2-vue-diagrams/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/diagram/umldiagramshapes/association-cs1" %}

### Aggregation

Aggregation is a binary association between a property and one or more composite objects which group together a set of instances.

Aggregation is decorated with a hollow diamond. To create an aggregation shape, define the relationship as “aggregation”.

The following code example illustrates how to create an aggregation.

### APP.VUE

```

<template>
  <div id="app">
    <ejs-diagram id="diagram" :width='width' :height='height'
    :connectors='connectors' ></ejs-diagram>
  </div>
</template>
<script>
  import Vue from 'vue';

```

```

import { DiagramPlugin,UmlClassifierShapeModel } from '@syncfusion/ej2-
vue-diagrams';
Vue.use(DiagramPlugin);
let connectors = [{
  id: "connector",
  //Define connector start and end points
  sourcePoint: { x: 100, y: 100 },
  targetPoint: { x: 300, y: 300 },
  type: "Straight",
  shape: {
    type: "UmlClassifier",
    //Set an relationship for connector
    relationship: "Aggregation"
  }
}]
export default {
  name: 'app'
  data() {
    return {
      width: "100%",
      height: "350px",
      connectors: connectors,
    }
  }
}
</script>
<style>
  @import "../node_modules/@syncfusion/ej2-vue-
diagrams/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/diagram/umldiagramshapes/aggregation-cs1" %}

### Composition

Composition is a “strong” form of “aggregation”. Composition is decorated with a black diamond. To create a composition shape, define the relationship property of connector as “composition”.

The following code example illustrates how to create a composition.

### APP.VUE

```

<template>
  <div id="app">
    <ejs-diagram id="diagram" :width='width' :height='height'
    :connectors='connectors' ></ejs-diagram>
  </div>
</template>
<script>
  import Vue from 'vue';
  import { DiagramPlugin,UmlClassifierShapeModel } from '@syncfusion/ej2-
vue-diagrams';
  Vue.use(DiagramPlugin);
  let connectors = [{
    id: "connector",
    //Define connector start and end points
    sourcePoint: { x: 100, y: 100 },
    targetPoint: { x: 300, y: 300 },

```

```

    type: "Straight",
    shape: {
      type: "UmlClassifier",
      //Set an relationship for connector
      relationship: "Composition"
    }
  }]
  export default {
    name: 'app'
    data() {
      return {
        width: "100%",
        height: "350px",
        connectors: connectors,
      }
    }
  }
</script>
<style>
  @import "../node_modules/@syncfusion/ej2-vue-
  diagrams/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/diagram/umldiagramshapes/composition-cs1" %}

### Dependency

Dependency is a directed relationship, which is used to show that some UML elements needs or depends on other model elements for specifications. Dependency is shown as dashed line with opened arrow. To create a dependency, define the relationship property of connector as “dependency”.

The following code example illustrates how to create an dependency.

### APP.VUE

```

<template>
  <div id="app">
    <ejs-diagram id="diagram" :width='width' :height='height'
    :connectors='connectors' ></ejs-diagram>
  </div>
</template>
<script>
  import Vue from 'vue';
  import { DiagramPlugin,UmlClassifierShapeModel } from '@syncfusion/ej2-
  vue-diagrams';
  Vue.use(DiagramPlugin);
  let connectors = [{
    id: "connector",
    //Define connector start and end points
    sourcePoint: { x: 100, y: 100 },
    targetPoint: { x: 300, y: 300 },
    type: "Straight",
    shape: {
      type: "UmlClassifier",
      //Set an relationship for connector
      relationship: "Dependency"
    }
  }

```

```

]]
export default {
  name: 'app'
  data() {
    return {
      width: "100%",
      height: "350px",
      connectors: connectors,
    }
  }
}
</script>
<style>
  @import "../node_modules/@syncfusion/ej2-vue-
  diagrams/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/diagram/umldiagramshapes/dependency-cs1" %}

### Inheritance

Inheritance is also called as “generalization”. Inheritance is a binary taxonomic directed relationship between a more general classifier (super class) and a more specific classifier (subclass). Inheritance is shown as a line with hollow triangle. To create an inheritance, define the relationship as “inheritance”.

To create an inheritance, define the relationship as “inheritance”.

The following code example illustrates how to create an inheritance.

### APP.VUE

```

<template>
  <div id="app">
    <ejs-diagram id="diagram" :width='width' :height='height'
    :connectors='connectors' ></ejs-diagram>
  </div>
</template>
<script>
  import Vue from 'vue';
  import { DiagramPlugin,UmlClassifierShapeModel } from '@syncfusion/ej2-
  vue-diagrams';
  Vue.use(DiagramPlugin);
  let connectors = [{
    id: "connector",
    //Define connector start and end points
    sourcePoint: { x: 100, y: 100 },
    targetPoint: { x: 300, y: 300 },
    type: "Straight",
    shape: {
      type: "UmlClassifier",
      //Set an relationship for connector
      relationship: "Inheritance"
    }
  }]
  export default {
    name: 'app'
    data() {
      return {

```

```

        width: "100%",
        height: "350px",
        connectors: connectors,
      }
    }
  }
</script>
<style>
  @import "../node_modules/@syncfusion/ej2-vue-
  diagrams/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/diagram/umldiagramshapes/inheritance-cs1" %}

### Multiplicity

Multiplicity is a definition of an inclusive interval of non-negative integers to specify the allowable number of instances of described element. The type of multiplicity are as follows.

1. OneToOne
2. ManyToOne
3. OneToMany
4. ManyToMany
  - By default the multiplicity will be considered as “OneToOne”.
  - The multiplicity property in UML allows you to specify large number of elements or some collection of elements.
  - The shape multiplicity’s source property is used to set the source label to connector and the target property is used to set the target label to connector.
  - To set an optionality or cardinality for the connector source label, use optional property.
  - The [lowerBounds](#) and [upperBounds](#) could be natural constants or constant expressions evaluated to natural (non negative) number. Upper bound could be also specified as asterisk ‘\*’ which denotes unlimited number of elements. Upper bound should be greater than or equal to the lower bound.
  - The following code example illustrates how to customize the multiplicity.

### APP.VUE

```

<template>
  <div id="app">
    <ejs-diagram id="diagram" :width='width' :height='height'
    :connectors='connectors' ></ejs-diagram>
  </div>
</template>
<script>
  import Vue from 'vue';
  import { DiagramPlugin,UmlClassifierShapeModel } from '@syncfusion/ej2-
  vue-diagrams';
  Vue.use(DiagramPlugin);
  let connectors = [{
    id: "connector",
    //Define connector start and end points
    sourcePoint: { x: 100, y: 100 },
    targetPoint: { x: 300, y: 300 },
  }

```

```

    type: "Straight",
    shape: {
      type: "UmlClassifier",
      relationship: "Dependency",
      multiplicity: {
        //Set multiplicity type
        type: "OneToMany",
        //Set source label to connector
        source: {
          optional: true,
          lowerBounds: 89,
          upperBounds: 67
        },
        //Set target label to connector
        target: {
          optional: true,
          lowerBounds: 78,
          upperBounds: 90
        }
      }
    }
  }
}]
export default {
  name: 'app'
  data() {
    return {
      width: "100%",
      height: "350px",
      connectors: connectors,
    }
  }
}
</script>
<style>
  @import "../../node_modules/@syncfusion/ej2-vue-
  diagrams/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/diagram/umldiagramshapes/multiplicity-cs1" %}

### How to add UML child at runtime

In UML nodes, child elements such as member, method and attribute can be added either programmatically or interactively.

#### Adding UML child through code

The [addChildToUmlNode](#) method is employed for dynamically adding a child to the UML node during runtime, providing flexibility in modifying the diagram structure programmatically.

The following code illustrates how to add methods to UML nodes in diagram.

```
`ts
```

```
let node = diagram.selectedItems.nodes[0];
```

```
let methods = { name: 'getHistory', style: { color: "red", }, parameters: [{ name: 'Date', style: {} }], type: 'History' };
```

```
diagram.addChildToUmlNode(node, methods, 'Method');
```

```
,
```

### APP.VUE

```
<template>
  <div id="app">
    <button @click="addMethod">addMethod</button>
    <ejs-diagram id="diagram" ref="diagramObj" :width='width'
:height='height' :nodes='nodes' ></ejs-diagram>
  </div>
</template>
<script>
  import Vue from 'vue';
  import { DiagramPlugin,UmlClassifierShapeModel } from '@syncfusion/ej2-
vue-diagrams';
  Vue.use(DiagramPlugin);
  let nodes = [{
    id: 'node1',
    offsetX: 150,
    offsetY: 150,
    style: {
      fill: '#26A0DA',
    },
    shape: {
      type: 'UmlClassifier',
      classShape: {
        attributes: [
          { name: 'accepted', type: 'Date', },
        ],
        methods: [{ name: 'getHistory', style: {}, parameters: [{
name: 'Date', style: {} }], type: 'History' }],
        name: 'Patient'
      },
      classifier: 'Class'
    },
  ],
  export default {
    name: 'app',
    data() {
      return {
        width: "100%",
        height: "450px",
        nodes: nodes,
      }
    },
    methods: {
      addMethod: function () {
        let diagram = this.$refs.diagramObj.ej2Instances;
        let node = diagram.nodes[0];
        let methods = {
          name: 'getHistory',
          style: { color: 'red' },
          parameters: [{ name: 'Date', style: {} }],
          type: 'History',
        };
      }
    }
  }
};
```



```

        diagram.addChildToUmlNode(node, methods, 'Method');
    },
},
}
</script>
<style>
    @import "../node_modules/@syncfusion/ej2-vue-
    diagrams/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/diagram/umldiagramshapes/UmlMethod-cs1" %}

The following code illustrates how to add attributes to UML nodes in diagram.

`ts

```
let node = diagram.selectedItems.nodes[0];
```

```
let attributes = { name: 'accepted', type: 'Date', style: { color: "red", } };
```

```
diagram.addChildToUmlNode(node, attributes, "Attribute");
```

`

### APP.VUE

```

<template>
    <div id="app">
        <button @click="addAttribute">addAttribute</button>
        <ejs-diagram id="diagram" ref="diagramObj" :width='width'
        :height='height' :nodes='nodes' ></ejs-diagram>
    </div>
</template>
<script>
    import Vue from 'vue';
    import { DiagramPlugin,UmlClassifierShapeModel } from '@syncfusion/ej2-
    vue-diagrams';
    Vue.use(DiagramPlugin);
    let nodes = [{
        id: 'node1',
        offsetX: 150,
        offsetY: 150,
        style: {
            fill: '#26A0DA',
        },
        shape: {
            type: 'UmlClassifier',
            classShape: {
                attributes: [
                    { name: 'accepted', type: 'Date', },
                ],
                methods: [{ name: 'getHistory', style: {}, parameters: [{
name: 'Date', style: {} }], type: 'History' }],
                name: 'Patient'
            },
            classifier: 'Class'
        },
    },
    ]
}

```

```

export default {
  name: 'app',
  data() {
    return {
      width: "100%",
      height: "450px",
      nodes: nodes,
    }
  },
  methods: {
    addAttribute: function () {
      let diagram = this.$refs.diagramObj.ej2Instances;
      let node = diagram.nodes[0];
      let attributes = { name: 'accepted', type: 'Date', style: {
color: "red", } };
      diagram.addChildToUmlNode(node, attributes, "Attribute");
    },
  },
}
</script>
<style>
  @import "../node_modules/@syncfusion/ej2-vue-
diagrams/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/diagram/umldiagramshapes/UmlAttribute-cs1" %}

The following code illustrates how to add members to UML nodes in diagram.

`ts

```
let node = diagram.selectedItems.nodes[0];
```

```
let members = { name: 'Checking new', style: { color: 'red', }, isSeparator: true };
```

```
diagram.addChildToUmlNode(node, members, "Member");
```

,

### **APP.VUE**

```

<template>
  <div id="app">
    <button @click="addMember">addMember</button>
    <ejs-diagram id="diagram" ref="diagramObj" :width='width'
:height='height' :nodes='nodes' ></ejs-diagram>
  </div>
</template>
<script>
  import Vue from 'vue';
  import { DiagramPlugin,UmlClassifierShapeModel } from '@syncfusion/ej2-
vue-diagrams';
  Vue.use(DiagramPlugin);
  let nodes = [{
    id: 'node1',
    offsetX: 150,
    offsetY: 150,
    style: {

```

```

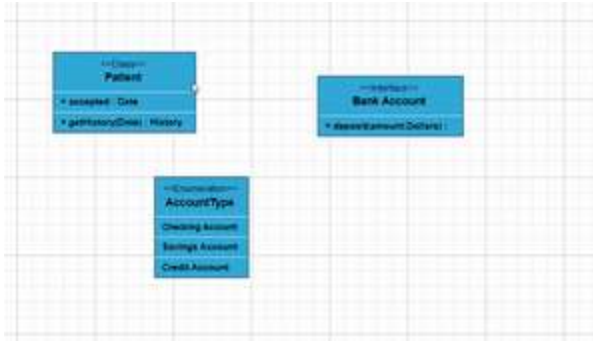
        fill: '#26A0DA',
      },
      shape: {
        type: 'UmlClassifier',
        enumerationShape: {
          name: 'AccountType',
          members: [
            {
              name: 'Checking Account',
            },
          ],
        },
        classifier: 'Enumeration'
      },
    ],
  },
  export default {
    name: 'app',
    data() {
      return {
        width: "100%",
        height: "450px",
        nodes: nodes,
      }
    },
    methods: {
      addMember: function () {
        let diagram = this.$refs.diagramObj.ej2Instances;
        let node = diagram.nodes[0];
        let members = { name: 'Checking new', style: { color: "red", },
isSeparator: true };
        diagram.addChildToUmlNode(node, members, "Member");
      },
    },
  }
</script>
<style>
  @import "../../node_modules/@syncfusion/ej2-vue-
diagrams/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/diagram/umldiagramshapes/UmlMember-cs1" %}

#### [Adding UML child through user interaction](#)

To include a child, select a node, move the mouse outside it, and position the pointer near the right side. A highlighter emerges between the two child elements. Click the highlighter to add a child type to the chosen UML node seamlessly. The following gif illustrates how to add Child through user interaction.



### Adding UML Nodes in Symbol palette

UML built-in shapes are efficiently rendered in a symbol palette. The `symbols` property is utilized to define UML symbols with the necessary classes and methods. By incorporating this feature, you can seamlessly augment the palette with a curated collection of predefined UML symbols, thereby enhancing the versatility of your UML diagramming application.

The following code example showcases the rendering of UML built-in shapes in a symbol palette.

#### APP.VUE

```

<template>
  <div style="width: 100%">
    <div id="palette-space" class="sb-mobile-palette">
      <ejs-symbolpalette
        id="symbolpalette"
        :palettes="palettes"
        :width="palettewidth"
        :height="paletteheight"
        :getNodeDefaults="palettegetNodeDefaults"
        :getSymbolInfo="getSymbolInfo"
        :symbolMargin="symbolMargin"
        :symbolHeight="symbolHeight"
        :symbolWidth="symbolWidth"
      ></ejs-symbolpalette>
    </div>
    <div id="diagram-space" class="sb-mobile-diagram">
      <ejs-diagram
        style="display: block"
        ref="diagramObject"
        id="diagram"
        :width="width"
        :height="height"
      ></ejs-diagram>
    </div>
  </div>
</template>
<script>
  import Vue from 'vue';
  import { DiagramPlugin,
    Diagram,
    SymbolPalette, SymbolPalettePlugin,
    SymbolInfo, } from '@syncfusion/ej2-vue-diagrams';
  Vue.use(DiagramPlugin);
  Vue.use(SymbolPalettePlugin);
  let umlShapes = [
  
```

```

{
  id: 'class',
  style: {
    fill: '#26A0DA',
  },
  borderColor: 'white',
  shape: {
    type: 'UmlClassifier',
    classShape: {
      attributes: [
        {
          name: 'accepted',
          type: 'Date',
          style: {
            color: 'red',
            fontFamily: 'Arial',
            textDecoration: 'Underline',
            italic: true,
          },
          isSeparator: true,
        },
      ],
    },
    methods: [
      {
        name: 'getHistory',
        style: {},
        parameters: [{ name: 'Date', style: {} }],
        type: 'History',
      },
    ],
    name: 'Patient',
  },
  classifier: 'Class',
},
{
  id: 'Interface',
  style: {
    fill: '#26A0DA',
  },
  borderColor: 'white',
  shape: {
    type: 'UmlClassifier',
    interfaceShape: {
      name: 'Bank Account',
    },
    classifier: 'Interface',
  },
},
{
  id: 'Enumeration',
  style: {
    fill: '#26A0DA',
  },
  borderColor: 'white',
  shape: {
    type: 'UmlClassifier',
  },
}

```

```

        enumerationShape: {
          name: 'AccountType',
          members: [
            {
              name: 'Checking Account',
              style: {},
            },
          ],
        },
        classifier: 'Enumeration',
      },
    ],
  ];

  export default {
    name: 'app',
    data() {
      return {
        width: '100%',
        height: '700px',
        palettes: [
          {
            id: 'uml',
            expanded: true,
            symbols: umlShapes,
            title: 'UML Shapes',
          },
        ],
        palettewidth: '100%',
        paletteheight: '700px',
        symbolHeight: 80,
        symbolWidth: 80,
        symbolMargin: {
          left: 12,
          right: 12,
          top: 12,
          bottom: 12,
        },
        palettegetNodeDefaults: (symbol) => {
          symbol.width = 100;
          symbol.height = 100;
        },
        symbolMargin: { left: 15, right: 15, top: 15, bottom: 15 },
        getSymbolInfo: (symbol) => {
          return { fit: true, description: { text: symbol.id } };
        },
      };
    },
  }
</script>
<style>
  @import
  "https://ej2.syncfusion.com/vue/documentation/../../node_modules/@syncfusion/ej2-vue-diagrams/styles/material.css";
</style>

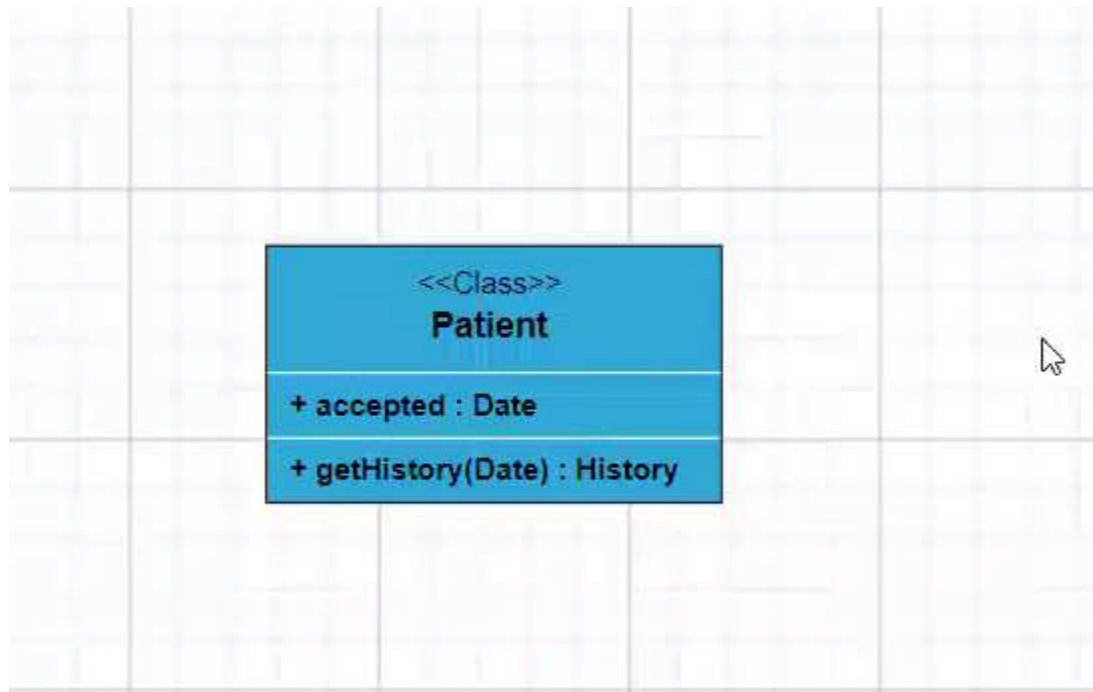
```

```
{% previewsample "page.domainurl/code-snippet/diagram/umldiagramshapes/Umlclass-cs1" %}
```

### Editing

You can edit the name, attributes, and methods of the class diagram shapes just double clicking, similar to editing a node annotation.

The following image illustrates how the text editor looks in an edit mode.



### UML Activity diagram

An Activity diagram functions as a visual flowchart, illustrating the progression from one activity to the next within a system. Each activity corresponds to a system operation, providing a clear depiction of the sequential flow in a dynamic process..

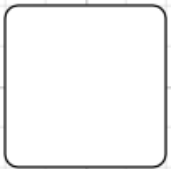
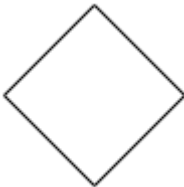
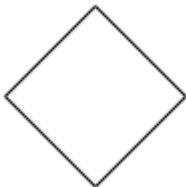




The purpose of an activity diagram can be described as follows.

1. Draw the activity flow of a system.
2. Describe the sequence from one activity to another.
3. Describe the parallel, branched, and concurrent flow of the system.

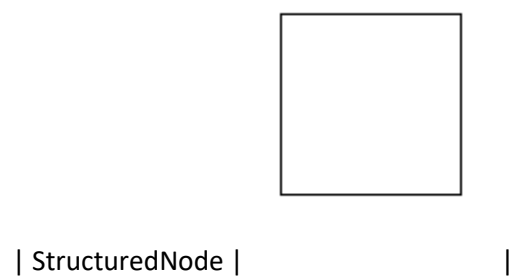
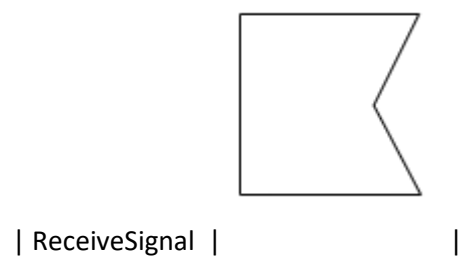
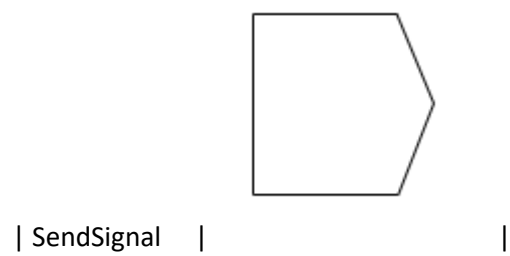
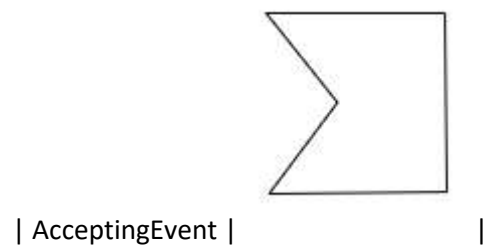
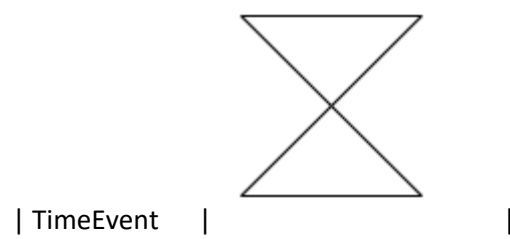
### UML Activity diagram Shapes

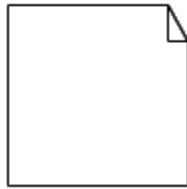
To create a UmlActivity, define type as "UmlActivity" and the list of built-in shapes as demonstrated as follows and it should be set in the "shape" property.

Shape	Image
-----	-----

Action		
Decision		
MergeNode		
InitialNode		
FinalNode		
ForkNode		
JoinNode		







| Note

The following code illustrates how to create a UmlActivity shapes.

#### APP.VUE

```

<template>
  <div id="app">
    <ejs-diagram id="diagram" :width='width' :height='height'
:nodes='nodes' ></ejs-diagram>
  </div>
</template>
<script>
  import Vue from 'vue';
  import { DiagramPlugin,UmlClassifierShapeModel } from '@syncfusion/ej2-
vue-diagrams';
  Vue.use(DiagramPlugin);
  let nodes = [{
    id: "UmlDiagram",
    //Set node size
    width: 100,
    height: 100,
    //position the node
    offsetX: 200,
    offsetY: 200,
    shape: {
      type: "UmlActivity",
      //Define UmlActivity shape
      shape: "Action"
    }
  }]
  export default {
    name: 'app'
    data() {
      return {
        width: "100%",
        height: "350px",
        nodes: nodes,
      }
    }
  }
</script>
<style>
  @import "../../node_modules/@syncfusion/ej2-vue-
diagrams/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/diagram/umldiagramshapes/UmlActivity-cs1" %}

### UmlActivity connector

To create an UmlActivity connector, define the type as "UmlActivity" and flow as either "Exception" or "Control" or "Object".

The following code illustrates how to create a UmlActivity connector.

#### APP.VUE

```
<template>
  <div id="app">
    <ejs-diagram id="diagram" :width='width' :height='height'
:connectors='connectors' ></ejs-diagram>
  </div>
</template>
<script>
  import Vue from 'vue';
  import { DiagramPlugin,UmlClassifierShapeModel } from '@syncfusion/ej2-
vue-diagrams';
  Vue.use(DiagramPlugin);
  let connectors = [{
    id: 'connector',
    type: 'Straight',
    //Define connector start and end points
    sourcePoint: { x: 100, y: 100 },
    targetPoint: { x: 200, y: 200 },
    shape: { type: 'UmlActivity', flow: 'Exception' }
  }]
  export default {
    name: 'app'
    data() {
      return {
        width: "100%",
        height: "350px",
        connectors: connectors,
      }
    }
  }
</script>
<style>
  @import "../../node_modules/@syncfusion/ej2-vue-
diagrams/styles/material.css";
</style>
```

{% previewsample "page.domainurl/code-snippet/diagram/umldiagramshapes/UmlActivityConnector-cs1" %}

### Connectors in Vue Diagram component

Connectors are objects used to create link between two points, nodes or ports to represent the relationships between them.

#### Create Connector

Connector can be created by defining the source and target point of the connector. The path to be drawn can be defined with a collection of segments. To explore the properties of a [connector](#), refer to [Connector Properties](#).

## Add connectors through connectors collection

- The [sourcePoint](#) and [targetPoint](#) properties of connector allow you to define the end points of a connector.

The following code example illustrates how to add a connector through connector collection.

**APP.VUE**

```
<template>
  <div id="app">
    <ejs-diagram id="diagram" :width='width' :height='height'
:connectors='connectors' ></ejs-diagram>
  </div>
</template>
<script>
  import Vue from 'vue';
  import { DiagramPlugin } from '@syncfusion/ej2-vue-diagrams';
  Vue.use(DiagramPlugin);
  let connectors = [{
    // Name of the connector
    id: "connector1",
    style: {
      strokeColor: '#6BA5D7',
      fill: '#6BA5D7',
      strokeWidth: 2
    },
    targetDecorator: {
      style: {
        fill: '#6BA5D7',
        strokeColor: '#6BA5D7'
      }
    },
    // Sets source and target points
    sourcePoint: {
      x: 100,
      y: 100
    },
    targetPoint: {
      x: 200,
      y: 200
    }
  }]
  export default {
    name: 'app'
    data() {
      return {
        width: "100%",
        height: "350px",
        connectors: connectors
      }
    }
  }
</script>
<style>
```

```
@import "../../../node_modules/@syncfusion/ej2-vue-  
diagrams/styles/material.css";  
</style>
```

{% previewsample "page.domainurl/code-snippet/diagram/connectors/Connectors-cs1" %}

Add connector at run time\*\*

- Connectors can be added at runtime by using public method, `diagram.add` and can be removed at runtime by using public method, `diagram.remove`.

The following code example illustrates how to add connector at runtime.

#### APP.VUE

```
<template>  
  <div id="app">  
    <ejs-diagram id="diagram" :width='width' :height='height'></ejs-  
diagram>  
  </div>  
</template>  
<script>  
  import Vue from 'vue';  
  import { DiagramPlugin, Diagram, ConnectorModel } from '@syncfusion/ej2-  
vue-diagrams';  
  Vue.use(DiagramPlugin);  
  let connector: ConnectorModel = {  
    // Name of the connector  
    id: "connector1",  
    style: {  
      strokeColor: '#6BA5D7',  
      fill: '#6BA5D7',  
      strokeWidth: 2  
    },  
    targetDecorator: {  
      style: {  
        fill: '#6BA5D7',  
        strokeColor: '#6BA5D7'  
      }  
    },  
    // Sets source and target points  
    sourcePoint: {  
      x: 100,  
      y: 100  
    },  
    targetPoint: {  
      x: 200,  
      y: 200  
    }  
  }  
  export default {  
    name: 'app'  
    data() {  
      return {  
        width: "100%",  

```

```

        height: "350px",
      }
    }
    mounted: function() {
      let diagramInstance: Diagram;
      let diagramObj: any = document.getElementById("diagram");
      diagramInstance = diagramObj.ej2_instances[0];
      // Adds to the diagram
      diagramInstance.add(connector)
    }
  }
</script>
<style>
  @import "../node_modules/@syncfusion/ej2-vue-
  diagrams/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/diagram/connectors/Connectorsatruntime-cs1" %}

### Connectors from palette

- Connectors can be predefined and added to the symbol palette. You can drop those connectors into the Diagram, when required.

For more information about adding connectors from symbol palette, refer to [\[Symbol Palette\]](#).

### Draw connectors

Connectors can be interactively drawn by clicking and dragging on the Diagram surface by using [\[drawingObject\]](#). For more information about drawing connectors, refer to [Draw Connectors](#).

### Update Connector at runtime

Various Connector properties such as [sourcePoint](#), [targetPoint](#), [style](#), [sourcePortID](#), [targetPortID](#) etc can be update at the run time .

- The following code example illustrates how to update a connector's source point, target point, styles properties at runtime.

### APP.VUE

```

<template>
  <div id="app">
    <ejs-diagram id="diagram" :width='width' :height='height'
    :connectors='connectors' ></ejs-diagram>
  </div>
</template>
<script>
  import Vue from 'vue';
  import { DiagramPlugin } from '@syncfusion/ej2-vue-diagrams';
  Vue.use(DiagramPlugin);
  let connectors = [{
    // Name of the connector
    id: "connector1",
    // Sets source and target points
    sourcePoint: {

```

```

        x: 100,
        y: 100
      },
      targetPoint: {
        x: 200,
        y: 200
      }
    }
  ]
}]
export default {
  name: 'app'
  data() {
    return {
      width: "100%",
      height: "350px",
      connectors: connectors
    }
  }
  mounted: function() {
    let diagramInstance: Diagram;
    let diagramObj: any = document.getElementById("diagram");
    diagramInstance = diagramObj.ej2_instances[0];
    diagramInstance.connectors[0].style.strokeColor = '#6BA5D7';
    diagramInstance.connectors[0].style.fill = '#6BA5D7';
    diagramInstance.connectors[0].style.strokeWidth = 2;
    diagramInstance.connectors[0].targetDecorator.style.fill =
'#6BA5D7';
    diagramInstance.connectors[0].targetDecorator.style.strokeColor =
'#6BA5D7';
    diagramInstance.connectors[0].sourcePoint.x = 150;
    diagramInstance.connectors[0].targetPoint.x = 150;
    diagramInstance.dataBind();
  }
}
</script>
<style>
  @import "../node_modules/@syncfusion/ej2-vue-
diagrams/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/diagram/connectors/Connectorsupdate-cs1" %}

### Connect nodes

- The [sourceID](#) and [targetID](#) properties allow to define the nodes to be connected.
- The [connectorSpacing](#) property allows you to define the distance between the source node and the connector. It is the minimum distance the connector will re-rout or the new segment will create.
- The following code example illustrates how to connect two nodes.

### APP.VUE

```

<template>
  <div id="app">

```

```

    <ejs-diagram id="diagram" :width='width' :height='height'
:nodes='nodes' :connectors='connectors' :getNodeDefaults='getNodeDefaults'
></ejs-diagram>
  </div>
</template>
<script>
  import Vue from 'vue';
  import { DiagramPlugin } from '@syncfusion/ej2-vue-diagrams';
  Vue.use(DiagramPlugin);
  let nodes = [{
    id: 'Start',
    width: 140,
    height: 50,
    offsetX: 300,
    offsetY: 100,
    annotations: [{
      id: 'label1',
      content: 'Start'
    }],
    shape: {
      type: 'Flow',
      shape: 'Terminator'
    }
  },
  {
    id: 'Init',
    width: 140,
    height: 50,
    offsetX: 300,
    offsetY: 300,
    shape: {
      type: 'Flow',
      shape: 'Process'
    },
    annotations: [{
      content: 'var i = 0;'
    }]
  }
  ];
  let connectors = [{
    id: "connector1",
    style: {
      strokeColor: '#6BA5D7',
      fill: '#6BA5D7',
      strokeWidth: 2
    },
    targetDecorator: {
      style: {
        fill: '#6BA5D7',
        strokeColor: '#6BA5D7'
      }
    },
    sourceID: "Start",
    targetID: "Init",
    connectorSpacing: 7,
    type: 'Orthogonal'
  }, ]

```



```

export default {
  name: 'app'
  data() {
    return {
      width: "100%",
      height: "350px",
      nodes: nodes,
      connectors: connectors,
      getNodeDefaults: (node) => {
        node.height = 100;
        node.width = 100;
        node.style.fill = '#6BA5D7';
        node.style.strokeColor = 'white';
        return node;
      },
    }
  }
}
</script>
<style>
  @import "../node_modules/@syncfusion/ej2-vue-
  diagrams/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/diagram/connectors/ConnectNode-cs1" %}

- When you remove NodeConstraints InConnect from Default, the node accepts only an outgoing connection to dock in it. Similarly, when you remove NodeConstraints OutConnect from Default, the node accepts only an incoming connection to dock in it.
- When you remove both InConnect and OutConnect NodeConstraints from Default, the node restricts connector to establish connection in it.
- The following code illustrates how to disable InConnect constraints.

`ts

//Initialize diagram

```

let diagram: Diagram = new Diagram({
  nodes:[
    {
      id: 'node', width: 100, height: 100, offsetX: 100, offsetY: 150,
      shape: { type: 'Basic', shape: 'Rectangle' },
      //Disable InConnect constraints
      constraints: NodeConstraints.Default & ~NodeConstraints.InConnect,
    }
  ]
});

```

```
diagram.appendTo('#diagram');
```

```
,
```

### Connections with ports

The [sourcePortID](#) and [targetPortID](#) properties allow to create connections between some specific points of source/target nodes.

The following code example illustrates how to create port to port connections.

#### APP.VUE

```
<template>
  <div id="app">
    <ejs-diagram id="diagram" :width='width' :height='height'
:nodes='nodes' :connectors='connectors' :getNodeDefaults='getNodeDefaults'
></ejs-diagram>
  </div>
</template>
<script>
  import Vue from 'vue';
  import { DiagramPlugin, PointPortModel, PortVisibility } from
'@syncfusion/ej2-vue-diagrams';
  Vue.use(DiagramPlugin);
  let port1: PointPortModel = {
    style: {
      strokeColor: '#366F8C',
      fill: '#366F8C'
    }
  }
  port1.shape = 'Circle';
  port1.id = 'nodeportnew'
  port1.visibility = PortVisibility.Visible
  port1.id = 'port';
  port1.offset = {
    x: 1,
    y: 1
  };
  let port2: PointPortModel = {
    style: {
      strokeColor: '#366F8C',
      fill: '#366F8C'
    }
  };
  port2.offset = {
    x: 1,
    y: 0.5
  };
  port2.id = 'port1';
  port2.visibility = PortVisibility.Visible
  port2.shape = 'Circle';
  let port3: PointPortModel = {
    style: {
      strokeColor: '#366F8C',
      fill: '#366F8C'
    }
  };
  port3.offset = {
```

```

        x: 0,
        y: 1
    };
    port3.id = 'newnodeport1';
    port3.visibility = PortVisibility.Visible
    port3.shape = 'Circle';
    let nodes = [{
        id: 'node',
        width: 100,
        height: 100,
        offsetX: 100,
        offsetY: 100,
        ports: [port1]
    },
    {
        id: 'node1',
        width: 100,
        height: 100,
        offsetX: 300,
        offsetY: 100,
        ports: [port2, port3]
    },
    ];
    let connectors = {
        id: "connector1",
        sourcePoint: {
            x: 100,
            y: 100
        },
        type: 'Orthogonal',
        targetPoint: {
            x: 200,
            y: 200
        },
        sourceID: 'node',
        targetID: 'node1',
        sourcePortID: 'port',
        targetPortID: 'port1'
    }
    export default {
        name: 'app'
        data() {
            return {
                width: "100%",
                height: "350px",
                nodes: nodes,
                connectors: [connectors],
                getNodeDefaults: (node) => {
                    node.height = 100;
                    node.width = 100;
                    node.style.fill = '#6BA5D7';
                    node.style.strokeColor = 'white';
                    return node;
                },
            }
        }
    }
}

```

```

</script>
<style>
  @import "../node_modules/@syncfusion/ej2-vue-
  diagrams/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/diagram/connectors/Connectorsportupdate-cs1" %}

similarly we can change the [sourcePortID] or [targetPortID] at the run time by the changing the port [sourcePortID](#) or [targetPortID](#)

The following code example illustrates how to create port to port connections.

#### APP.VUE

```

<template>
  <div id="app">
    <ejs-diagram id="diagram" :width='width' :height='height'
    :nodes='nodes' :connectors='connectors' :getNodeDefaults='getNodeDefaults'
    ></ejs-diagram>
  </div>
</template>
<script>
  import Vue from 'vue';
  import { DiagramPlugin, PointPortModel, PortVisibility } from
  '@syncfusion/ej2-vue-diagrams';
  Vue.use(DiagramPlugin);
  let port1: PointPortModel = {
    style: {
      strokeColor: '#366F8C',
      fill: '#366F8C'
    }
  }
  port1.shape = 'Circle';
  port1.id = 'nodeportnew'
  port1.visibility = PortVisibility.Visible
  port1.id = 'port';
  port1.offset = {
    x: 1,
    y: 1
  };
  let port2: PointPortModel = {
    style: {
      strokeColor: '#366F8C',
      fill: '#366F8C'
    }
  };
  port2.offset = {
    x: 1,
    y: 0.5
  };
  port2.id = 'port1';
  port2.visibility = PortVisibility.Visible
  port2.shape = 'Circle';
  let port3: PointPortModel = {
    style: {
      strokeColor: '#366F8C',

```

```

        fill: '#366F8C'
      }
    };
    port3.offset = {
      x: 0,
      y: 1
    };
    port3.id = 'newnodeport1';
    port3.visibility = PortVisibility.Visible;
    port3.shape = 'Circle';
    let nodes = [{
      id: 'node',
      width: 100,
      height: 100,
      offsetX: 100,
      offsetY: 100,
      ports: [port1]
    },
    {
      id: 'node1',
      width: 100,
      height: 100,
      offsetX: 300,
      offsetY: 100,
      ports: [port2, port3]
    }
  ];
  let connectors = {
    id: "connector1",
    sourcePoint: {
      x: 100,
      y: 100
    },
    type: 'Orthogonal',
    targetPoint: {
      x: 200,
      y: 200
    },
    sourceID: 'node',
    targetID: 'node1',
    sourcePortID: 'port',
    targetPortID: 'port1'
  }
  export default {
    name: 'app'
    data() {
      return {
        width: "100%",
        height: "350px",
        nodes: nodes,
        connectors: [connectors],
        getNodeDefaults: (node) => {
          node.height = 100;
          node.width = 100;
          node.style.fill = '#6BA5D7';
          node.style.strokeColor = 'white';
          return node;
        }
      }
    }
  }

```

```

    },
  },
  mounted: function() {
    let diagramInstance: Diagram;
    let diagramObj: any = document.getElementById("diagram");
    diagramInstance = diagramObj.ej2_instances[0];
    diagramInstance.connectors[0].targetPortID = 'newnodeport1'
  }
}
</script>
<style>
  @import "../node_modules/@syncfusion/ej2-vue-
  diagrams/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/diagram/connectors/ConnectorsSegments-cs1" %}

- When you set PortConstraints to InConnect, the port accepts only an incoming connection to dock in it. Similarly, when you set PortConstraints to OutConnect, the port accepts only an outgoing connection to dock in it.
- When you set PortConstraints to None, the port restricts connector to establish connection in it.

`ts

//Initialize diagram

```

let diagram: Diagram = new Diagram({
nodes:[
{
id: 'node', width: 100, height: 100, offsetX: 100, offsetY: 150,
shape: { type: 'Basic', shape: 'Rectangle' },
ports: [
//Enable portConstraints Inconnect
{ id: 'port', height: 10, width: 10, offset: { x: 1, y: 0.5 }, constraints: PortConstraints.InConnect },
]
}
]
});
diagram.appendTo('#diagram');

```

### Segments

The path of the connector is defined with a collection of segments. There are three types of segments.

### Straight

To create a straight line, you should specify the [type](#) of the segment as “straight” and add a straight segment to [segments](#) collection and need to specify [type](#) for the connector. The following code example illustrates how to create a default straight segment.

#### APP.VUE

```
<template>
  <div id="app">
    <ejs-diagram id="diagram" :width='width' :height='height'
:connectors='connectors' ></ejs-diagram>
  </div>
</template>
<script>
  import Vue from 'vue';
  import { DiagramPlugin } from '@syncfusion/ej2-vue-diagrams';
  Vue.use(DiagramPlugin);
  let connectors = [{
    id: "connector1",
    type: 'Straight',
    segments: [{
      // Defines the segment type of the connector
      type: 'Straight'
    }],
    style: {
      strokeColor: '#6BA5D7',
      fill: '#6BA5D7',
      strokeWidth: 2
    },
    targetDecorator: {
      style: {
        fill: '#6BA5D7',
        strokeColor: '#6BA5D7'
      }
    },
    sourcePoint: {
      x: 100,
      y: 100
    },
    targetPoint: {
      x: 200,
      y: 200
    }
  ]
}
export default {
  name: 'app'
  data() {
    return {
      width: "100%",
      height: "350px",
      connectors: connectors
    }
  }
}
</script>
<style>
```

```
@import "../../../node_modules/@syncfusion/ej2-vue-
diagrams/styles/material.css";
</style>
```

{% previewsample "page.domainurl/code-snippet/diagram/connectors/ConnectorsSegmentsPoints-cs1"  
%}

The [point](#) property of straight segment allows you to define the end point of it. The following code example illustrates how to define the end point of a straight segment.

#### APP.VUE

```
<template>
  <div id="app">
    <ejs-diagram id="diagram" :width='width' :height='height'
    :connectors='connectors' ></ejs-diagram>
  </div>
</template>
<script>
  import Vue from 'vue';
  import { DiagramPlugin } from '@syncfusion/ej2-vue-diagrams';
  Vue.use(DiagramPlugin);
  let connectors = [{
    id: "connector1",
    // Defines the segment type of the connector
    segments: [{
      type: 'Straight',
      // Defines the point of the segment
      point: {
        x: 100,
        y: 150
      }
    }],
    style: {
      strokeColor: '#6BA5D7',
      fill: '#6BA5D7',
      strokeWidth: 2
    },
    targetDecorator: {
      style: {
        fill: '#6BA5D7',
        strokeColor: '#6BA5D7'
      }
    },
    type: 'Straight',
    sourcePoint: {
      x: 100,
      y: 100
    },
    targetPoint: {
      x: 200,
      y: 200
    }
  }],
  export default {
    name: 'app'
```



```

    data() {
      return {
        width: "100%",
        height: "350px",
        connectors: connectors
      }
    }
  }
</script>
<style>
  @import "../node_modules/@syncfusion/ej2-vue-
  diagrams/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/diagram/connectors/ConnectorsOrthoSegments-cs1" %}

### Orthogonal

Orthogonal segments are used to create segments that are perpendicular to each other.

Set the segment [type](#) as “orthogonal” to create a default orthogonal segment and need to specify [type](#). The following code example illustrates how to create a default orthogonal segment.

- Multiple segments can be defined one after another. To create a connector with multiple segments, define and add the segments to [connector.segments](#) collection. The Following code example illustrates how to create a connector with multiple segments.
- The property [maxSegmentThumb](#) is used to limit the segment thumb in the connector.

### APP.VUE

```

<template>
  <div id="app">
    <ejs-diagram id="diagram" :width='width' :height='height'
    :connectors='connectors' ></ejs-diagram>
  </div>
</template>
<script>
  import Vue from 'vue';
  import { DiagramPlugin, ConnectorConstraints ,ConnectorEditing, Diagram}
  from '@syncfusion/ej2-vue-diagrams';
  Diagram.Inject(ConnectorEditing);
  Vue.use(DiagramPlugin);
  let connectors = [{
    id: "connector1",
    // Define the type of the segment
    type: 'Orthogonal',
    segments: [{
      type: 'Orthogonal'
    }],
    style: {
      strokeColor: '#6BA5D7',
      fill: '#6BA5D7',
      strokeWidth: 2
    },
    targetDecorator: {

```

```

        style: {
            fill: '#6BA5D7',
            strokeColor: '#6BA5D7'
        },
        sourcePoint: {
            x: 100,
            y: 100
        },
        targetPoint: {
            x: 200,
            y: 200
        },
        maxSegmentThumb: 3,
        constraints: ConnectorConstraints.Default |
ConnectorConstraints.DragSegmentThumb
    ]]
    export default {
        name: 'app',
        data() {
            return {
                width: "100%",
                height: "350px",
                connectors: connectors
            }
        }
    }
</script>
<style>
    @import "../../node_modules/@syncfusion/ej2-vue-
diagrams/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/diagram/connectors/ConnectorsOverlapping-cs1" %}

The [length](#) and [direction](#) properties allow to define the flow and length of segment. The following code example illustrates how to create customized orthogonal segments.

#### APP.VUE

```

<template>
    <div id="app">
        <ejs-diagram id="diagram" :width='width' :height='height'
        :connectors='connectors' ></ejs-diagram>
    </div>
</template>
<script>
    import Vue from 'vue';
    import { DiagramPlugin } from '@syncfusion/ej2-vue-diagrams';
    Vue.use(DiagramPlugin);
    let connectors = [{
        id: "connector1",
        type: 'Orthogonal',
        segments: [{
            type: 'Orthogonal',
            // Defines the direction for the segment lines

```

```

        direction: 'Right',
        // Defines the length for the segment lines
        length: 50
    }],
    style: {
        strokeColor: '#6BA5D7',
        fill: '#6BA5D7',
        strokeWidth: 2
    },
    targetDecorator: {
        style: {
            fill: '#6BA5D7',
            strokeColor: '#6BA5D7'
        }
    },
    sourcePoint: {
        x: 100,
        y: 100
    },
    targetPoint: {
        x: 200,
        y: 200
    }
},
{
    id: "connector2",
    type: 'Orthogonal',
    // Defines multile segemnts for the connectors
    segments: [{
        type: 'Orthogonal',
        direction: 'Bottom',
        length: 150
    },
    {
        type: 'Orthogonal',
        direction: 'Right',
        length: 150
    }
    ],
    style: {
        strokeColor: '#6BA5D7',
        fill: '#6BA5D7',
        strokeWidth: 2
    },
    targetDecorator: {
        style: {
            fill: '#6BA5D7',
            strokeColor: '#6BA5D7'
        }
    },
    sourcePoint: {
        x: 300,
        y: 100
    },
    targetPoint: {
        x: 400,
        y: 200
    }
}

```

```

    }
  }
]
export default {
  name: 'app'
  data() {
    return {
      width: "100%",
      height: "350px",
      connectors: connectors
    }
  }
}
</script>
<style>
  @import "../node_modules/@syncfusion/ej2-vue-
  diagrams/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/diagram/connectors/ConnectorsBezier-cs1" %}

Note: We need mention the segment type as same as what we mentioned in connector type. There should be no contradiction between connector type and segment type.

### Avoid overlapping

Orthogonal segments are automatically re-routed, in order to avoid overlapping with the source and target nodes. The following preview illustrate how orthogonal segments are re-routed.

### APP.VUE

```

<template>
  <div id="app">
    <ejs-diagram id="diagram" :width='width' :height='height'
    :nodes='nodes' :connectors='connectors' :getNodeDefaults='getNodeDefaults'
    ></ejs-diagram>
  </div>
</template>
<script>
  import Vue from 'vue';
  import { DiagramPlugin, PointPortModel, PortVisibility } from
  '@syncfusion/ej2-vue-diagrams';
  Vue.use(DiagramPlugin);
  let nodeport: PointPortModel = {
    style: {
      strokeColor: '#366F8C',
      fill: '#366F8C'
    }
  };
  nodeport.shape = 'Circle';
  nodeport.visibility = PortVisibility.Visible
  nodeport.id = 'port';
  nodeport.offset = {
    x: 0,
    y: 0.5
  };
  let port2: PointPortModel = {

```

```

        style: {
            strokeColor: '#366F8C',
            fill: '#366F8C'
        }
    }
    port2.offset = {
        x: 0,
        y: 0.5
    };
    port2.id = 'port1';
    port2.visibility = PortVisibility.Visible
    port2.shape = 'Circle';
let nodes = [{
    id: 'node',
    width: 100,
    height: 100,
    offsetX: 100,
    offsetY: 100,
    ports: [nodeport]
},
{
    id: 'node1',
    width: 100,
    height: 100,
    offsetX: 300,
    offsetY: 100,
    ports: [port2]
},
];
let connectors = {
    id: "connector1",
    style: {
        strokeColor: '#6BA5D7',
        fill: '#6BA5D7',
        strokeWidth: 2
    },
    targetDecorator: {
        style: {
            fill: '#6BA5D7',
            strokeColor: '#6BA5D7'
        }
    },
    type: 'Orthogonal',
    sourcePoint: {
        x: 100,
        y: 100
    },
    targetPoint: {
        x: 200,
        y: 200
    },
    sourceID: 'node',
    targetID: 'node1',
    sourcePortID: 'port',
    targetPortID: 'port1'
}
export default {

```

```

name: 'app'
data() {
  return {
    width: "100%",
    height: "350px",
    nodes: nodes,
    connectors: [connectors],
    getNodeDefaults: (node) => {
      node.height = 100;
      node.width = 100;
      node.style.fill = '#6BA5D7';
      node.style.strokeColor = 'white';
      return node;
    },
  }
}
</script>
<style>
  @import "../node_modules/@syncfusion/ej2-vue-
  diagrams/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/diagram/connectors/ConnectorsBezierPoints-cs1" %}

### How to customize Orthogonal Segment Thumb Shape

The orthogonal connector has a number of segments in between the source and the target point. The segments are rendered with the default shape rhombus. Now, the option has been provided to change the segment thumb shape using the [segmentThumbShape](#) property. The predefined shapes provided are as follows:

- Rhombus
- Square
- Rectangle
- Ellipse
- Arrow
- Diamond
- OpenArrow
- Circle
- Fletch
- OpenFetch
- IndentedArrow
- OutdentedArrow
- DoubleArrow

You can customize the style of the thumb shape by overriding the class e-orthogonal-thumb.

### APP.VUE

```

<template>
  <div id="app">
    <ejs-diagram id="diagram" :width='width' :height='height'
      :connectors='connectors'

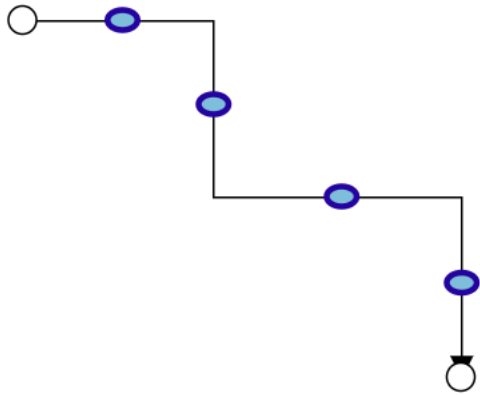
```

```

        :getConnectorDefaults='getConnectorDefaults'
        :segmentThumbShape="segmentThumbShape"
    >
    </ejs-diagram>
</div>
</template>
<script>
    import Vue from 'vue';
    import { DiagramPlugin, ConnectorConstraints, ConnectorEditing, Diagram }
    from '@syncfusion/ej2-vue-diagrams';
    Diagram.Inject(ConnectorEditing);
    Vue.use(DiagramPlugin);
    let connectors = [{
        id : 'connector',
        // Define the type of the segment
        type : 'Orthogonal',
        sourcePoint : { x: 250, y: 250 },
        targetPoint : { x: 350, y: 350 },
        segments : [
            {
                type: 'Orthogonal',
                // Defines the direction for the segment lines
                direction: "Right",
                // Defines the length for the segment lines
                length: 70
            },
            {
                type: 'Orthogonal',
                direction: "Bottom",
                length: 20
            }
        ]
    }
    ]
    export default {
        name: 'app',
        data() {
            return {
                width: "900px",
                height: "500px",
                connectors: connectors,
                segmentThumbShape: 'Square',
                getConnectorDefaults: (connector) => {
                    connector.constraints = ConnectorConstraints.Default |
ConnectorConstraints.DragSegmentThumb;
                }
            }
        }
    }
</script>
<style>
    @import "../../node_modules/@syncfusion/ej2-vue-
diagrams/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/diagram/connectors/ConnectorsThumbshape-cs1" %}



Use the following CSS to customize the segment thumb shape.

```
`scss
.e-orthogonal-thumb {
fill: rgb(126, 190, 219);
stroke: #24039e;
stroke-width: 3px;
}
`
```

### Bezier

Bezier segments are used to create curve segments and the curves are configurable either with the control points or with vectors.

To create a bezier segment, the [segment.type](#) is set as `bezier` and need to specify [type](#) for the connector. The following code example illustrates how to create a default Bezier segment.

### APP.VUE

```
<template>
  <div id="app">
    <ejs-diagram id="diagram" :width='width' :height='height'
:connectors='connectors' ></ejs-diagram>
  </div>
</template>
<script>
  import Vue from 'vue';
  import { DiagramPlugin } from '@syncfusion/ej2-vue-diagrams';
  Vue.use(DiagramPlugin);
```



```

let connectors = [{
  id: 'connector1',
  style: {
    strokeColor: '#6BA5D7',
    fill: '#6BA5D7',
    strokeWidth: 2
  },
  targetDecorator: {
    style: {
      fill: '#6BA5D7',
      strokeColor: '#6BA5D7'
    }
  },
  type: 'Bezier',
  segments: [{
    // Defines the type of the segment
    type: 'Bezier',
  }],
  sourcePoint: {
    x: 50,
    y: 100
  },
  targetPoint: {
    x: 150,
    y: 200
  },
}, ]
export default {
  name: 'app'
  data() {
    return {
      width: "100%",
      height: "350px",
      connectors: connectors
    }
  }
}
</script>
<style>
  @import "../node_modules/@syncfusion/ej2-vue-
  diagrams/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/diagram/connectors/ConnectorsBezier-cs2" %}

The [point1](#) and [point2](#) properties of bezier segment enable you to set the control points. The following code example illustrates how to configure the Bezier segments with control points.

#### APP.VUE

```

<template>
  <div id="app">
    <ejs-diagram id="diagram" :width='width' :height='height'
    :connectors='connectors' ></ejs-diagram>
  </div>
</template>

```

```

<script>
  import Vue from 'vue';
  import { DiagramPlugin } from '@syncfusion/ej2-vue-diagrams';
  Vue.use(DiagramPlugin);
  let connectors = [{
    id: 'connector3',
    type: 'Bezier',
    style: {
      strokeColor: '#6BA5D7',
      fill: '#6BA5D7',
      strokeWidth: 2
    },
    targetDecorator: {
      style: {
        fill: '#6BA5D7',
        strokeColor: '#6BA5D7'
      }
    },
    segments: [{
      type: 'Bezier',
      // First control point: an absolute position from the page origin
      point1: {
        x: 100,
        y: 100
      },
      // Second control point: an absolute position from the page origin
      point2: {
        x: 200,
        y: 200
      }
    }
  ]],
  sourcePoint: {
    x: 100,
    y: 200
  },
  targetPoint: {
    x: 200,
    y: 100
  },
}, ]
export default {
  name: 'app'
  data() {
    return {
      width: "100%",
      height: "350px",
      connectors: connectors
    }
  }
}
</script>
<style>
  @import "../../node_modules/@syncfusion/ej2-vue-diagrams/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/diagram/connectors/ConnectorsBezierPoints-cs2" %}

The [vector1](#) and [vector2](#) properties of bezier segment enable you to define the vectors. The following code illustrates how to configure a bezier curve with vectors.

#### APP.VUE

```
<template>
  <div id="app">
    <ejs-diagram id="diagram" :width='width' :height='height'
:connectors='connectors' ></ejs-diagram>
  </div>
</template>
<script>
  import Vue from 'vue';
  import { DiagramPlugin } from '@syncfusion/ej2-vue-diagrams';
  Vue.use(DiagramPlugin);
  let connectors = [{
    id: 'connector2',
    style: {
      strokeColor: '#6BA5D7',
      fill: '#6BA5D7',
      strokeWidth: 2
    },
    targetDecorator: {
      style: {
        fill: '#6BA5D7',
        strokeColor: '#6BA5D7'
      }
    },
    // Defines the type of the segment
    type: 'Bezier',
    segments: [{
      type: 'Bezier',
      // Length and angle between the source point and the first control
point
      vector1: {
        distance: 100,
        angle: 90
      },
      // Length and angle between the target point and the second control
point
      vector2: {
        distance: 45,
        angle: 270
      }
    },
  ]},
  sourcePoint: {
    x: 100,
    y: 100
  },
  targetPoint: {
    x: 200,
    y: 200
  },
}, ]
export default {
```

```

    name: 'app'
    data() {
      return {
        width: "100%",
        height: "350px",
        connectors: connectors
      }
    }
  }
</script>
<style>
  @import "../node_modules/@syncfusion/ej2-vue-
  diagrams/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/diagram/connectors/ConnectorsBezierVector-cs1" %}

### Avoid overlapping with bezier

By default, when there are no segments defined for a bezier connector, the bezier segments will be created automatically and routed in such a way that avoids overlapping with the source and target nodes.

,

<template>

<div id="app">

<ejs-diagram id="diagram" :width='width' :height='height' :nodes='nodes' :connectors='connectors'  
:getNodeDefaults='getNodeDefaults'

:getConnectorDefaults='getConnectorDefaults' ></ejs-diagram>

</div>

</template>

<script>

import Vue from 'vue';

import { DiagramPlugin,Diagram,NodeModel,ConnectorModel,ConnectorEditing,ConnectorConstraints }  
from '@syncfusion/ej2-vue-diagrams';

Diagram.Inject(ConnectorEditing);

Vue.use(DiagramPlugin);

let nodes: NodeModel[] = [{

id: 'Start',

offsetX: 250,

offsetY: 150,

annotations: [{ content: 'Start' }]

},

```
{
  id: 'End',
  offsetX: 450,
  offsetY: 200,
  annotations: [{ content: 'End' }]
  });
let connectors: ConnectorModel[] = [{
  id: "connector1",
  style: {
    strokeColor: '#6BA5D7',
    fill: '#6BA5D7',
    strokeWidth: 2
  },
  targetDecorator: { shape: 'None' },
  // ID of the source and target nodes
  sourceID: "Start",
  targetID: "End",
  type: 'Bezier'
  });
export default {
  name: 'app'
  data() {
    return {
      width: "100%",
      height: "600px",
      nodes:nodes,
      connectors:connectors,
      getNodeDefaults: (node)=>{
        node.height = 100;
        node.width = 100;
        node.shape = { type: 'Basic', shape: 'Rectangle' }
        node.style.fill = '#6BA5D7';
        node.style.strokeColor = 'white';
```

```

return node;
},
getConnectorDefaults: (connector)=> {
  connector.constraints = ConnectorConstraints.Default | ConnectorConstraints.DragSegmentThumb;
}
}
}
}
}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-vue-diagrams/styles/material.css";
</style>
`

```

Also, the intermediate point of two adjacent bezier segments can be edited interactively based on the `bezierSettings.segmentEditOrientation` property of the connector class.

#### *How to interact with the bezier segments efficiently*

While interacting with multiple bezier segments, maintain their control points at the same distance and angle by using the `bezierSettings.smoothness` property of the connector class.

| BezierSmoothness value | Description   |
|------------------------|---|
| ----- -----            |   |
| SymmetricDistance      | Both control points of adjacent segments will be at the same distance when any one of them is editing.                |
| SymmetricAngle         | Both control points of adjacent segments will be at the same angle when any one of them is editing.                   |
| Default                | Both control points of adjacent segments will be at the same angle and same distance when any one of them is editing. |
| None                   | Segment's control points are interacted independently from each other.  |

```

<template>
<div id="app">
<ejs-diagram id="diagram" :width='width' :height='height' :nodes='nodes' :connectors='connectors'
:getNodeDefaults='getNodeDefaults'
:getConnectorDefaults='getConnectorDefaults' ></ejs-diagram>
</div>
</template>

```

```
<script>
import Vue from 'vue';

import { DiagramPlugin,Diagram,NodeModel,ConnectorModel,ConnectorEditing,ConnectorConstraints }
from '@syncfusion/ej2-vue-diagrams';

Diagram.Inject(ConnectorEditing);

Vue.use(DiagramPlugin);

let nodes: NodeModel[] = [{
  id: 'Start',
  offsetX: 250,
  offsetY: 150,
  annotations: [{ content: 'Start' }],
  ports: [{
    id: 'StartPort',
    visibility: PortVisibility.Visible,
    shape: 'Circle',
    offset: { x: 1, y: 0.5 },
    style: { strokeColor: '#366F8C', fill: '#366F8C' }
  ]
},
{
  id: 'End',
  offsetX: 450,
  offsetY: 200,
  annotations: [{ content: 'End' }],
  ports: [{
    id: 'EndPort',
    visibility: PortVisibility.Visible,
    shape: 'Circle',
    offset: { x: 0, y: 0.5 },
    style: { strokeColor: '#366F8C', fill: '#366F8C' }
  ]
}
];

let connectors: ConnectorModel[] = [{
```

```
id: "connector1",
style: {
  strokeColor: '#6BA5D7',
  fill: '#6BA5D7',
  strokeWidth: 2
},
targetDecorator: { shape: 'None' },
// ID of the source and target nodes
sourceID: "Start",
targetID: "End",
type: 'Bezier',
// Configuring settings for bezier interactions
bezierSettings = { smoothness: BezierSmoothness.SymmetricAngle }
});
export default {
  name: 'app'
  data() {
    return {
      width: "100%",
      height: "600px",
      nodes:nodes,
      connectors:connectors,
      getNodeDefaults: (node)=>{
        node.height = 100;
        node.width = 100;
        node.shape = { type: 'Basic', shape: 'Rectangle' }
        node.style.fill = '#6BA5D7';
        node.style.strokeColor = 'white';
        return node;
      },
      getConnectorDefaults: (connector)=> {
        connector.constraints = ConnectorConstraints.Default | ConnectorConstraints.DragSegmentThumb;
      }
    }
  }
}
```



```

}
}
}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-vue-diagrams/styles/material.css";
</style>
`

```

Also, the visibility of control points can be controlled using the `bezierSettings.controlPointsVisibility` property of the connector class.

### APP.VUE

```

<template>
  <div id="app">
    <ejs-diagram id="diagram" :width='width' :height='height'
      :nodes='nodes' :connectors='connectors' :getNodeDefaults='getNodeDefaults'
      :getConnectorDefaults='getConnectorDefaults' ></ejs-diagram>
  </div>
</template>
<script>
  import Vue from 'vue';
  import {
    DiagramPlugin, Diagram, NodeModel, ConnectorModel, ConnectorEditing, ConnectorCon
    straints, PortVisibility, ControlPointsVisibility } from '@syncfusion/ej2-
    vue-diagrams';
  Diagram.Inject(ConnectorEditing);
  Vue.use(DiagramPlugin);
  let nodes = [{
    id: 'Start',
    offsetX: 250,
    offsetY: 150,
    annotations: [{ content: 'Start' }],
    ports: [{
      id: 'StartPort',
      visibility: PortVisibility.Visible,
      shape: 'Circle',
      offset: { x: 1, y: 0.5 },
      style: { strokeColor: '#366F8C', fill: '#366F8C' }
    }]
  },
  {
    id: 'End',
    offsetX: 450,
    offsetY: 200,
    annotations: [{ content: 'End' }],
    ports: [{
      id: 'EndPort',
      visibility: PortVisibility.Visible,
      shape: 'Circle',
      offset: { x: 0, y: 0.5 },

```

```

        style: { strokeColor: '#366F8C', fill: '#366F8C' }
      }}
    ]];
    let connectors = [{
      id: "connector1",
      style: {
        strokeColor: '#6BA5D7',
        fill: '#6BA5D7',
        strokeWidth: 2
      },
      targetDecorator: { shape: 'None' },
      // ID of the source and target nodes
      sourceID: "Start",
      targetID: "End",
      type: 'Bezier',
      // Configuring settings for bezier interactions
      bezierSettings : { controlPointsVisibility:
ControlPointsVisibility.Source | ControlPointsVisibility.Target }
    }];
    export default {
      name: 'app',
      data() {
        return {
          width: "100%",
          height: "600px",
          nodes:nodes,
          connectors:connectors,
          getNodeDefaults: (node)=>{
            node.height = 100;
            node.width = 100;
            node.shape = { type: 'Basic', shape: 'Rectangle' }
            node.style.fill = '#6BA5D7';
            node.style.strokeColor = 'white';
            return node;
          },
          getConnectorDefaults: (connector)=> {
            connector.constraints = ConnectorConstraints.Default |
ConnectorConstraints.DragSegmentThumb;
          }
        }
      }
    }
  }
</script>
<style>
  @import "../node_modules/@syncfusion/ej2-vue-
diagrams/styles/material.css";
</style>

```

```
{% previewsample "page.domainurl/code-
snippet/diagram/connectors/ConnectorBezierAvoidOverlapping-cs1" %}
```

## Decorator

- Start and end points of a connector can be decorated with some customizable shapes like arrows, circles, diamond or path. You can decorate the connection end points with the [sourceDecorator](#) and [targetDecorator](#) properties of connector.
- The [shape](#) property of [sourceDecorator](#) allows to define the shape of the decorators. Similarly, the [shape](#) property of [targetDecorator](#) allows to define the shape of the decorators.
- To create custom shape for source decorator, use [pathData](#) property. similarly, to create custom shape for target decorator, use [pathData](#) property.
- The following code example illustrates how to create decorators of various shapes.

**APP.VUE**

```

<template>
  <div id="app">
    <ejs-diagram id="diagram" :width='width' :height='height'
:connectors='connectors' ></ejs-diagram>
  </div>
</template>
<script>
  import Vue from 'vue';
  import { DiagramPlugin } from '@syncfusion/ej2-vue-diagrams';
  Vue.use(DiagramPlugin);
  let connectors = [{
    id: "connector1",
    type: 'Straight',
    // Decorator shape- circle
    sourceDecorator: {
      shape: 'Circle',
      // Defines the style for the sourceDecorator
      style: {
        // Defines the strokeWidth for the sourceDecorator
        strokeWidth: 3,
        // Defines the strokeColor for the sourceDecorator
        strokeColor: 'red'
      },
    },
    // Decorator shape - Diamond
    targetDecorator: {
      // Defines the custom shape for the connector's target decorator
      shape: 'Custom',
      //Defines the path for the connector's target decorator
      pathData: 'M80.5,12.5 C80.5,19.127417 62.59139,24.5 40.5,24.5
C18.40861,24.5 0.5,19.127417 0.5,12.5' +
        'C0.5,5.872583 18.40861,0.5 40.5,0.5 C62.59139,0.5
80.5,5.872583 80.5,12.5 z'
      //defines the style for the target decorator
      style: {
        // Defines the strokeWidth for the targetDecorator
        strokeWidth: 3,
        // Defines the strokeColor for the sourceDecorator
        strokeColor: 'green',
        // Defines the opacity for the sourceDecorator
        opacity: .8
      }
    }
  ]

```

```

    },
    },
    sourcePoint: {
      x: 100,
      y: 100
    },
    targetPoint: {
      x: 200,
      y: 200
    }
  },
  {
    id: "connectors2",
    type: 'Straight',
    // Decorator shape - IndentedArrow
    sourceDecorator: {
      shape: 'IndentedArrow',
      style: {
        strokeWidth: 3,
        strokeColor: 'blue'
      },
    },
    // Decorator shape - OutdentedArrow
    targetDecorator: {
      shape: 'OutdentedArrow',
      style: {
        strokeWidth: 3,
        strokeColor: 'yellow'
      },
    },
    sourcePoint: {
      x: 400,
      y: 100
    },
    targetPoint: {
      x: 300,
      y: 200
    }
  }
]
export default {
  name: 'app'
  data() {
    return {
      width: "100%",
      height: "350px",
      connectors: connectors
    }
  }
}
</script>
<style>
  @import "../node_modules/@syncfusion/ej2-vue-
  diagrams/styles/material.css";
</style>

```

```
{% previewsample "page.domainurl/code-snippet/diagram/connectors/ConnectorsDecorator-cs1" %}
```

### Padding

Padding is used to leave the space between the Connector's end point and the object to where it is connected.

- The [sourcePadding](#) property of connector defines space between the source point and the source node of the connector.
- The [targetPadding](#) property of connector defines space between the end point and the target node of the connector.
- The following code example illustrates how to leave space between the connection end points and source and target nodes.

### APP.VUE

```
<template>
  <div id="app">
    <ejs-diagram id="diagram" :width='width' :height='height'
      :nodes='nodes' :connectors='connectors' :getNodeDefaults='getNodeDefaults'
    ></ejs-diagram>
  </div>
</template>
<script>
  import Vue from 'vue';
  import { DiagramPlugin } from '@syncfusion/ej2-vue-diagrams';
  Vue.use(DiagramPlugin);
  let nodes = [{
    id: 'node',
    width: 100,
    height: 100,
    offsetX: 100,
    offsetY: 100,
  },
  {
    id: 'node1',
    width: 100,
    height: 100,
    offsetX: 300,
    offsetY: 100,
  }
];
  let connectors = [{
    id: "connector1",
    style: {
      strokeColor: '#6BA5D7',
      fill: '#6BA5D7',
      strokeWidth: 2
    },
    targetDecorator: {
      style: {
        fill: '#6BA5D7',
        strokeColor: '#6BA5D7'
      }
    },
    sourceID: "node",
```

```

    targetID: "node1",
    // Set Source Padding value
    sourcePadding:20,
    // Set Target Padding value
    targetPadding:20
  }, ]
export default {
  name: 'app'
  data() {
    return {
      width: "100%",
      height: "350px",
      nodes: nodes,
      connectors: connectors,
      getNodeDefaults: (node) => {
        node.height = 100;
        node.width = 100;
        node.style.fill = '#6BA5D7';
        node.style.strokeColor = 'white';
        return node;
      },
    }
  }
}
</script>
<style>
  @import "../node_modules/@syncfusion/ej2-vue-
  diagrams/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/diagram/connectors/ConnectNode-cs2" %}

### Hit padding

- The [hitPadding](#) property enables you to define the clickable area around the connector path. The default value for hitPadding is 10.
- The following code example illustrates how to specify hit padding for connector.

### APP.VUE

```

<template>
  <div id="app">
    <ejs-diagram id="diagram" :width='width' :height='height'
    :connectors='connectors' :getConnectorDefaults='getConnectorDefaults'
    ></ejs-diagram>
  </div>
</template>
<script>
  import Vue from 'vue';
  import { Diagram, DiagramPlugin, ConnectorEditing, ConnectorConstraints }
  from '@syncfusion/ej2-vue-diagrams';
  Vue.use(DiagramPlugin);
  Diagram.Inject(ConnectorEditing);
  let connectors = [{
    // Name of the connector

```

```

id: "connector1",
type: "Orthogonal",
//set hit padding
hitPadding: 50,
style: {
  strokeColor: '#6BA5D7',
  fill: '#6BA5D7',
  strokeWidth: 2
},
targetDecorator: {
  style: {
    fill: '#6BA5D7',
    strokeColor: '#6BA5D7'
  }
},
sourcePoint: { x: 100, y: 100 },
targetPoint: { x: 300, y: 300 }
}, ]
export default {
  name: 'app',
  data() {
    return {
      width: "100%",
      height: "350px",
      connectors: connectors,
      getConnectorDefaults: (connector) => {
        connector.constraints = ConnectorConstraints.Default |
ConnectorConstraints.DragSegmentThumb;
        return connector;
      },
    }
  }
}
</script>
<style>
  @import "../node_modules/@syncfusion/ej2-vue-
diagrams/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/diagram/connectors/ConnectorHitPadding-cs1" %}

## Flip

The diagram Provides support to flip the connector. The [flip](#) is performed to give the mirrored image of the original element.

The flip types are as follows:

- HorizontalFlip

[Horizontal](#) is used to interchange the connector source and target x points.

- VerticalFlip

[Vertical](#) is used to interchange the connector source and target y points.

- Both

[Both](#) is used to interchange the source point as target point and target point as source point

### APP.VUE

```
<template>
  <div id="app">
    <ejs-diagram id="diagram" :width='width' :height='height'
:connectors='connectors' ></ejs-diagram>
  </div>
</template>
<script>
  import Vue from 'vue';
  import { DiagramPlugin } from '@syncfusion/ej2-vue-diagrams';
  Vue.use(DiagramPlugin);
  let connectors = [{
    // Name of the connector
    id: "connector1",
    style: {
      strokeColor: '#6BA5D7',
      fill: '#6BA5D7',
      strokeWidth: 2
    },
    targetDecorator: {
      style: {
        fill: '#6BA5D7',
        strokeColor: '#6BA5D7'
      }
    },
    sourcePoint: {
      x: 100,
      y: 100
    },
    targetPoint: {
      x: 200,
      y: 200
    },
    // Flip the connector in horizontal direction
    flip:"Horizontal"
  }
]
export default {
  name: 'app'
  data() {
    return {
      width: "100%",
      height: "350px",
      connectors: connectors
    }
  }
}
</script>
<style>
  @import "../../node_modules/@syncfusion/ej2-vue-
diagrams/styles/material.css";
</style>
```



```
{% previewsample "page.domainurl/code-snippet/diagram/connectors/ConnectorsDecorator-cs2" %}
```

Note: The flip is not applicable when the connectors connect in nodes

### Bridging

Line Bridging creates a bridge for lines to smartly cross over other lines, at points of intersection. By default [bridgeDirection](#) is set to top. Depending upon the direction given bridging direction appears. Bridging can be enabled/disabled either with the `[connector.constraints]` or `[diagram.constraints]`. The following code example illustrates how to enable line bridging.

### APP.VUE

```
<template>
  <div id="app">
    <ejs-diagram id="diagram" :width='width' :height='height'
      :nodes='nodes' :connectors='connectors' :constraints='constraints'
      :getNodeDefaults='getNodeDefaults'
      :getConnectorDefaults='getConnectorDefaults'></ejs-diagram>
    </div>
  </template>
<script>
  import Vue from 'vue';
  import { Diagram, DiagramPlugin, DiagramConstraints, ConnectorBridging }
  from '@syncfusion/ej2-vue-diagrams';
  Diagram.Inject(ConnectorBridging);
  Vue.use(DiagramPlugin);
  let nodes = [{
    id: 'Transaction',
    width: 150,
    height: 60,
    offsetX: 300,
    offsetY: 60,
    shape: {
      type: 'Flow',
      shape: 'Terminator'
    },
    annotations: [{
      id: 'labell1',
      content: 'Start Transaction',
      offset: {
        x: 0.5,
        y: 0.5
      }
    }
  ]},
  {
    id: 'Verification',
    width: 150,
    height: 60,
    offsetX: 300,
    offsetY: 250,
    shape: {
      type: 'Flow',
      shape: 'Process'
    }
  },
```

```

        annotations: [{
          id: 'label2',
          content: 'Verification',
          offset: {
            x: 0.5,
            y: 0.5
          }
        }]
      }
    ]
  };
  let connectors = [{
    id: 'connector1',
    type: 'Straight',
    sourceID: 'Transaction',
    targetID: 'Verification'
  }, {
    id: 'connector2',
    type: 'Straight',
    sourcePoint: {
      x: 200,
      y: 130
    },
    targetPoint: {
      x: 400,
      y: 130
    }
  }, {
    id: 'connector3',
    type: 'Straight',
    sourcePoint: {
      x: 200,
      y: 170
    },
    targetPoint: {
      x: 400,
      y: 170
    }
  }
  ]
  export default {
    name: 'app'
    data() {
      return {
        width: "100%",
        height: "350px",
        constraints: DiagramConstraints.Default |
DiagramConstraints.Bridging,
        nodes: nodes,
        connectors: connectors,
        getNodeDefaults: (node) => {
          node.height = 100;
          node.width = 100;
          node.style.fill = '#6BA5D7';
          node.style.strokeColor = 'white';
          return node;
        },
      },
    },
  }

```

```

        getConnectorDefaults: (obj) => {
            obj.style.strokeColor = '#6BA5D7';
            obj.style.fill = '#6BA5D7';
            obj.style.strokeWidth = 2;
            obj.targetDecorator.style.fill = '#6BA5D7';
            obj.targetDecorator.style.strokeColor = '#6BA5D7';
            return obj;
        },
    },
}
}
</script>
<style>
    @import "../node_modules/@syncfusion/ej2-vue-
    diagrams/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/diagram/connectors/ConnectorsBridging-cs1" %}

Note: We Need to inject Connector bridging module into the diagram.

You can use [bridgeSpace](#) property of connectors to define the width for line bridging.

Limitation: Bezier segments do not support bridging.

### Corner radius

Corner radius allows to create connectors with rounded corners. The radius of the rounded corner is set with [cornerRadius](#) property.

### APP.VUE

```

<template>
    <div id="app">
        <ejs-diagram id="diagram" :width='width' :height='height'
        :nodes='nodes' :connectors='connectors'
        :getNodeDefaults='getNodeDefaults'></ejs-diagram>
    </div>
</template>
<script>
    import Vue from 'vue';
    import { DiagramPlugin } from '@syncfusion/ej2-vue-diagrams';
    Vue.use(DiagramPlugin);
    let nodes = [{
        id: 'node1',
        width: 100,
        height: 100,
        offsetX: 100,
        offsetY: 100,
    },
    {
        id: 'node2',
        width: 100,
        height: 100,
        offsetX: 100,
        offsetY: 350,
    },
    ];

```

```

let connectors = [{
  id: "connector1",
  type: 'Orthogonal',
  style: {
    strokeColor: '#6BA5D7',
    fill: '#6BA5D7',
    strokeWidth: 2
  },
  targetDecorator: {
    style: {
      fill: '#6BA5D7',
      strokeColor: '#6BA5D7'
    }
  },
  // Sets the radius for the rounded corner
  cornerRadius: 10,
  sourceID: 'node1',
  targetID: 'node2',
  segments: [{
    type: 'Orthogonal',
    direction: 'Right',
    length: 50
  }],
}],
}]
export default {
  name: 'app'
  data() {
    return {
      width: "100%",
      height: "350px",
      nodes: nodes,
      connectors: connectors,
      getNodeDefaults: (node) => {
        node.height = 100;
        node.width = 100;
        node.style.fill = '#6BA5D7';
        node.style.strokeColor = 'white';
        return node;
      },
    }
  }
}
</script>
<style>
  @import "../node_modules/@syncfusion/ej2-vue-
  diagrams/styles/material.css";
</style>

```

```
{% previewsample "page.domainurl/code-snippet/diagram/connectors/ConnectorsCornerRadius-cs1"
%}
```

### Appearance

- The Connector's [strokeWidth](#), [strokeColor](#), [strokeDashArray](#) and [opacity](#) properties are used to customize the appearance of the connector segments.

- The [visible](#) property of the connector enables or disables the visibility of connector.
- Default values for all the connectors can be set using the `getConnectorDefaults` properties. For example, if all connectors have the same type or having the same property then such properties can be moved into `getConnectorDefaults`.

### Segment Appearance

- The following code example illustrates how to customize the segment appearance.

#### APP.VUE

```
<template>
  <div id="app">
    <ejs-diagram id="diagram" :width='width' :height='height'
:connectors='connectors' :getConnectorDefaults='getConnectorDefaults'
></ejs-diagram>
  </div>
</template>
<script>
  import Vue from 'vue';
  import { DiagramPlugin } from '@syncfusion/ej2-vue-diagrams';
  Vue.use(DiagramPlugin);
  let connectors = [{
    id: "connector1",
    targetDecorator: {
      style: {
        strokeColor: '#6BA5D7',
        fill: '#6BA5D7',
        strokeWidth: 2
      }
    },
    style: {
      // Stroke color
      strokeColor: '#6BA5D7',
      fill: '#6BA5D7',
      // Stroke width of the line
      strokeWidth: 2,
      // Line style
      strokeDashArray: '2,2'
    },
    sourcePoint: {
      x: 100,
      y: 100
    },
    targetPoint: {
      x: 200,
      y: 200
    },
    segments: [{
      type: 'Orthogonal',
      direction: 'Right',
      length: 50
    }],
  },
  {
    {
```

```

    id: "connector2",
    // Set the visibility of the connector to false
    visible: false,
    targetDecorator: {
      style: {
        strokeColor: '#6BA5D7',
        fill: '#6BA5D7',
        strokeWidth: 2
      }
    },
    style: {
      // Stroke color
      strokeColor: '#6BA5D7',
      fill: '#6BA5D7',
      // Stroke width of the line
      strokeWidth: 2,
      // Line style
      strokeDashArray: '2,2'
    },
    sourcePoint: {
      x: 300,
      y: 300
    },
    targetPoint: {
      x: 400,
      y: 400
    },
    segments: [{
      type: 'Orthogonal',
      direction: 'Right',
      length: 50
    }],
  },
}
]
export default {
  name: 'app'
  data() {
    return {
      width: "100%",
      height: "350px",
      connectors: connectors,
      getConnectorDefaults: (obj) => {
        obj.type = 'Orthogonal'
        return obj;
      },
    }
  }
}
</script>
<style>
  @import "../../node_modules/@syncfusion/ej2-vue-
  diagrams/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/diagram/connectors/ConnectorsAppearance-cs1" %}

### Decorator Appearance

- The source decorator's [strokeColor](#), [strokeWidth](#) and [strokeDashArray](#) properties are used to customize the color and width and appearance of the decorator.
- To set the border stroke color, stroke width and stroke dash array for the target decorator, use [strokeColor](#), [strokeWidth](#) and [strokeDashArray](#).
- To set the size for source decorator, use width and height property. Similarly, to set the size for target decorator, use width and height.

The following code example illustrates how to customize the appearance of the decorator.

#### APP.VUE

```
<template>
  <div id="app">
    <ejs-diagram id="diagram" :width='width' :height='height'
:connectors='connectors' ></ejs-diagram>
  </div>
</template>
<script>
  import Vue from 'vue';
  import { DiagramPlugin } from '@syncfusion/ej2-vue-diagrams';
  Vue.use(DiagramPlugin);
  let connectors = [{
    id: "connector1",
    type: 'Straight',
    style: {
      strokeColor: '#6BA5D7',
      fill: '#6BA5D7',
      strokeWidth: 2
    },
    bridgeSpace: 20,
    // Customize the target decorator
    targetDecorator: {
      style: {
        // Fill color of the decorator
        fill: '#6BA5D7',
        // Stroke color of the decorator
        strokeColor: '#6BA5D7'
      }
    },
    sourcePoint: {
      x: 100,
      y: 100
    },
    targetPoint: {
      x: 200,
      y: 200
    }
  }]
  export default {
    name: 'app'
    data() {
      return {
        width: "100%",
```

```

        height: "350px",
        connectors: connectors
    }
}
</script>
<style>
    @import "../node_modules/@syncfusion/ej2-vue-
diagrams/styles/material.css";
</style>

```

```
{% previewsample "page.domainurl/code-snippet/diagram/connectors/ConnectorsDecAppearance-cs1"
%}
```

### Interaction

- Diagram allows to edit the connectors at runtime. To edit the connector segments at runtime, refer to [Connection Editing](#).

### Automatic line routing

Diagram provides additional flexibility to re-route the diagram connectors. A connector will frequently re-route itself when a shape moves next to it. The following screenshot illustrates how the connector automatically re-routes the segments.

- Dependency LineRouting module should be injected to the application as the following code snippet.

```

<script>
import {DiagramPlugin,LineRouting,Diagram,DiagramConstraints } from '@syncfusion/ej2-vue-
diagrams';
Diagram.Inject(LineRouting);
Vue.use(DiagramPlugin);
</script>

```

- Now, the line routing constraints must be included to the default diagram constraints to enable automatic line routing support like below.

```
/
```

- Initialize the Diagram

```
*/
```



```

<ejs-diagram #diagram [constraints]='constraints'>
<template>
<div id="app">
<ejs-diagram :constraints='constraints' ></ejs-diagram>
</div>
</template>
<script>
// Enable line routing constraints.
let constraints = DiagramConstraints.Default | DiagramConstraints.LineRouting;
</script>

```

- The following code block shows how to create the diagram with specifying nodes, connectors, constraints, and necessary modules for line routing.

#### APP.VUE

```

<template>
  <div id="app">
    <ejs-diagram id="diagram" :width='width' :height='height'
:constraints='constraints' :nodes='nodes' :connectors='connectors'
:getNodeDefaults='getNodeDefaults'></ejs-diagram>
  </div>
</template>
<script>
  import Vue from 'vue';
  import { DiagramPlugin, LineRouting, Diagram, DiagramConstraints } from
'@syncfusion/ej2-vue-diagrams';
  Diagram.Inject(LineRouting);
  Vue.use(DiagramPlugin);
  let nodes = [
    { id: 'shape1', offsetX: 100, offsetY: 100, width: 120, height: 50 },
    { id: 'shape2', offsetX: 300, offsetY: 300, width: 120, height: 50 },
    { id: 'shape3', offsetX: 150, offsetY: 200, width: 120, height: 50 }
  ];
  let connectors = [
    { id: 'connector', sourceID: 'shape1', targetID: 'shape2', type:
'Orthogonal' }
  ];
  function getNodeDefaults(obj) {
    obj = { style: { strokeColor: '#6BA5D7', fill: '#6BA5D7' } };
    return obj;
  }
  let constraints = DiagramConstraints.Default |
DiagramConstraints.LineRouting;
  export default {
    name: 'app',
    data () {
      return {

```

```

        width: "100%",
        height: "350px",
        connectors: connectors,
        nodes: nodes,
        constraints: constraints,
        getNodeDefaults: (obj) => {
            obj = { style: { strokeColor: '#6BA5D7', fill: '#6BA5D7' } };
            return obj;
        },
    },
}
}
</script>
<style>
    @import "../node_modules/@syncfusion/ej2-vue-
diagrams/styles/material.css";
    @import "../node_modules/@syncfusion/ej2-popups/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/diagram/connectors/ConnectorsLineRouting-cs1" %}

- In some situations, automatic line routing enabled diagram needs to ignore a specific connector from automatic line routing. So, in this case, auto routing feature can be disabled to the specific connector using the [constraints](#) property of the connector like the following code snippet.

## APP.VUE

```

<template>
    <div id="app">
        <ejs-diagram id="diagram" :width='width' :height='height'
        :constraints='constraints' :nodes='nodes' :connectors='connectors'
        :getNodeDefaults='getNodeDefaults'></ejs-diagram>
    </div>
</template>
<script>
    import Vue from 'vue';
    import {
        DiagramPlugin, LineRouting, Diagram, DiagramConstraints, ConnectorConstraints }
    from '@syncfusion/ej2-vue-diagrams';
    Diagram.Inject(LineRouting);
    Vue.use(DiagramPlugin);
    let nodes = [
        { id: 'shape1', offsetX: 100, offsetY: 100, width: 120, height: 50 },
        { id: 'shape2', offsetX: 350, offsetY: 300, width: 120, height: 50 },
        { id: 'shape3', offsetX: 150, offsetY: 200, width: 120, height: 50 },
        { id: 'shape4', offsetX: 300, offsetY: 200, width: 120, height: 50 }
    ];
    let connectors = [
        { id: 'connector', sourceID: 'shape1', targetID: 'shape2', type:
        'Orthogonal', annotations: [{ offset: .7, content: 'Routing \n enabled',
        style: { fill: "white" } } ] },
        { id: 'connector2', sourceID: 'shape1', targetID: 'shape2', annotations:
        [{ offset: .6, content: 'Routing \n disabled', style: { fill: "white" } } ] },
    ];

```

```

type: 'Orthogonal', constraints: ConnectorConstraints.Default &
~ConnectorConstraints.InheritLineRouting }
];
function getNodeDefaults(obj) {
  obj = { style: { strokeColor: '#6BA5D7', fill: '#6BA5D7' } };
  return obj;
}
let constraints = DiagramConstraints.Default |
DiagramConstraints.LineRouting;
export default {
  name: 'app',
  data () {
    return {
      width: "100%",
      height: "350px",
      connectors:connectors,
      nodes:nodes,
      constraints:constraints,
      getNodeDefaults: (obj) => {
        obj = { style: { strokeColor: '#6BA5D7', fill: '#6BA5D7' } };
        return obj;
      },
    }
  }
}
</script>
<style>
  @import "../node_modules/@syncfusion/ej2-vue-
diagrams/styles/material.css";
  @import "../node_modules/@syncfusion/ej2-popups/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/diagram/connectors/ConnectorsLineRoutingDisabled-cs1" %}

### Constraints

- The [constraints](#) property of connector allows to enable/disable certain features of connectors.
- To enable or disable the constraints refer [constraints](#)

The following code illustrates how to disable selection.

### APP.VUE

```

<template>
  <div id="app">
    <ejs-diagram id="diagram" :width='width' :height='height'
:connectors='connectors' ></ejs-diagram>
  </div>
</template>
<script>
  import Vue from 'vue';
  import { DiagramPlugin,ConnectorConstraints } from '@syncfusion/ej2-vue-
diagrams';

```

```

Vue.use(DiagramPlugin);
let connectors = [{
  id: "connector1",
  // Disables selection constraints
  constraints: ConnectorConstraints.Default &
~ConnectorConstraints.Select,
  type: 'Straight',
  style: {
    strokeColor: '#6BA5D7',
    fill: '#6BA5D7',
    strokeWidth: 2
  },
  targetDecorator: {
    style: {
      fill: '#6BA5D7',
      strokeColor: '#6BA5D7'
    }
  },
  sourcePoint: {
    x: 100,
    y: 100
  },
  targetPoint: {
    x: 200,
    y: 200
  }
}]
export default {
  name: 'app'
  data() {
    return {
      width: "100%",
      height: "350px",
      connectors: connectors
    }
  }
}
</script>
<style>
  @import "../node_modules/@syncfusion/ej2-vue-
  diagrams/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/diagram/connectors/ConnectorsConstraints-cs1" %}

### Custom Properties

- The [addInfo](#) property of connectors allows to maintain additional information to connectors.

```
`javascript
```

```

let connectors: ConnectorModel = {
  id: 'connector1',
  // Defines the information about the connector

```

```

addInfo:'centralconnector',
type: 'Straight',
sourceID: 'Transaction',
targetID: 'Verification'
};
`

```

### Stack Order

- The connectors [zIndex](#) property specifies the stack order of an connector. A connector with greater stack order is always in front of an connector with a lower stack order.

The following code illustrates how to render connector based on the stack order.

### APP.VUE

```

<template>
  <div id="app">
    <ejs-diagram id="diagram" :width='width' :height='height'
      :connectors='connectors' :getConnectorDefaults='getConnectorDefaults'></ejs-
      diagram>
    </div>
  </template>
<script>
  import Vue from 'vue';
  import { DiagramPlugin, ConnectorConstraints } from '@syncfusion/ej2-vue-
  diagrams';
  Vue.use(DiagramPlugin);
  let connectors = [{
    id: 'connector1',
    // Defines the z-index value for the connector
    zIndex: 2,
    type: 'Straight',
    sourcePoint: {
      x: 300,
      y: 100
    },
    targetPoint: {
      x: 300,
      y: 200
    }
  },
  {
    id: 'connector2',
    type: 'Straight',
    // Defines the z-index value for the connector
    zIndex: 1,
    sourcePoint: {
      x: 100,
      y: 100
    },
    targetPoint: {
      x: 200,

```

```

        y: 200
      }
    }
  ]
  export default {
    name: 'app'
    data() {
      return {
        width: "100%",
        height: "350px",
        connectors: connectors,
        getConnectorDefaults: (obj) => {
          obj.style.strokeColor = '#6BA5D7';
          obj.style.fill = '#6BA5D7';
          obj.style.strokeWidth = 2;
          obj.targetDecorator.style.fill = '#6BA5D7';
          obj.targetDecorator.style.strokeColor = '#6BA5D7';
          return obj;
        },
      }
    }
  }
</script>
<style>
  @import "../node_modules/@syncfusion/ej2-vue-
  diagrams/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/diagram/connectors/zindex-cs1" %}

### Enable Connector Splitting

The connectors are used to create a link between two points, ports, or nodes to represent the relationship between them. Split the connector between two nodes when dropping a new node onto an existing connector and create a connection between the new node and existing nodes by setting the [enableConnectorSplit](#) as true. The default value of the [enableConnectorSplit](#) is false.

The following code illustrates how to split the connector and create a connection with new node.

### APP.VUE

```

<template>
  <div id="app">
    <ejs-diagram id="diagram" :width='width' :height='height'
    :connectors='connectors' :nodes='nodes'
    :enableConnectorSplit='enableConnectorSplit'></ejs-diagram>
  </div>
</template>
<script>
  import Vue from 'vue';
  import { DiagramPlugin, Diagram, DiagramConstraints, ConnectorConstraints }
  from '@syncfusion/ej2-vue-diagrams';
  Vue.use(DiagramPlugin);
  let nodes = [
    { id: 'node1', offsetX: 150, offsetY: 150, width: 100, height: 100,
    annotations: [{ content: 'node1' }] },

```

```

    { id: 'node2', offsetX: 650, offsetY: 150, width: 100, height: 100,
      annotations: [{ content: 'node2' }] },
    { id: 'node3', offsetX: 490, offsetY: 290, width: 100, height: 100,
      annotations: [{ content: 'node3' }] }
  ];
  let connectors = [{
    id: 'connector1', sourceID: "node1", targetID: "node2",
    constraints: ConnectorConstraints.Default |
    ConnectorConstraints.AllowDrop
  }];
  export default {
    name: 'app'
    data() {
      return {
        width: "100%",
        height: "350px",
        nodes: nodes,
        connectors: connectors,
        enableConnectorSplit: true
      }
    }
  }
</script>
<style>
  @import "../node_modules/@syncfusion/ej2-vue-
  diagrams/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/diagram/connectors/ConnectorSplit-cs1" %}



### See Also

- [How to add annotations to the connector](#)
- [How to enable/disable the behavior of the node](#)
- [How to add connectors to the symbol palette](#)
- [How to perform the interaction on the connector](#)
- [How to create diagram connectors using drawing tools](#)

## Group in Vue Diagram component

Group is used to cluster multiple nodes and connectors into a single element. It acts like a container for its children (nodes, groups, and connectors). Every change made to the group also affects the children. Child elements can be edited individually.

### Create group

#### Add group when initializing diagram

A group can be added to the diagram model through [nodes](#) collection. To define an object as group, add the child objects to the [children](#) collection of the group. The following code illustrates how to create a group node.

- The [padding](#) property of a group node defines the spacing between the group node's edges and its children.
- While creating group, its child node need to be declared before the group declaration.
- Add a node to the existing group child by using the `diagram.group` method.
- The group's `diagram.unGroup` method is used to define whether the group can be ungrouped or not.
- A group can be added into a child of another group.

### APP.VUE

```
<template>
  <div id="app">
    <ejs-diagram id="diagram" :width='width' :height='height'
:nodes='nodes' :getNodeDefaults='getNodeDefaults' ></ejs-diagram>
  </div>
</template>
<script>
  import Vue from 'vue';
  import { DiagramPlugin } from '@syncfusion/ej2-vue-diagrams';
  Vue.use(DiagramPlugin);
  let nodes = [{
    id: "rectangle1",
    offsetX: 100,
    offsetY: 100,
    width: 100,
    height: 100,
    annotations: [{
      content: 'rectangle1'
    }]
  },
  {
    id: "rectangle2",
    offsetX: 200,
    offsetY: 200,
    width: 100,
    height: 100,
    annotations: [{
      content: 'rectangle2'
    }]
  },
  {
    id: "rectangle3",
```



```

        offsetX: 400,
        offsetY: 300,
        width: 100,
        height: 100,
        style: {
            fill: 'darkCyan',
            strokeWidth: 2
        },
        annotations: [{
            content: 'rectangle2'
        }]
    }
    // Grouping node 1 and node 2 into a single group
    {
        id: 'group',
        children: ['rectangle1', 'rectangle2'],
        padding: {left: 10, right: 10, top: 10, bottom: 10}
    }
]
export default {
    name: 'app'
    data() {
        return {
            width: "100%",
            height: "350px",
            nodes: nodes,
            getNodeDefaults: (node) => {
                node.height = 100;
                node.width = 100;
                node.style.fill = '#6BA5D7';
                node.style.strokeColor = 'white';
                return node;
            },
        }
    }
    mounted: function() {
        let diagramInstance: Diagram;
        let diagramObj: any = document.getElementById("diagram");
        diagramInstance = diagramObj.ej2_instances[0];
        diagramInstance.selectAll();
        // Adding the third node into the existing group
        diagramInstance.group();
    }
}
</script>
<style>
    @import "../node_modules/@syncfusion/ej2-vue-
    diagrams/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/diagram/group/group-cs1" %}

The following code illustrates how a ungroup at runtime.

### APP.VUE

```
<template>
```

```

<div id="app">
  <ejs-diagram id="diagram" :width='width' :height='height'
:nodes='nodes' :getNodeDefaults='getNodeDefaults' ></ejs-diagram>
</div>
</template>
<script>
import Vue from 'vue';
import { DiagramPlugin } from '@syncfusion/ej2-vue-diagrams';
Vue.use(DiagramPlugin);
let nodes = [{
  id: "rectangle1",
  offsetX: 100,
  offsetY: 100,
  width: 100,
  height: 100,
  annotations: [{
    content: 'rectangle1'
  }]
}, {
  id: "rectangle2",
  offsetX: 200,
  offsetY: 200,
  width: 100,
  height: 100,
  annotations: [{
    content: 'rectangle2'
  }]
},
{
  id: 'group',
  children: ['rectangle1', 'rectangle2']
},
]
export default {
  name: 'app'
  data() {
    return {
      width: "100%",
      height: "350px",
      nodes: nodes,
      getNodeDefaults: (node) => {
        node.height = 100;
        node.width = 100;
        node.style.fill = '#6BA5D7';
        node.style.strokeColor = 'white';
        return node;
      },
    }
  }
  mounted: function() {
    let diagramInstance: Diagram;
    let diagramObj: any = document.getElementById("diagram");
    diagramInstance = diagramObj.ej2_instances[0];
    diagramInstance.selectAll();
    // Ungroup the selected group into nodes
    diagramInstance.unGroup();
  }
}

```

```

}
</script>
<style>
  @import ".../node_modules/@syncfusion/ej2-vue-
  diagrams/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/diagram/group/ungroup-cs1" %}

### Add group at runtime

A group node can be added at runtime by using the client-side method `diagram.add`.

The following code illustrates how a group node is added at runtime.

### APP.VUE

```

<template>
  <div id="app">
    <ejs-diagram id="diagram" :width='width' :height='height'
    :nodes='nodes' :getNodeDefaults='getNodeDefaults'></ejs-diagram>
  </div>
</template>
<script>
  import Vue from 'vue';
  import { DiagramPlugin, NodeModel } from '@syncfusion/ej2-vue-diagrams';
  Vue.use(DiagramPlugin);
  let nodes = [{
    id: "rectangle1",
    offsetX: 100,
    offsetY: 100,
    width: 100,
    height: 100,
    annotations: [{
      content: 'rectangle1'
    }]
  },
  {
    id: "rectangle2",
    offsetX: 200,
    offsetY: 200,
    width: 100,
    height: 100,
    annotations: [{
      content: 'rectangle2'
    }]
  }
  ];
  let group: NodeModel = {
    id: 'group2',
    children: ['rectangle1', 'rectangle2']
  };
  export default {
    name: 'app'
    data() {
      return {
        width: "100%",
        height: "350px",
        nodes: nodes,

```

```

        getNodeDefaults: (node) => {
            node.height = 100;
            node.width = 100;
            node.style.fill = '#6BA5D7';
            node.style.strokeColor = 'white';
            return node;
        },
    },
}
}
mounted: function() {
    let diagramInstance: Diagram;
    let diagramObj: any = document.getElementById("diagram");
    diagramInstance = diagramObj.ej2_instances[0];
    diagramInstance.selectAll();
    // Add the group into the diagram
    diagramInstance.add(group);
}
}
</script>
<style>
    @import "../node_modules/@syncfusion/ej2-vue-
    diagrams/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/diagram/group/groupadd-cs1" %}

### Add children To group at runtime

A childNode can be added to the specified Group at runtime by utilizing the client-side method **diagramInstance.addChildToGroup**.

This functionality is achieved by passing the group and existing children as arguments to the method.

The following code illustrates how a child node and a group node can be passed as arguments to the method and executed at runtime.

```
`html
```

```
diagramInstance.addChildToGroup(groupNode, childNode);
```

```
,
```

### Remove children from group at runtime

A specific child from a group node can be removed at runtime by utilizing the client-side method **diagramInstance.removeChildFromGroup**.

This functionality is achieved by passing the group and its children as arguments to the method.

The following code illustrates how a child node is removed from a group at runtime.

```
`html
```

```
diagramInstance.removeChildFromGroup (groupNode, childNode);
```

```
,
```

### APP.VUE

```
<template>
```

```

<div id="app">
  <button @click="addChild">addChild</button>
  <button @click="removeChild">removeChild</button>
  <ejs-diagram id="diagram" :width='width' :height='height'
:nodes='nodes'></ejs-diagram>
</div>
</template>
<script>
  import Vue from 'vue';
  import { DiagramPlugin } from '@syncfusion/ej2-vue-diagrams';
  Vue.use(DiagramPlugin);
  let node = {
    id: 'node1', width: 150, height: 100, offsetX: 100, offsetY: 100,
    annotations: [{ content: 'Node1' }]
  };
  let node2 = {
    id: 'node2', width: 80, height: 130, offsetX: 200, offsetY: 200,
    annotations: [{ content: 'Node2' }]
  };
  let group = {
    id: 'group1', children: ['node1', 'node2']
  };
  let node3 = {
    id: 'node3', width: 100, height: 100, offsetX: 300, offsetY: 300,
    annotations: [{ content: 'Node3' }]
  };
  export default {
    name: 'app',
    data() {
      return {
        width: "100%",
        height: "450px",
        nodes: [node,node2,node3,group],
      }
    },
    methods: {
      addChild: function () {
        let diagram = this.$refs.diagram.ej2Instances;
        diagram.addChildToGroup(group, 'node3');
      },
      removeChild: function () {
        let diagram = this.$refs.diagram.ej2Instances;
        diagram.removeChildFromGroup(group, 'node3');
      },
    },
  }
</script>
<style>
  @import "../../node_modules/@syncfusion/ej2-vue-
diagrams/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/diagram/group/groupchild-cs1" %}

## Container

Containers are used to automatically measure and arrange the size and position of the child elements in a predefined manner. There are two types of containers available.

### Canvas

- The canvas panel supports absolute positioning and provides the least layout functionality to its contained diagram elements.
- Canvas allows you to position its contained elements by using the margin and alignment properties.
- Rendering alone possible in canvas container.
- It allows elements to be either vertically or horizontally aligned.
- Child can be defined with the collection [canvas.children](#) property.
- Basic element can be defined with the collection of `basicElements`.

### Stack

- Stack panel is used to arrange its children in a single line or stack order, either vertically or horizontally.
- It controls spacing by setting margin properties of child and padding properties of group. By default, a stack panel's [orientation](#) is vertical.

The following code illustrates how to add a stack panel.

### APP.VUE

```
<template>
  <div id="app">
    <ejs-diagram id="diagram" :width='width' :height='height'
      :nodes='nodes' :setNodeTemplate='setNodeTemplate'
      :getNodeDefaults='getNodeDefaults'></ejs-diagram>
  </div>
</template>
<script>
  import Vue from 'vue';
  import { DiagramPlugin, NodeModel, StackPanel, TextElement } from
  '@syncfusion/ej2-vue-diagrams';
  Vue.use(DiagramPlugin);
  let count = 11;
  function getTextElement(text: string) {
    let textElement: TextElement = new TextElement();
    textElement.id = "text" + count;
    textElement.width = 50;
    textElement.height = 20;
    textElement.content = text;
    count++;
    return textElement;
  }
  function addRows(column: StackPanel) {
    column.children.push(getTextElement('Row1'));
    column.children.push(getTextElement('Row2'));
    column.children.push(getTextElement('Row3'));
    column.children.push(getTextElement('Row4'));
  }
</script>
```

```

    }
    let nodes = [
      {
        id: 'node5',
        width: 100,
        height: 100,
        offsetX: 100,
        offsetY: 100,
        style: {
          strokeColor: '#6BA5D7',
          fill: '#6BA5D7'
        },
        annotations: [{
          content: 'Custom Template',
          offset: {
            y: 1
          },
          verticalAlignment: 'Top'
        }]
      }
    ]
  }
  export default {
    name: 'app'
    data () {
      return {
        width: "100%",
        height: "350px",
        nodes: nodes,
        getNodeDefaults: (node: NodeModel) => {
          node.height = 100;
          node.width = 100;
          node.style.fill = '#6BA5D7';
          node.style.strokeColor = 'white';
          return node;
        },
        setNodeTemplate: (obj: NodeModel, diagram: Diagram): StackPanel => {
          if (obj.id.indexOf('node5') !== -1) {
            // It will be replaced with grid panel
            let table: StackPanel = new StackPanel();
            table.orientation = 'Horizontal';
            table.padding.left
            let column1: StackPanel = new StackPanel();
            column1.children = [];
            column1.children.push(getTextElement('Column1'));
            addRows(column1);
            let column2: StackPanel = new StackPanel();
            column2.children = [];
            column2.children.push(getTextElement('Column2'));
            addRows(column2);
            table.children = [column1, column2];
            return table;
          }
          return null;
        },
      },
    }
  }
}

```

```

</script>
<style>
  @import "../node_modules/@syncfusion/ej2-vue-
  diagrams/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/diagram/group/canvas-cs1" %}

### Difference between a basic group and containers

| Group | Container |

| ----- | ----- |

| It arranges the child elements based on the child elements position and size properties. | Each container has a predefined behavior to measure and arrange its child elements. Canvas and stack containers are supported in the diagram. |

| The Padding, Min, and Max Size properties are not applicable for basic group. | It is applicable for container. |

| The Children's margin and alignment properties are not applicable for basic group. | It is applicable for container. |

### Interaction

You can edit the group and its children at runtime. For more information about how to interact with a group, refer to [Edit Groups](#).

### See Also

- [How to add annotations to the node](#)
- [How to add ports to the node](#)
- [How to enable/disable the behavior of the node](#)
- [How to add nodes to the symbol palette](#)
- [How to create diagram nodes using drawing tools](#)
- [How to perform the interaction on the group](#)

### Swim lane in Vue Diagram component

Swimlane is a type of diagram nodes, which is typically used to visualize the relationship between a business process and the department responsible for it by focusing on the logical relationships between activities.

#### Create a swimlane

To create a swimlane, the type of shape should be set as [swimlane](#). By Default swimlane's are arranged vertically.

The following code example illustrates how to define a swimlane object.

#### APP.VUE

```

<template>
  <div id="app">
    <ejs-diagram id="diagram" :width='width' :height='height'
    :nodes='nodes' ></ejs-diagram>
  </div>

```



```

</template>
<script>
  import Vue from 'vue';
  import { DiagramPlugin } from '@syncfusion/ej2-vue-diagrams';
  Vue.use(DiagramPlugin);
  let nodes = [{
    shape: {
      // Set the node type as swimlane
      type: 'SwimLane',
      lanes: [
        {
          id: 'stackCanvas1',
          height: 100,
        },
      ],
      phases: [
        {
          id: 'phase1', offset: 170,
          header: { annotation: { content: 'Phase' } }
        },
      ],
      phaseSize: 20,
    },
    offsetX: 300, offsetY: 200,
    height: 200,
    width: 350
  }]
  export default {
    name: 'app'
    data() {
      return {
        width: "100%",
        height: "350px",
        nodes: nodes,
      }
    }
  }
</script>
<style>
  @import "../node_modules/@syncfusion/ej2-vue-diagrams/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/diagram/swimlane/swimlaneheader-cs1" %}

### Headers

Header was the primary element for swimlanes. The [header](#) property of swimlane allows you to define its textual description and to customize its appearance.

Note: By using this header, the swimlane interaction will be performed, like selection, dragging, etc.

The following code example illustrates how to define a swimlane header.

### APP.VUE

```

<template>
  <div id="app">

```

```

    <ejs-diagram id="diagram" :width='width' :height='height'
:nodes='nodes' ></ejs-diagram>
  </div>
</template>
<script>
  import Vue from 'vue';
  import { DiagramPlugin } from '@syncfusion/ej2-vue-diagrams';
  Vue.use(DiagramPlugin);
  let nodes = [{
    shape: {
      type: 'SwimLane',
      orientation: 'Horizontal',
      // Intialize header to swimlane
      header: {
        annotation: { content: 'ONLINE PURCHASE STATUS', style:
{ fill: '#111111' } },
        height: 50, style: { fontSize: 11 },
      },
      lanes: [
        {
          id: 'stackCanvas1',
          height: 100,
        },
      ],
      phases: [
        {
          id: 'phase1', offset: 170,
          header: { annotation: { content: 'Phase' } }
        },
      ],
      phaseSize: 20,
    },
    offsetX: 300, offsetY: 200,
    height: 200,
    width: 350
  ]
}
export default {
  name: 'app'
  data() {
    return {
      width: "100%",
      height: "350px",
      nodes: nodes,
    }
  }
}
</script>
<style>
  @import "../../node_modules/@syncfusion/ej2-vue-
diagrams/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/diagram/swimlane/swimlaneheader-cs2" %}

### Customization of headers

The height and width of swimlane header can be customized with [weight](#) and [height](#) properties of swimlane header. set fill color of header by using the [style](#) property. The orientation of swimlane can be customized with the [orientation](#) property of the header.

Note: By default the swimlane orientation has Horizontal.

The following code example illustrates how to customize the swimlane header..

#### APP.VUE

```
<template>
  <div id="app">
    <ejs-diagram id="diagram" :width='width' :height='height'
:nodes='nodes' ></ejs-diagram>
  </div>
</template>
<script>
import Vue from 'vue';
import { DiagramPlugin } from '@syncfusion/ej2-vue-diagrams';
Vue.use(DiagramPlugin);
let nodes = [{
  shape: {
    type: 'SwimLane',
    orientation: 'Horizontal',
    // customize the swimlane header
    header: {
      annotation: { content: 'SALES PROCESS FLOW CHART', },
      height: 70, style: { fontSize: 11 }, style: { fill: 'pink'
},
    },
    lanes: [
      {
        id: 'stackCanvas1',
        height: 100,
      },
    ],
    phases: [
      {
        id: 'phase1', offset: 170,
        header: { annotation: { content: 'Phase' } }
      },
    ],
    phaseSize: 20,
  },
  offsetX: 300, offsetY: 200,
  height: 200,
  width: 350
}]
export default {
  name: 'app'
  data() {
    return {
      width: "100%",
      height: "350px",
      nodes: nodes,
    }
  }
}
```

```

    }
  }
</script>
<style>
  @import ".../node_modules/@syncfusion/ej2-vue-
  diagrams/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/diagram/swimlane/headercustomize-cs1" %}

#### *Dynamic customization of swimlane header*

You can customize the swimlane header style and text properties dynamically. The following code illustrates how to dynamically customize the lane header.

#### **APP.VUE**

```

<template>
  <div id="app">
    <ejs-diagram id="diagram" :width='width' :height='height'
    :nodes='nodes' ></ejs-diagram>
  </div>
</template>
<script>
  import Vue from 'vue';
  import { DiagramPlugin } from '@syncfusion/ej2-vue-diagrams';
  Vue.use(DiagramPlugin);
  let nodes = [{
    shape: {
      type: 'SwimLane',
      orientation: 'Horizontal',
      // customize the swimlane header
      header: {
        annotation: { content: 'SALES PROCESS FLOW CHART', },
        height: 70, style: { fontSize: 11 }, style: { fill: 'pink' }
      },
    },
    lanes: [
      {
        id: 'stackCanvas1',
        height: 100,
      },
    ],
    phases: [
      {
        id: 'phase1', offset: 170,
        header: { annotation: { content: 'Phase' } }
      },
    ],
    phaseSize: 20,
  },
  {
    offsetX: 300, offsetY: 200,
    height: 200,
    width: 350
  }
  ]
  export default {
    name: 'app'
  }

```

```

data() {
  return {
    width: "100%",
    height: "350px",
    nodes: nodes,
  }
}
mounted: function() {
  let diagramInstance: Diagram;
  let diagramObj: any = document.getElementById("diagram");
  diagramInstance = diagramObj.ej2_instances[0];
  diagramInstance.nodes[0].shape.header.style.fill = 'red'
  diagramInstance.dataBind();
}
}
</script>
<style>
  @import "../node_modules/@syncfusion/ej2-vue-
  diagrams/styles/material.css";
</style>

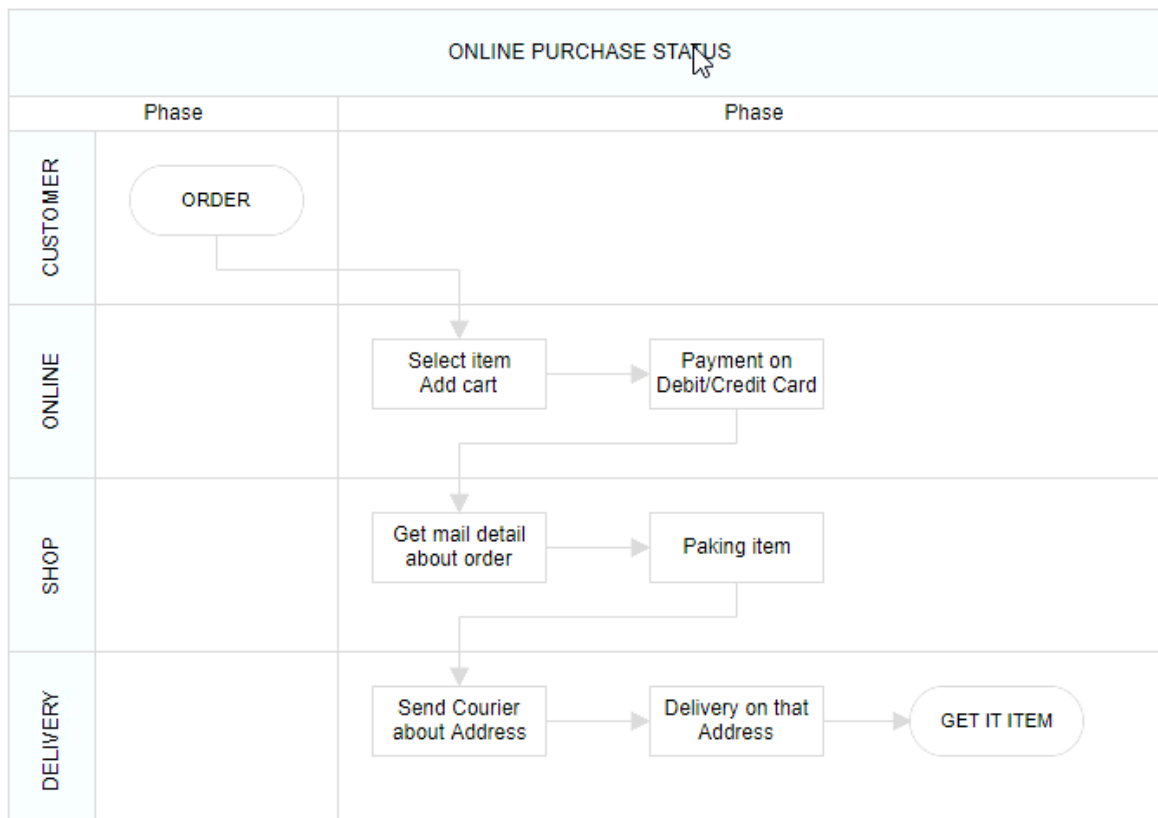
```

{% previewsample "page.domainurl/code-snippet/diagram/swimlane/dynamicheader-cs1" %}

#### Header editing

Diagram provides the support to edit swimlane headers at runtime. We achieve the header editing by double click event. Double clicking the header label will enables the editing of that.

The following image illustrates how to edit the swimlane header.



## Lanes

Lane is a functional unit or a responsible department of a business process that helps to map a process within the functional unit or in between other functional units.

The number of [lanes](#) can be added to swimlane. The lanes are automatically stacked inside swimlane based on the order they are added.

### Create an empty lane

- The lanes `id` is used to define the name of the lane and its further used to find the lane at runtime and do any customization.
- We can provide additional information to the lane by using the [addInfo](#) property of the lane.

The following code example illustrates how to define a swimlane with lane.

### APP.VUE

```
<template>
  <div id="app">
    <ejs-diagram id="diagram" :width='width' :height='height'
:nodes='nodes' ></ejs-diagram>
  </div>
</template>
<script>
  import Vue from 'vue';
  import { DiagramPlugin } from '@syncfusion/ej2-vue-diagrams';
  Vue.use(DiagramPlugin);
  let nodes = [{
    shape: {
      type: 'SwimLane',
      orientation: 'Horizontal',
      header: {
        annotation: { content: 'ONLINE PURCHASE STATUS', style:
{ fill: '#111111' } } },
        height: 50, style: { fontSize: 11 },
      },
      // initialize the lane of swimlane
      lanes: [
        {
          id: 'stackCanvas1',
          // set the lane height
          height: 100,
          // set the lane info
          addInfo:{name:'lane1'}
        },
      ],
      phases: [
        {
          id: 'phase1', offset: 170,
          header: { annotation: { content: 'Phase' } }
        },
      ],
      phaseSize: 20,
    },
    offsetX: 300, offsetY: 200,
    height: 200,
```

```

        width: 350
    ]]
    export default {
      name: 'app'
      data() {
        return {
          width: "100%",
          height: "350px",
          nodes: nodes,
        }
      }
    }
  </script>
<style>
  @import "../node_modules/@syncfusion/ej2-vue-
  diagrams/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/diagram/swimlane/emptylane-cs1" %}

[Create lane header](#)

- The [header](#) property of lane allows you to textually describe the lane and to customize the appearance of the description.

The following code example illustrates how to define a lane header.

#### APP.VUE

```

<template>
  <div id="app">
    <ejs-diagram id="diagram" :width='width' :height='height'
    :nodes='nodes' ></ejs-diagram>
  </div>
</template>
<script>
  import Vue from 'vue';
  import { DiagramPlugin } from '@syncfusion/ej2-vue-diagrams';
  Vue.use(DiagramPlugin);
  let nodes = [{
    shape: {
      type: 'SwimLane',
      orientation: 'Horizontal',
      // Intialize header to swimlane
      header: {
        annotation: { content: 'ONLINE PURCHASE STATUS', style:
{ fill: '#111111' } },
        height: 50, style: { fontSize: 11 },
      },
      // Intialize lane to swimlane
      lanes: [
        {
          id: 'stackCanvas1',
          height: 100,
          // Intialize header to lane
          header: {

```

```

        annotation: { content: 'CUSTOMER' }, width: 50,
        style: { fontSize: 11 }
      },
    ],
    phases: [
      {
        id: 'phase1', offset: 170,
        header: { annotation: { content: 'Phase' } }
      },
    ],
    phaseSize: 20,
  },
  offsetX: 300, offsetY: 200,
  height: 200,
  width: 350
}]
export default {
  name: 'app'
  data() {
    return {
      width: "100%",
      height: "350px",
      nodes: nodes,
    }
  }
}
</script>
<style>
  @import "../node_modules/@syncfusion/ej2-vue-
  diagrams/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/diagram/swimlane/laneheader-cs1" %}

### Customizing lane header

- The size of lane can be controlled by using the [width](#) and [height](#) properties of the lane.
- The appearance of the lane can be set by using the [style](#) properties.

The following code example illustrates how to customize the lane header.

#### APP.VUE

```

<template>
  <div id="app">
    <ejs-diagram id="diagram" :width='width' :height='height'
    :nodes='nodes' ></ejs-diagram>
  </div>
</template>
<script>
  import Vue from 'vue';
  import { DiagramPlugin } from '@syncfusion/ej2-vue-diagrams';
  Vue.use(DiagramPlugin);
  let nodes = [{
    shape: {

```



```

        type: 'SwimLane',
        orientation: 'Horizontal',
        //Intialize header to swimlane
        header: {
            annotation: { content: 'ONLINE PURCHASE STATUS', style:
{ fill: '#111111' } },
            height: 50, style: { fontSize: 11 },
        },
        lanes: [
            {
                id: 'stackCanvas1',
                height: 100,
                // customization of lane header
                header: {
                    annotation: { content: 'Online Consumer' }, width:
30,
                    style: { fontSize: 11 },style: { fill: 'red' }
                },
            },
        ],
        phases: [
            {
                id: 'phase1', offset: 170,
                header: { annotation: { content: 'Phase' } }
            },
        ],
        phaseSize: 20,
    },
    offsetX: 300, offsetY: 200,
    height: 200,
    width: 350
}]
export default {
    name: 'app'
    data() {
        return {
            width: "100%",
            height: "350px",
            nodes: nodes,
        }
    }
}
</script>
<style>
    @import "../node_modules/@syncfusion/ej2-vue-
diagrams/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/diagram/swimlane/laneheadercustomize-cs1" %}

#### [Dynamic customization of lane header](#)

You can customize the lane header style and text properties dynamically. The following code illustrates how to dynamically customize the lane header.

#### **APP.VUE**

```

<template>
  <div id="app">
    <ejs-diagram id="diagram" :width='width' :height='height'
:nodes='nodes' ></ejs-diagram>
  </div>
</template>
<script>
  import Vue from 'vue';
  import { DiagramPlugin } from '@syncfusion/ej2-vue-diagrams';
  Vue.use(DiagramPlugin);
  let nodes = [{
    shape: {
      type: 'SwimLane',
      orientation: 'Horizontal',
      //Intialize header to swimlane
      header: {
        annotation: { content: 'ONLINE PURCHASE STATUS', style:
{ fill: '#111111' } },
        height: 50, style: { fontSize: 11 },
      },
      lanes: [
        {
          id: 'stackCanvas1',
          height: 100,
          // customization of lane header
          header: {
            annotation: { content: 'Online Consumer' }, width:
30,
            style: { fontSize: 11 }, style: { fill: 'red' }
          },
        },
      ],
      phases: [
        {
          id: 'phase1', offset: 170,
          header: { annotation: { content: 'Phase' } }
        },
      ],
      phaseSize: 20,
    },
    offsetX: 300, offsetY: 200,
    height: 200,
    width: 350
  ]
}
export default {
  name: 'app'
  data() {
    return {
      width: "100%",
      height: "350px",
      nodes: nodes,
    }
  }
  mounted: function() {
    let diagramInstance: Diagram;
    let diagramObj: any = document.getElementById("diagram");
    diagramInstance = diagramObj.ej2_instances[0];
  }
}

```

```

        let lane : nodeModel = diagramInstance.nodes[0];
        lane.shape.lanes[0].header.style.fill = 'red';
        diagramInstance.dataBind();
    }
}
</script>
<style>
    @import "../../node_modules/@syncfusion/ej2-vue-
    diagrams/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/diagram/swimlane/dynamiclaneheader-cs1" %}

#### Add lane at runtime

You can add the a lane at runtime by using the client side API method called `addLanes`.The following code illustrates how to dynamically add lane to swimlane.

#### APP.VUE

```

<template>
    <div id="app">
        <ejs-diagram id="diagram" :width='width':height='height'
        :nodes='nodes' ></ejs-diagram>
    </div>
</template>
<script>
    import Vue from 'vue';
    import { DiagramPlugin } from '@syncfusion/ej2-vue-diagrams';
    Vue.use(DiagramPlugin);
    let nodes = [{
        shape: {
            type: 'SwimLane',
            orientation: 'Horizontal',
            //Intialize header to swimlane
            header: {
                annotation: { content: 'ONLINE PURCHASE STATUS', style:
{ fill: '#111111' } } },
                height: 50, style: { fontSize: 11 },
            },
            lanes: [
                {
                    id: 'stackCanvas1',
                    height: 100,
                    // customization of lane header
                    header: {
                        annotation: { content: 'Online Consumer' }, width:
30,
                        style: { fontSize: 11 },style: { fill: 'red' }
                    },
                },
            ],
            phases: [
                {
                    id: 'phasel', offset: 170,
                    header: { annotation: { content: 'Phase' } }
                },
            ],
        },
    ]
}

```

```

    ],
    phaseSize: 20,
  },
  offsetX: 300, offsetY: 200,
  height: 200,
  width: 350
}]
export default {
  name: 'app'
  data() {
    return {
      width: "100%",
      height: "350px",
      nodes: nodes,
    }
  }
  mounted: function() {
    let diagramInstance: Diagram;
    let diagramObj: any = document.getElementById("diagram");
    diagramInstance = diagramObj.ej2_instances[0];
    let lane = [{id:"lane1",height:100,}];
    diagramInstance.addLanes(diagramInstance.nodes[0],lane,1);
  }
}
</script>
<style>
  @import "../node_modules/@syncfusion/ej2-vue-
  diagrams/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/diagram/swimlane/addlanes-cs1" %}

#### [Add children to lane](#)

To add nodes to lane,you should add [children](#) collection of the lane.

The following code example illustrates how to add nodes to lane.

#### **APP.VUE**

```

<template>
  <div id="app">
    <ejs-diagram id="diagram" :width='width' :height='height'
    :nodes='nodes' ></ejs-diagram>
  </div>
</template>
<script>
  import Vue from 'vue';
  import { DiagramPlugin } from '@syncfusion/ej2-vue-diagrams';
  Vue.use(DiagramPlugin);
  let nodes = [{
    shape: {
      type: 'SwimLane',
      orientation: 'Horizontal',
      // Intialize header to swimlane
      header: {
        annotation: { content: 'ONLINE PURCHASE STATUS', style:
        { fill: '#111111' } } },

```

```

        height: 50, style: { fontSize: 11 },
      },
      lanes: [
        {
          id: 'stackCanvas1',
          height: 100,
          header: {
            annotation: { content: 'CUSTOMER' }, width: 50,
            style: { fontSize: 11 }
          },
          // Set the children of lane
          children: [
            {
              id: 'node1',
              annotations: [
                {
                  content: 'Consumer learns \n of
product',
                  style: { fontSize: 11 }
                }
              ],
              margin: { left: 60, top: 30 },
              height: 40, width: 100,
            }, {
              id: 'node2',
              shape: { type: 'Flow', shape: 'Decision' },
              annotations: [
                {
                  content: 'Does \n Consumer want \nthe
product',
                  style: { fontSize: 11 }
                }
              ],
              margin: { left: 200, top: 20 },
              height: 60, width: 120,
            },
          ],
        },
      ],
      phases: [
        {
          id: 'phase1', offset: 170,
          header: { annotation: { content: 'Phase' } }
        },
      ],
      phaseSize: 20,
    },
    offsetX: 300, offsetY: 200,
    height: 200,
    width: 350
  ]
}
export default {
  name: 'app'
  data() {
    return {
      width: "100%",
      height: "350px",
    }
  }
}

```

```
        nodes: nodes,
      }
    }
  }
</script>
<style>
  @import "../node_modules/@syncfusion/ej2-vue-
  diagrams/styles/material.css";
</style>
```

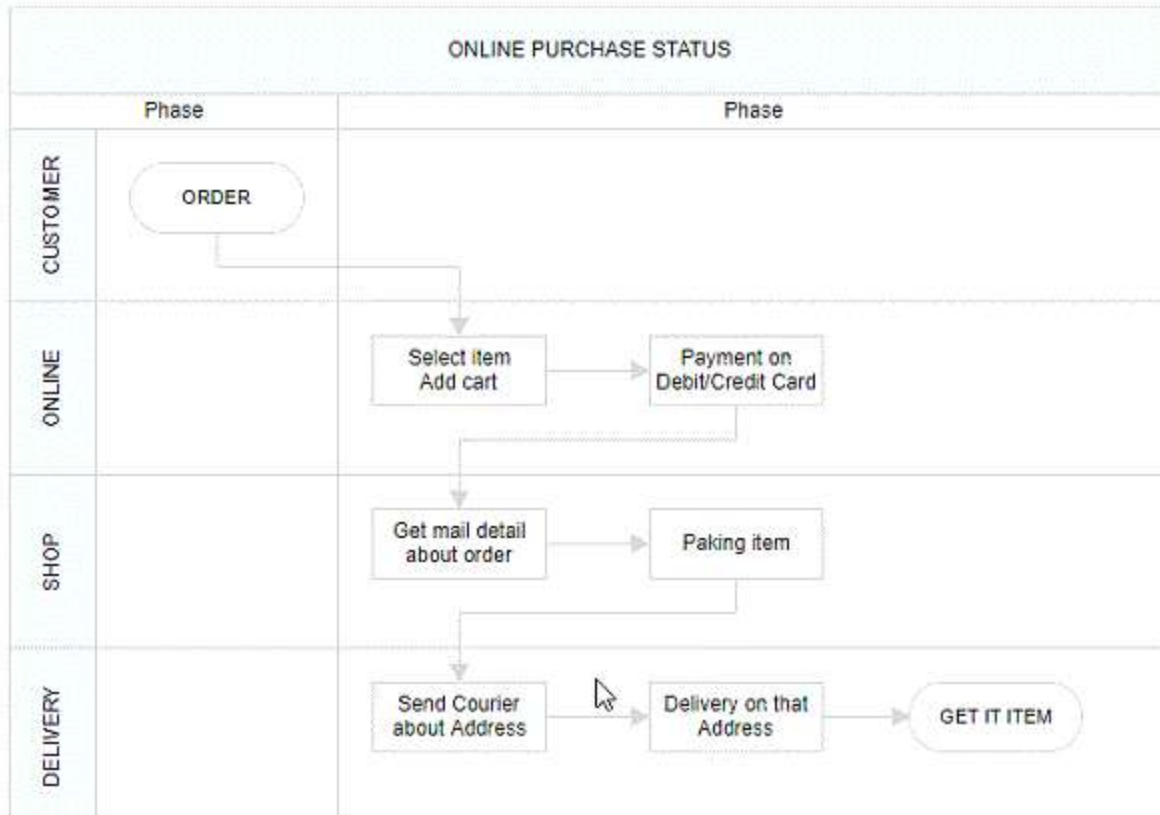
{% previewsample "page.domainurl/code-snippet/diagram/swimlane/lanechildern-cs1" %}

*Lane interaction*

*Resizing lane*

- Lane can be resized in the bottom and left direction.
- Lane can be resized by using resize selector of the lane.
- Once you can resize the lane, the swimlane will be resized automatically.
- The lane can be resized either resizing the selector or the tight bounds of the child object. If the child node move to edge of the lane it can be automatically resized.

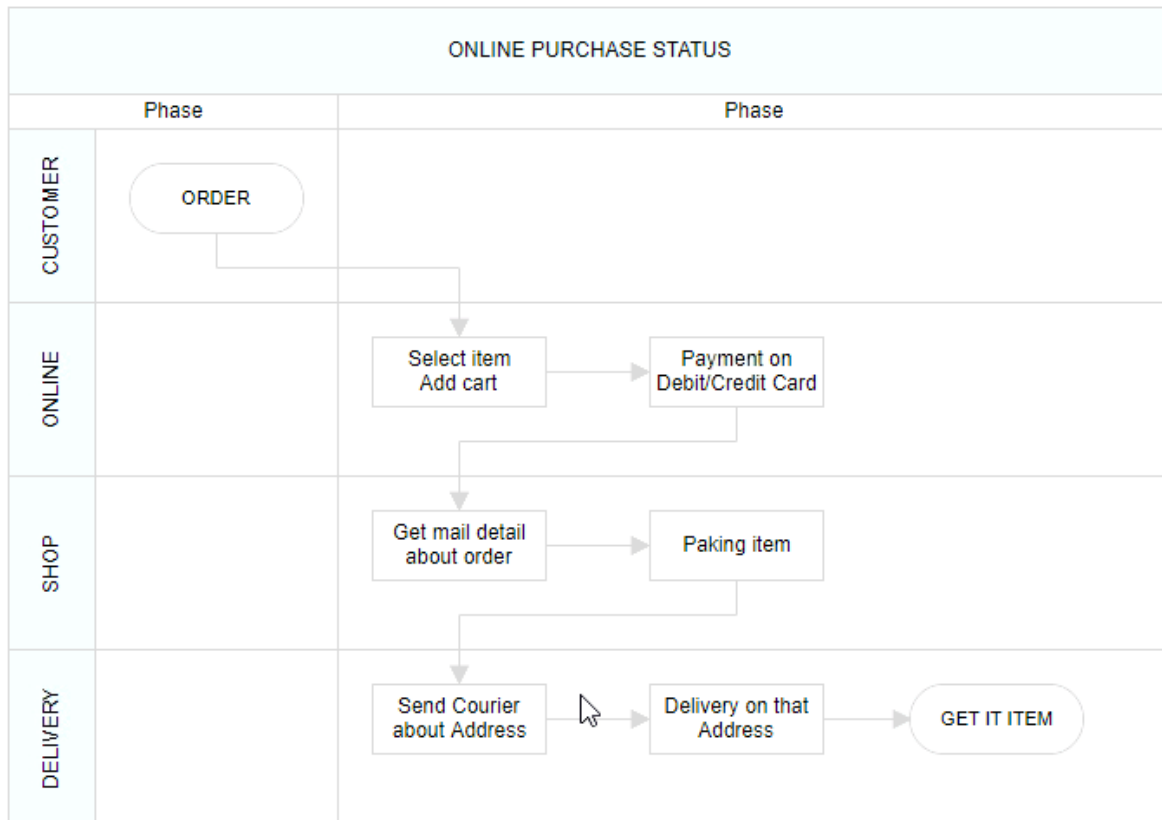
The following image illustrates how resize the lane.



### Lane swapping

- Lanes can be swapped using drag the lanes over another lane.
- Helper should intimate the insertion point while lane swapping.

The following image illustrates how swapping the lane.



### Disable Swimlane Lane swapping

You can disable swimlane lane swapping by using the property called `canMove`.

The following code illustrates how to disable swimlane lane swapping.

#### APP.VUE

```

<template>
  <div id="app">
    <ejs-diagram id="diagram" :width='width' :height='height'
    :nodes='nodes' ></ejs-diagram>
  </div>
</template>
<script>
  import Vue from 'vue';
  import { DiagramPlugin } from '@syncfusion/ej2-vue-diagrams';
  Vue.use(DiagramPlugin);
  let nodes = [{
    shape: {
      type: 'SwimLane',
      orientation: 'Horizontal',
      //Intialize header to swimlane
      header: {
        annotation: { content: 'ONLINE PURCHASE STATUS', style:
{ fill: '#111111' } },
        height: 50, style: { fontSize: 11 },
      },
    },
  ],

```



```

        lanes: [
          {
            id: 'stackCanvas1',
            height: 100,
            // customization of lane header
            header: {
              annotation: { content: 'Online Consumer' }, width:
30,
              style: { fontSize: 11, fill: 'red' }
            },
            canMove: false,
          },
        ],
        phases: [
          {
            id: 'phase1', offset: 170,
            header: { annotation: { content: 'Phase' } }
          },
        ],
        phaseSize: 20,
      },
      offsetX: 300, offsetY: 200,
      height: 200,
      width: 350
    ]
  }
}
export default {
  name: 'app'
  data() {
    return {
      width: "100%",
      height: "350px",
      nodes: nodes,
    }
  }
  mounted: function() {
    let diagramInstance: Diagram;
    let diagramObj: any = document.getElementById("diagram");
    diagramInstance = diagramObj.ej2_instances[0];
    let lane = [{id:"lane1",height:100,canMove: false}];
    diagramInstance.addLanes(diagramInstance.nodes[0], lane, 1);
  }
}
</script>
<style>
  @import "../node_modules/@syncfusion/ej2-vue-
diagrams/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/diagram/swimlane/addlanes-cs2" %}

[Resize helper](#)

- The special resize helper will be used to resize the lanes.
- The resize cursor will be available on the left and bottom direction alone.
- Once resize the lane the swimlane will be resized automatically.

*Children interaction in lanes*

- You can resize the child node within swimlanes.
- You can drag the child nodes within lane.
- Interchange the child nodes from one lane to another lane.
- Drag and drop the child nodes from lane to diagram.
- Drag and drop the child nodes from diagram to lane.
- Based on the child node interactions, the lane size should be updated.

The following image illustrates children interaction in lane.

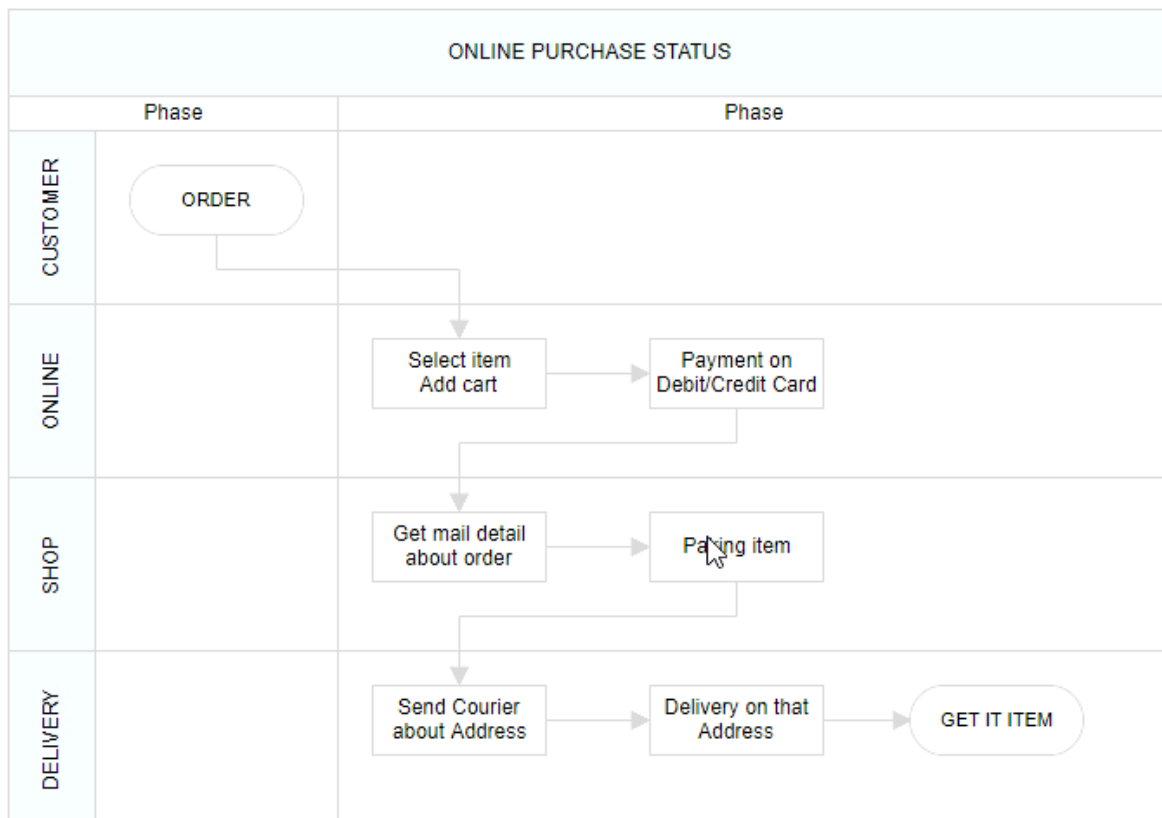
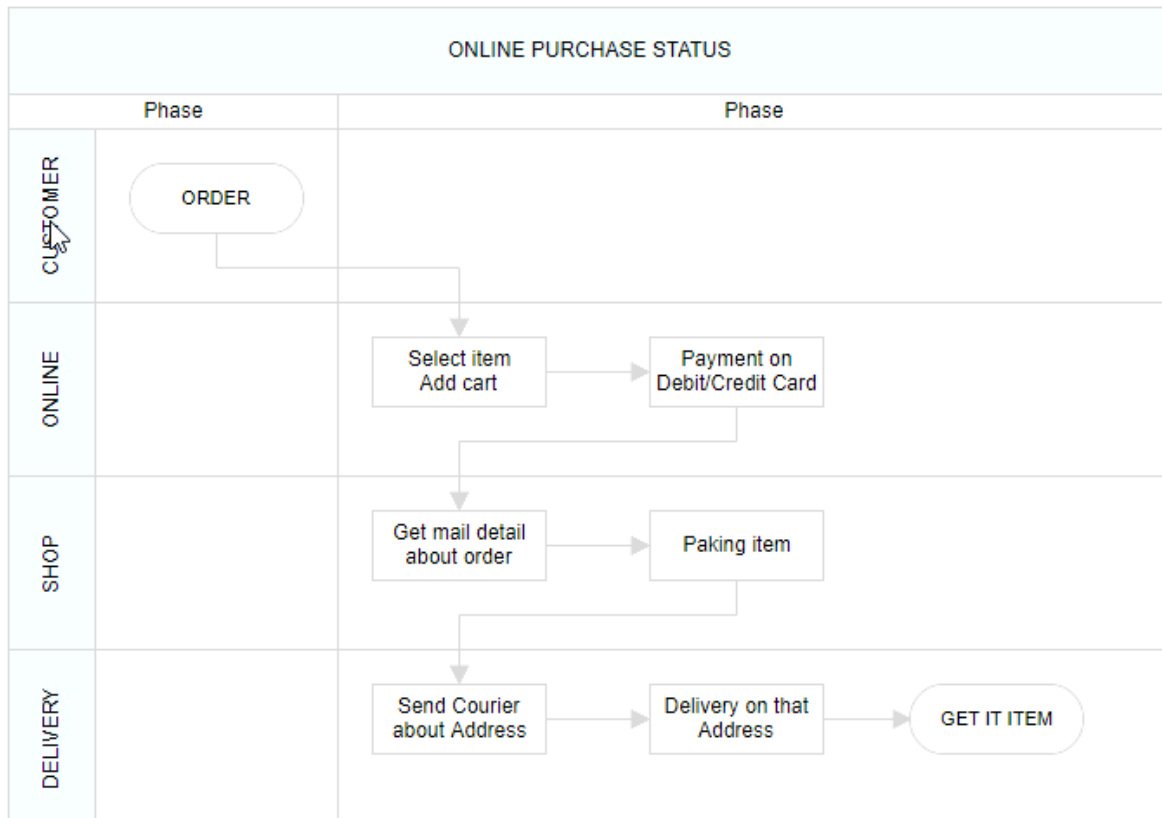
*Lane header editing*

Diagram provides the support to edit Lane headers at runtime. We achieve the header editing by double click event. Double clicking the header label will enables the editing of that.

The following image illustrates how to edit the swimlane header.

The following image illustrates how to edit the lane header.



### Phase

Phase are the subprocess which will split each lane as horizontally or vertically based on the swimlane orientation. The multiple number of [Phase](#) can be added to swimlane.

The following code example illustrates how to add phase at swimlane.

### APP.VUE

```
<template>
  <div id="app">
    <ejs-diagram id="diagram" :width='width' :height='height'
:nodes='nodes' ></ejs-diagram>
  </div>
</template>
<script>
  import Vue from 'vue';
  import { DiagramPlugin } from '@syncfusion/ej2-vue-diagrams';
  Vue.use(DiagramPlugin);
  let nodes = [{
    shape: {
      type: 'SwimLane',
      orientation: 'Horizontal',
      //Intialize header to swimlane
      header: {
        annotation: { content: 'ONLINE PURCHASE STATUS', style: {
fill: '#111111' } },
        height: 50, style: { fontSize: 11 },
      },
    },
  ],
```

```

        lanes: [
          {
            id: 'stackCanvas1',
            height: 100,
            header: {
              annotation: { content: 'CUSTOMER' }, width: 50,
              style: { fontSize: 11 }
            },
            children: [
              {
                id: 'node1',
                annotations: [
                  {
                    content: 'Consumer learns \n of product',
                    style: { fontSize: 11 }
                  }
                ],
                margin: { left: 60, top: 30 },
                height: 40, width: 100,
              }, {
                id: 'node2',
                shape: { type: 'Flow', shape: 'Decision' },
                annotations: [
                  {
                    content: 'Does \n Consumer want \nthe product',
                    style: { fontSize: 11 }
                  }
                ],
                margin: { left: 200, top: 20 },
                height: 60, width: 120,
              },
            ],
          },
        ],
        // Set phase to swimlane
        phases: [
          {
            id: 'phase1', offset: 120,
            header: { annotation: { content: 'Phase' } }
          }, {
            id: 'phase2', offset: 200,
            header: { annotation: { content: 'Phase' } }
          },
        ],
        phaseSize: 20,
      },
      offsetX: 300, offsetY: 200,
      height: 200,
      width: 350
    ]
  }
}
export default {
  name: 'app'
  data() {
    return {
      width: "100%",
      height: "350px",
      nodes: nodes,

```

```

    }
  }
}
</script>
<style>
  @import "../node_modules/@syncfusion/ej2-vue-
diagrams/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/diagram/swimlane/phase-cs1" %}

#### *Dynamically add phase to lane*

You can add the a phase at runtime by using client side API method called **addPhases**. The following code example illustrates how to add phase at run time.

#### **APP.VUE**

```

<template>
  <div id="app">
    <ejs-diagram id="diagram" :width='width' :height='height'
:nodes='nodes' ></ejs-diagram>
  </div>
</template>
<script>
  import Vue from 'vue';
  import { DiagramPlugin } from '@syncfusion/ej2-vue-diagrams';
  Vue.use(DiagramPlugin);
  let nodes = [{
    shape: {
      type: 'SwimLane',
      orientation: 'Horizontal',
      //Intialize header to swimlane
      header: {
        annotation: { content: 'ONLINE PURCHASE STATUS', style:
{ fill: '#111111' } },
        height: 50, style: { fontSize: 11 },
      },
      lanes: [
        {
          id: 'stackCanvas1',
          height: 100,
          // customization of lane header
          header: {
            annotation: { content: 'Online Consumer' }, width:
30,
            style: { fontSize: 11 },style: { fill: 'red' }
          },
        },
      ],
      phases: [
        {
          id: 'phase1', offset: 170,
          header: { annotation: { content: 'Phase' } }
        },
      ],
      phaseSize: 20,
    }
  ]
}

```

```

        },
        offsetX: 300, offsetY: 200,
        height: 200,
        width: 350
    ]}
    export default {
        name: 'app'
        data() {
            return {
                width: "100%",
                height: "350px",
                nodes: nodes,
            }
        }
        mounted: function() {
            let diagramInstance: Diagram;
            let diagramObj: any = document.getElementById("diagram");
            diagramInstance = diagramObj.ej2_instances[0];
            let phase = [{
                id: 'phase3', offset: 220,
                header: { annotation: { content: 'Phase' } }
            }]
            diagramInstance.addPhases(diagramInstance.nodes[0], phase);
        }
    }
</script>
<style>
    @import "../../node_modules/@syncfusion/ej2-vue-
    diagrams/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/diagram/swimlane/addphases-cs1" %}

### Customizing phase

- The length of region can be set by using the [offset](#) property of the phase.
- Every phase region can be textually described with the [header](#) property of the phase
- You can increase the width of phase by using [phaseSize](#) property of swimlane.
- We can provide additional information to the phase by using the [addInfo](#) property of the phase.

The following code example illustrates how to customize the phase in swimlane.

### APP.VUE

```

<template>
    <div id="app">
        <ejs-diagram id="diagram" :width='width' :height='height'
        :nodes='nodes' ></ejs-diagram>
    </div>
</template>
<script>
    import Vue from 'vue';
    import { DiagramPlugin } from '@syncfusion/ej2-vue-diagrams';
    Vue.use(DiagramPlugin);
    let nodes = [{

```

```

    shape: {
      type: 'SwimLane',
      orientation: 'Horizontal',
      //Intialize header to swimlane
      header: {
        annotation: { content: 'ONLINE PURCHASE STATUS', style: {
fill: '#111111' } },
        height: 50, style: { fontSize: 11 },
      },
      lanes: [
        {
          id: 'stackCanvas1',
          height: 100,
          header: {
            annotation: { content: 'CUSTOMER' }, width: 50,
            style: { fontSize: 11 }
          },
          children: [
            {
              id: 'node1',
              annotations: [
                {
                  content: 'Consumer learns \n of product',
                  style: { fontSize: 11 }
                }
              ],
              margin: { left: 60, top: 30 },
              height: 40, width: 100,
            }, {
              id: 'node2',
              shape: { type: 'Flow', shape: 'Decision' },
              annotations: [
                {
                  content: 'Does \n Consumer want \nthe product',
                  style: { fontSize: 11 }
                }
              ],
              margin: { left: 200, top: 20 },
              height: 60, width: 120,
            },
          ],
        },
      ],
      phases: [
        {
          id: 'phase1', offset: 120,
          // set the phase info
          addInfo:{name:'phase1'},
          header: { annotation: { content: 'Phase' }
},style:{fill:'red'}
        },{
          id: 'phase2', offset: 200,
          header: { annotation: { content: 'Phase' } }
        },
      ],
      phaseSize: 20,
    },

```

```

        offsetX: 300, offsetY: 200,
        height: 200,
        width: 350
    }]
    export default {
        name: 'app'
        data() {
            return {
                width: "100%",
                height: "350px",
                nodes: nodes,
            }
        }
    }
</script>
<style>
    @import "../../node_modules/@syncfusion/ej2-vue-
    diagrams/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/diagram/swimlane/phasecustomize-cs1" %}

#### *Phase interaction*

##### *Resizing*

- The phase can be resized by using its selector.
- You must select the phase header to enable the phase selection.
- Once the phase can be resized, the lane size will be updated automatically.

##### *Resizing helper*

- The special resize selector will be used to resize the phase.
- The resize cursor will be available on the left and bottom direction for horizontal, and the top and bottom direction for vertical swimlane.

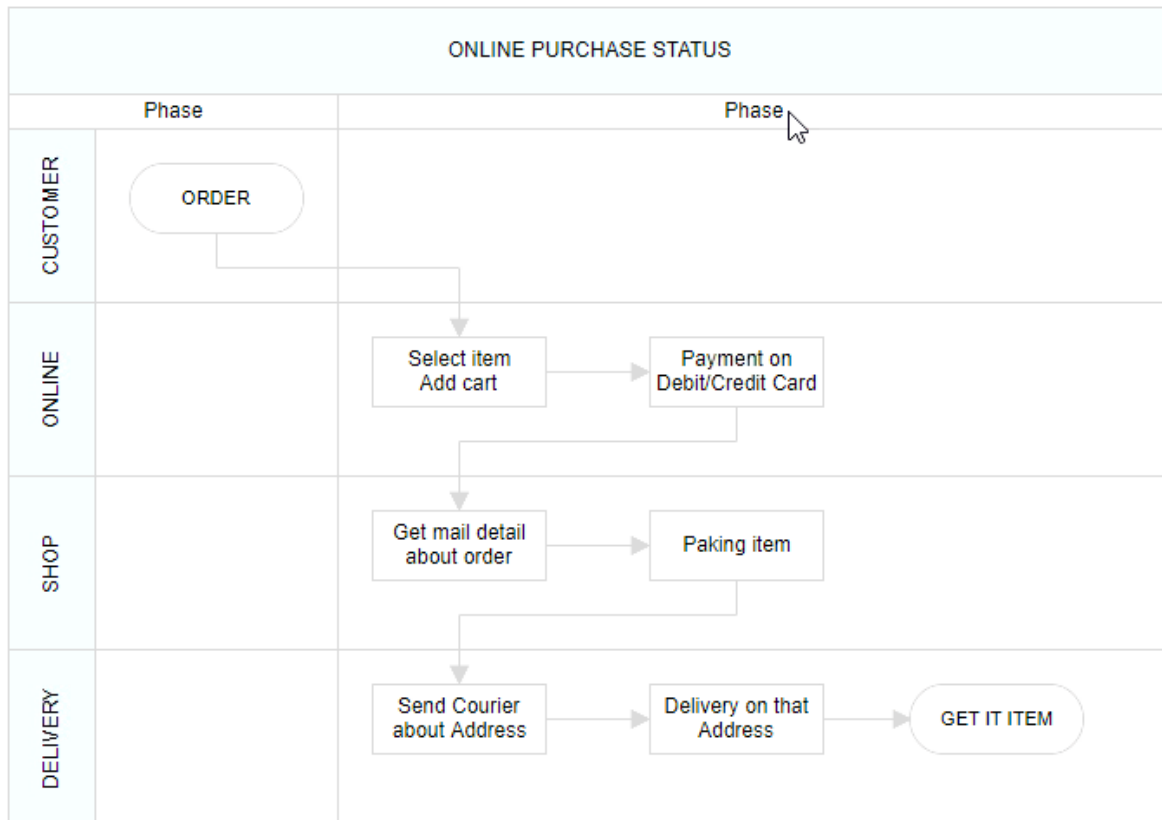
##### *Phase header editing*

Diagram provides the support to edit phase headers at runtime. We achieve the header editing by double click event. Double clicking the header label will enables the editing of that.

The following image illustrates how to edit the swimlane header.



The following image illustrates how to edit the phase header.



### Add swimlane to palette

Diagram provides support to add swimlane and phases to symbol palette. The following code sample illustrate how to add swimlane and phases to palette.

The following code example illustrates how to customize the phase in swimlane.

### APP.VUE

```

<template>
  <div id="app">
    <ejs-symbolpalette id="symbolpalette" :expandMode='expandMode'
:palettes='palettes'
    :getSymbolInfo='getSymbolInfo' :symbolMargin='symbolMargin'
:width='palettewidth' :getNodeDefaults='palettegetNodeDefaults'
:symbolPreview='symbolPreview' :height='paletteheight'
:symbolHeight='symbolHeight'
    :symbolWidth='symbolWidth'></ejs-
symbolpalette>
  </div>
</template>
<script>
  import Vue from 'vue';
  import { DiagramPlugin,
Diagram,
NodeModel,
UndoRedo,
ConnectorModel,

```

```

Node,
Connector,
SymbolPalette, SymbolPalettePlugin,
SymbolInfo,
ShapeModel} from '@syncfusion/ej2-vue-diagrams';
import { ExpandMode } from "@syncfusion/ej2-vue-navigations";
Vue.use(DiagramPlugin);
Vue.use(SymbolPalettePlugin);
let swimlaneShapes : NodeModel[] = [
    {
        id: 'stackCanvas1',
        shape: {
            type: 'SwimLane', lanes: [
                {
                    id: 'lane1',
                    style: { strokeColor: 'black' }, height: 60,
width: 150,
                    header: { width: 50, height: 50, style: {
strokeColor: 'black', fontSize: 11 } },
                }
            ],
            orientation: 'Horizontal', isLane: true
        },
        height: 60,
        width: 140,
        offsetX: 70,
        offsetY: 30,
    }, {
        id: 'stackCanvas2',
        shape: {
            type: 'SwimLane',
            lanes: [
                {
                    id: 'lane1',
                    style: { strokeColor: 'black' }, height: 150,
width: 60,
                    header: { width: 50, height: 50, style: {
strokeColor: 'black', fontSize: 11 } },
                }
            ],
            orientation: 'Vertical', isLane: true
        },
        height: 140,
        width: 60,
        // style: { fill: '#f5f5f5' },
        offsetX: 70,
        offsetY: 30,
    }, {
        id: 'verticalPhase',
        shape: {
            type: 'SwimLane',
            phases: [{ style: { strokeWidth: 1, strokeDashArray:
'3,3', strokeColor: '#A9A9A9' }, }],
            annotations: [{ text: '' }],
            orientation: 'Vertical', isPhase: true
        },
        height: 60,

```

```

        width: 140
      }, {
        id: 'horizontalPhase',
        shape: {
          type: 'SwimLane',
          phases: [{ style: { strokeWidth: 1, strokeDashArray:
'3,3', strokeColor: '#A9A9A9' }, }],
          annotations: [{ text: '' }],
          orientation: 'Horizontal', isPhase: true
        },
        height: 60,
        width: 140
      }
    ];
    let palette: any;
    let size: any;
    let expand: any;
    export default {
      name: 'app'
      data() {
        return {
          //Defines how many palettes can be at expanded mode at a time
          expandMode: "Multiple",
          //Defines the palette collection
          palettes: [{
            id: 'swimlane',
            expanded: true,
            symbols: swimlaneShapes,
            title: 'Swimlane Shapes',
            iconCss: 'e-ddb-icons e-basic'
          }],
          symbolMargin: {
            left: 15,
            right: 15,
            top: 15,
            bottom: 15
          },
          palettegetNodeDefaults: (node: NodeModel): void => {
            node.width = 100;
            node.height = 100;
            node.style.strokeColor = '#3A3A3A';
          },
          getSymbolInfo: (symbol: NodeModel): SymbolInfo => {
            return {
              fit: true
            };
          },
          symbolPreview: {
            height: 100,
            width: 100,
            offset: {
              x: 0.5,
              y: 0.5
            }
          },
          palettewidth: "100%",
          paletteheight: "700px",

```

```

        symbolHeight: 60,
        symbolWidth: 60,
    };
}
}
</script>
<style>
    @import "../node_modules/@syncfusion/ej2-vue-
    diagrams/styles/material.css";
</style>

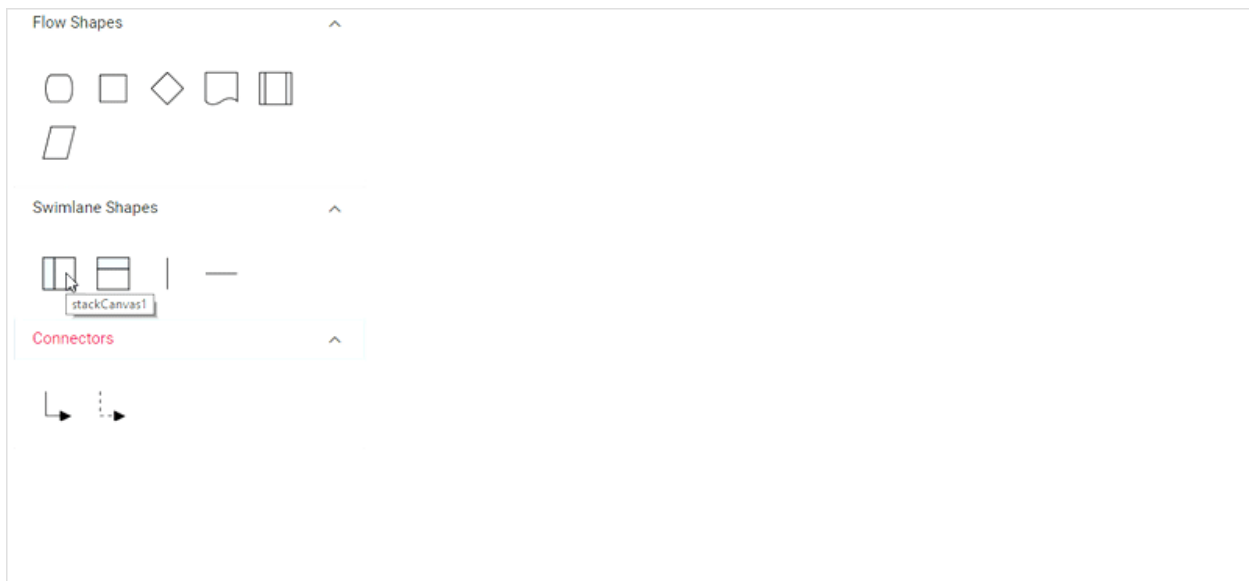
```

{% previewsample "page.domainurl/code-snippet/diagram/swimlane/palette-cs1" %}

[Drag and drop swimlane to palette](#)

- The drag and drop support for swimlane shapes has been provided.
- When you drag and drop the lane shape, if the diagram already contains swimlane with the same orientation, the lane will be added and stacked inside a swimlane based on the order. Otherwise, it will be added a new swimlane.
- The phase will only drop on swimlane shape with same orientation.

The following image illustrates how to drag symbol from palette.



### Limitations

- Connectors cannot be canceled when added directly to swimlane. You must initialize the connector through connector collection.
- We cannot edit the phase line style.

### Labels in Vue Diagram component

[Annotation](#) is a block of text that can be displayed over a node or connector. Annotation is used to textually represent an object with a string that can be edited at runtime. Multiple annotations can be added to a node/connector.

<!-- markdownlint-disable MD033 -->

### Create annotation

An annotation can be added to a node/connector by defining the annotation object and adding that to the annotation collection of the node/connector. The [content](#) property of annotation defines the text to be displayed. The following code illustrates how to create a annotation.

### APP.VUE

```
<template>
  <div id="app">
    <ejs-diagram id="diagram" :width='width' :height='height'
    :nodes='nodes' :connectors='connectors' ></ejs-diagram>
  </div>
</template>
<script>
  import Vue from 'vue';
  import { DiagramPlugin } from '@syncfusion/ej2-vue-diagrams';
  Vue.use(DiagramPlugin);
  let nodes = [{
    // Position of the node
    offsetX: 100,
    offsetY: 100,
    // Size of the node
    width: 100,
    height: 100,
    style: {
      fill: '#6BA5D7',
      strokeColor: 'white'
    },
    // Sets the Annotation for the Node
    annotations: [{
      // Sets the text to be displayed
      content: 'Annotation'
    }]
  }];
  let connectors = [{
    sourcePoint: {
      x: 300,
      y: 100
    },
    targetPoint: {
      x: 400,
      y: 300
    },
    type: 'Orthogonal',
    style: {
      strokeColor: '#6BA5D7'
    },
    // Sets the Annotation for the Connector
    annotations: [{
      // Sets the text to be displayed
      content: 'Annotation'
    }]
  }]
  export default {
    name: 'app'
  }
</script>
```

```

        data() {
            return {
                width: "100%",
                height: "350px",
                nodes: nodes,
                connectors: connectors
            }
        }
    }
</script>
<style>
    @import "../node_modules/@syncfusion/ej2-vue-
    diagrams/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/diagram/labels/Annotation-cs1" %}

#### Add annotations at runtime

- Annotations can be added at runtime by using the client-side method [addLabels](#). The following code illustrates how to add a annotation to a node.
- The annotation's [ID](#) property is used to define the name of the annotation and its further used to find the annotation at runtime and do any customization.

#### APP.VUE

```

<template>
    <div id="app">
        <ejs-diagram id="diagram" :width='width' :height='height'
        :nodes='nodes' ></ejs-diagram>
    </div>
</template>
<script>
    import Vue from 'vue';
    import { DiagramPlugin ,ShapeAnnotationModel} from '@syncfusion/ej2-vue-
    diagrams';
    Vue.use(DiagramPlugin);
    let nodes = [{
        // Position of the node
        offsetX: 250,
        offsetY: 250,
        // Size of the node
        width: 100,
        height: 100,
        style: {
            fill: '#6BA5D7',
            strokeColor: 'white'
        },
    },
    ]
    export default {
        name: 'app'
        data() {
            return {
                width: "100%",
                height: "350px",
            }
        }
    }

```

```

        nodes: nodes,
      }
    }
    mounted: function() {
      let diagramInstance: Diagram;
      let diagramObj: any = document.getElementById("diagram");
      diagramInstance = diagramObj.ej2_instances[0];
      let annotation: ShapeAnnotationModel[] = [{
        id: 'label1',
        content: 'Annotation'
      }]
      //Method to add labels at run time
      diagramInstance.addLabels(diagramInstance.nodes[0], annotation);
      diagramInstance.dataBind();
    }
  }
</script>
<style>
  @import "../node_modules/@syncfusion/ej2-vue-
  diagrams/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/diagram/labels/Run-cs1" %}

### Remove annotation

A collection of annotations can be removed from the node by using client-side method [removeLabels](#). The following code illustrates how to remove a annotation to a node.

### APP.VUE

```

<template>
  <div id="app">
    <ejs-diagram id="diagram" :width='width' :height='height'
    :nodes='nodes' ></ejs-diagram>
  </div>
</template>
<script>
  import Vue from 'vue';
  import { DiagramPlugin, shapeAnnotationModel } from '@syncfusion/ej2-vue-
  diagrams';
  Vue.use(DiagramPlugin);
  let nodes = [{
    // Position of the node
    offsetX: 250,
    offsetY: 250,
    // Size of the node
    width: 100,
    height: 100,
    style: {
      fill: '#6BA5D7',
      strokeColor: 'white'
    },
    // Sets the annotation for the node
    annotations: [{
      id: 'label1',
      // Sets the text to be displayed

```

```

        content: 'Annotation'
      }]
    }]
    export default {
      name: 'app'
      data() {
        return {
          width: "100%",
          height: "350px",
          nodes: nodes,
        }
      }
      mounted: function() {
        let diagramInstance: Diagram;
        let diagramObj: any = document.getElementById("diagram");
        diagramInstance = diagramObj.ej2_instances[0];
        let annotation: shapeAnnotationModel[] = [{
          id: 'label1',
          content: 'Annotation'
        }];
        diagramInstance.removeLabels(diagramInstance.nodes[0],
        annotation);
      }
    }
  </script>
  <style>
    @import "../../node_modules/@syncfusion/ej2-vue-
    diagrams/styles/material.css";
  </style>

```

{% previewsample "page.domainurl/code-snippet/diagram/labels/Update-cs1" %}

### Update annotation at runtime

You can change any annotation properties at runtime and update it through the client-side method `dataBind`.

The following code example illustrates how to change the annotation properties.

### APP.VUE

```

<template>
  <div id="app">
    <ejs-diagram id="diagram" :width='width' :height='height'
    :nodes='nodes' ></ejs-diagram>
  </div>
</template>
<script>
  import Vue from 'vue';
  import { DiagramPlugin } from '@syncfusion/ej2-vue-diagrams';
  Vue.use(DiagramPlugin);
  let nodes = [{
    // Position of the node
    offsetX: 250,
    offsetY: 250,
    // Size of the node
    width: 100,

```



```

        height: 100,
        style: {
            fill: '#6BA5D7',
            strokeColor: 'white'
        },
        // Sets the annotation for the node
        annotations: [{
            content: 'Annotation'
        }]
    }]
    export default {
        name: 'app'
        data() {
            return {
                width: "100%",
                height: "350px",
                nodes: nodes,
            }
        }
        mounted: function() {
            let diagramInstance: Diagram;
            let diagramObj: any = document.getElementById("diagram");
            diagramInstance = diagramObj.ej2_instances[0];
            // Adds to the Diagram
            diagramInstance.nodes[0].annotations[0].content = 'Updated
Annotation';
            //Method to update the annotation at run time
            diagramInstance.dataBind();
        }
    }
</script>
<style>
    @import "../node_modules/@syncfusion/ej2-vue-
diagrams/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/diagram/labels/Update-cs2" %}

### Alignment

Annotation can be aligned relative to the node boundaries. It has [margin](#), [offset](#), horizontal, and vertical alignment settings. It is quite tricky when all four alignments are used together but gives more control over alignment.

### Offset

The offset property of annotation is used to align the annotations based on fractions. 0 represents top/left corner, 1 represents bottom/right corner, and 0.5 represents half of width/height.

Set the size for a nodes annotation by using [width](#) and [height](#) properties.

The following code shows the relationship between the annotation position (black color circle) and offset (fraction values).

### APP.VUE

```

<template>
  <div id="app">

```

```

    <ejs-diagram id="diagram" :width='width' :height='height'
:nodes='nodes' ></ejs-diagram>
  </div>
</template>
<script>
  import Vue from 'vue';
  import { DiagramPlugin } from '@syncfusion/ej2-vue-diagrams';
  Vue.use(DiagramPlugin);
  let nodes = [{
    // Position of the node
    offsetX: 250,
    offsetY: 250,
    // Size of the node
    width: 100,
    height: 100,
    style: {
      fill: '#6BA5D7',
      strokeColor: 'white'
    },
    // Sets the annotation for the node
    annotations: [{
      // Sets the content for the annotation
      content: 'Annotation',
      //Sets the offset for the content
      offset: {
        x: 0,
        y: 1
      }
    }]
  }]
  export default {
    name: 'app'
    data() {
      return {
        width: "100%",
        height: "350px",
        nodes: nodes,
      }
    }
  }
</script>
<style>
  @import "../../node_modules/@syncfusion/ej2-vue-
diagrams/styles/material.css";
</style>

```

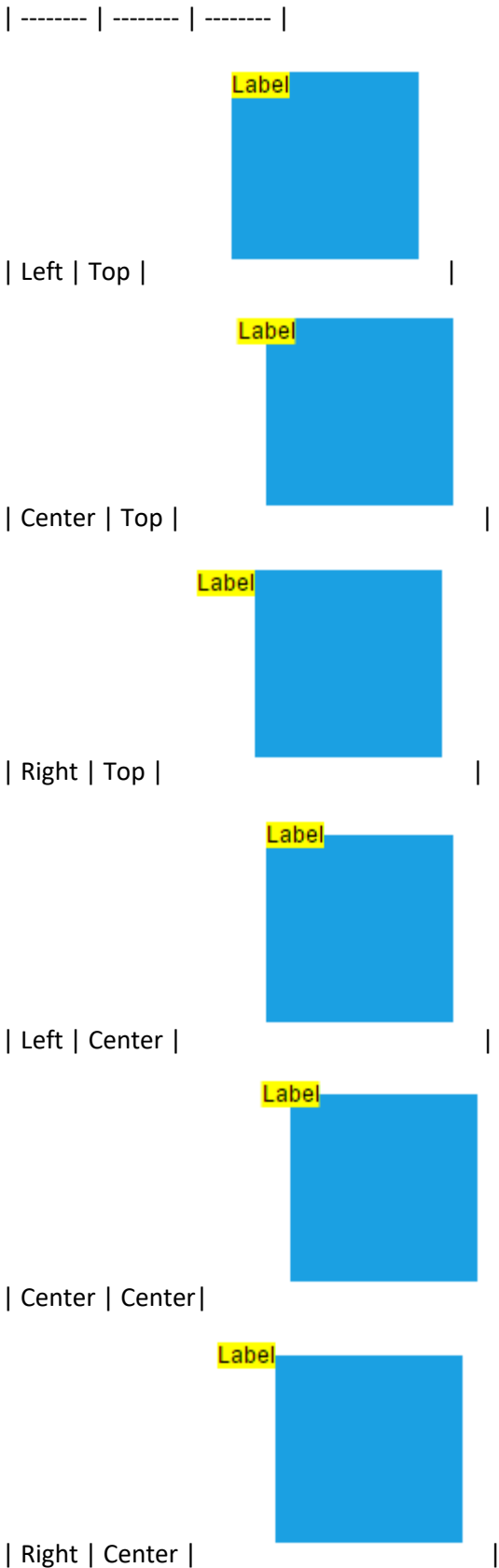
{% previewsample "page.domainurl/code-snippet/diagram/labels/Update-cs3" %}

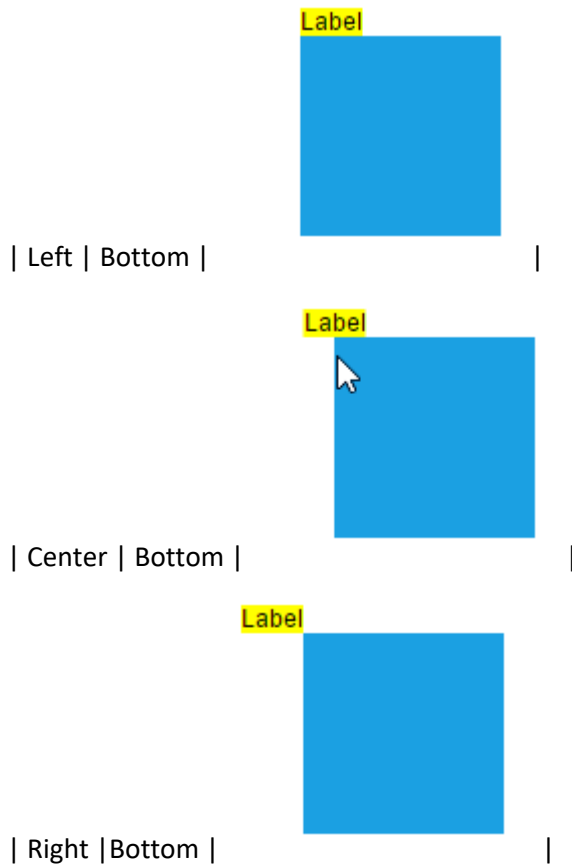
### Horizontal and vertical alignment

The [horizontalAlignment](#) property of annotation is used to set how the annotation is horizontally aligned at the annotation position determined from the fraction values. The [verticalAlignment](#) property is used to set how annotation is vertically aligned at the annotation position.

The following tables illustrates all the possible alignments visually with 'offset (0, 0)'.

| Horizontal Alignment | Vertical Alignment | Output with Offset(0,0) |





The following codes illustrates how to align annotations.

#### **APP.VUE**

```
<template>
  <div id="app">
    <ejs-diagram id="diagram" :width='width' :height='height'
:nodes='nodes' ></ejs-diagram>
  </div>
</template>
<script>
  import Vue from 'vue';
  import { DiagramPlugin } from '@syncfusion/ej2-vue-diagrams';
  Vue.use(DiagramPlugin);
  let nodes = [{
    // Position of the node
    offsetX: 250,
    offsetY: 250,
    // Size of the node
    width: 100,
    height: 100,
    style: {
      fill: '#6BA5D7',
      strokeColor: 'white'
    },
    // Sets the annotation for the node
    annotations: [{
      content: 'Annotation',
```

```

        // Sets the horizontal alignment as left
        horizontalAlignment: 'Left',
        // Sets the vertical alignment as Center
        verticalAlignment: 'Center'
    }]
  }]
  export default {
    name: 'app'
    data() {
      return {
        width: "100%",
        height: "350px",
        nodes: nodes,
      }
    }
  }
</script>
<style>
  @import "../node_modules/@syncfusion/ej2-vue-
  diagrams/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/diagram/labels/Alignment-cs1" %}

### Annotation alignment with respect to segments

The offset and alignment properties of annotation allows you to align the connector annotations with respect to the segments.

The following code example illustrates how to align connector annotations.

### APP.VUE

```

<template>
  <div id="app">
    <ejs-diagram id="diagram" :width='width' :height='height'
    :nodes='nodes' :connectors='connectors'></ejs-diagram>
  </div>
</template>
<script>
  import Vue from 'vue';
  import { DiagramPlugin } from '@syncfusion/ej2-vue-diagrams';
  Vue.use(DiagramPlugin);
  let nodes = [{
    id: 'node1',
    // Position of the node
    offsetX: 100,
    offsetY: 100,
    // Size of the node
    width: 100,
    height: 100,
    style: {
      fill: '#6BA5D7',
      strokeColor: 'white'
    },
    // Sets the annotation for the node
    annotations: [{

```

```

        content: 'Task1'
      ]]
    },
    {
      id: 'node2',
      // Position of the node
      offsetX: 300,
      offsetY: 100,
      // Size of the node
      width: 100,
      height: 100,
      style: {
        fill: '#6BA5D7',
        strokeColor: 'white'
      },
      // Sets the annotation for the node
      annotations: [{
        content: 'Task2'
      }]
    }
  ]];
  let connectors = [{
    sourceID: 'node1',
    targetID: 'node2',
    type: 'Orthogonal',
    style: {
      strokeColor: '#6BA5D7',
      strokeWidth: 2
    },
    // Sets the annotation for the connector
    annotations: [{
      content: '0',
      // Sets the offset for the content
      offset: 0
    }, {
      content: '1',
      // Sets the offset for the content
      offset: 1
    }]
  }]
  export default {
    name: 'app'
    data() {
      return {
        width: "100%",
        height: "350px",
        nodes: nodes,
        connectors: connectors
      }
    }
  }
</script>
<style>
  @import "../node_modules/@syncfusion/ej2-vue-
  diagrams/styles/material.css";
</style>

```

```
{% previewsample "page.domainurl/code-snippet/diagram/labels/Segment-cs1" %}
```

### Margin

[Margin](#) is an absolute value used to add some blank space in any one of its four sides. The annotations can be displaced with the margin property.

The following code example illustrates how to align a annotation based on its `offset`, `horizontalAlignment`, `verticalAlignment`, and `margin` values.

### APP.VUE

```
<template>
  <div id="app">
    <ejs-diagram id="diagram" :width='width' :height='height'
:nodes='nodes' ></ejs-diagram>
  </div>
</template>
<script>
  import Vue from 'vue';
  import { DiagramPlugin } from '@syncfusion/ej2-vue-diagrams';
  Vue.use(DiagramPlugin);
  let nodes = [{
    id: 'node1',
    // Position of the node
    offsetX: 100,
    offsetY: 100,
    // Size of the node
    width: 100,
    height: 100,
    style: {
      fill: '#6BA5D7',
      strokeColor: 'white'
    },
    // Sets the annotation for the connector
    annotations: [{
      content: 'Task1',
      // Sets the margin for the content
      margin: {
        top: 10
      },
      horizontalAlignment: 'Center',
      verticalAlignment: 'Top',
      offset: {
        x: 0.5,
        y: 1
      }
    }
  ]
}]
export default {
  name: 'app'
  data() {
    return {
      width: "100%",
      height: "350px",
      nodes: nodes,
    }
  }
}
```

```

    }
  </script>
  <style>
    @import ".../node_modules/@syncfusion/ej2-vue-
    diagrams/styles/material.css";
  </style>

```

{% previewsample "page.domainurl/code-snippet/diagram/labels/Margin-cs1" %}

### Text align

The [textAlign](#) property of annotation allows you to set how the text should be aligned (left, right, center, or justify) inside the text block. The following codes illustrate how to set `textAlign` for an annotation.

### APP.VUE

```

<template>
  <div id="app">
    <ejs-diagram id="diagram" :width='width' :height='height'
    :nodes='nodes' ></ejs-diagram>
  </div>
</template>
<script>
  import Vue from 'vue';
  import { DiagramPlugin } from '@syncfusion/ej2-vue-diagrams';
  Vue.use(DiagramPlugin);
  let nodes = [{
    id: 'node1',
    // Position of the node
    offsetX: 100,
    offsetY: 100,
    // Size of the node
    width: 100,
    height: 100,
    style: {
      fill: '#6BA5D7',
      strokeColor: 'white'
    },
    // Sets the annotation for the NOde
    annotations: [{
      content: 'Text align is set as Left',
      // Sets the textAlign as left for the content
      style: {
        textAlign: 'Left'
      }
    }]
  }]
  export default {
    name: 'app'
    data() {
      return {
        width: "100%",
        height: "350px",
        nodes: nodes,
      }
    }
  }

```



```

</script>
<style>
  @import "../node_modules/@syncfusion/ej2-vue-
  diagrams/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/diagram/labels/TextAlign-cs1" %}

### Hyperlink

Diagram provides a support to add a [hyperlink](#) for the nodes/connectors annotation. It can also be customized.

A User can open the hyperlink in the new window, the same tab and the new tab by using the [hyperlinkOpenState](#) property

### APP.VUE

```

<template>
  <div id="app">
    <ejs-diagram id="diagram" :width='width' :height='height'
    :nodes='nodes' ></ejs-diagram>
  </div>
</template>
<script>
  import Vue from 'vue';
  import { DiagramPlugin } from '@syncfusion/ej2-vue-diagrams';
  Vue.use(DiagramPlugin);
  let nodes = [{
    id: 'node1',
    // Position of the node
    offsetX: 100,
    offsetY: 100,
    // Size of the node
    width: 100,
    height: 100,
    style: {
      fill: '#6BA5D7',
      strokeColor: 'white'
    },
    // Sets the annotation for the Node
    annotations: [{
      hyperlink: {
        link: 'https://hr.syncfusion.com/home',
        //Set the link to open in the current tab
        hyperlinkOpenState: 'CurrentTab'
      }
    }]
  }]
  export default {
    name: 'app'
    data() {
      return {
        width: "100%",
        height: "350px",
        nodes: nodes,
      }
    }
  }

```

```

    }
  }
</script>
<style>
  @import "../node_modules/@syncfusion/ej2-vue-
  diagrams/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/diagram/labels/Link-cs1" %}

### Template Support for Annotation

Diagram provides template support for annotation. you should define a SVG/HTML content as string in the annotation's [template](#) property.

The following code illustrates how to define a template in node's annotation. similarly, you can define it in connectors.

### APP.VUE

```

<template>
  <div id="app">
    <ejs-diagram id="diagram" :width='width':height='height'
    :nodes='nodes' :connectors='connectors' ></ejs-diagram>
  </div>
</template>
<script>
  import Vue from 'vue';
  import { DiagramPlugin } from '@syncfusion/ej2-vue-diagrams';
  Vue.use(DiagramPlugin);
  let nodes = [{
    id: 'node1',
    // Position of the node
    offsetX: 100,
    offsetY: 100,
    // Size of the node
    width: 100,
    height: 100,
    style: {
      fill: '#6BA5D7',
      strokeColor: 'white'
    },
    // Sets the annotation for the Node
    annotations: [{
      // Set an template for annotation
      template: '<div><input type="button" value="Submit"></div>'
    }]
  }]
  let connectors = [{
    sourcePoint: {
      x: 300,
      y: 100
    },
    targetPoint: {
      x: 400,
      y: 300
    },
  ],

```

```

        type: 'Orthogonal',
        style: {
            strokeColor: '#6BA5D7'
        },
        // Sets the Annotation for the Connector
        annotations: [{
            // Set an template for annotation
            height: 60, width: 100, offset: 0.5,
            template: '<div><input type="button" value="Submit"></div>'
        }]
    }];
    export default {
        name: 'app'
        data() {
            return {
                width: "100%",
                height: "350px",
                nodes: nodes,
                connectors: connectors
            }
        }
    }
</script>
<style>
    @import "../node_modules/@syncfusion/ej2-vue-
    diagrams/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/diagram/labels/labeltemplate-cs1" %}

### Wrapping

When text overflows node boundaries, you can control it by using [text wrapping](#). So, it is wrapped into multiple lines. The wrapping property of annotation defines how the text should be wrapped. The following code illustrates how to wrap a text in a node.

### APP.VUE

```

<template>
    <div id="app">
        <ejs-diagram id="diagram" :width='width' :height='height'
        :nodes='nodes' ></ejs-diagram>
    </div>
</template>
<script>
    import Vue from 'vue';
    import { DiagramPlugin } from '@syncfusion/ej2-vue-diagrams';
    Vue.use(DiagramPlugin);
    let nodes = [{
        id: 'node1',
        // Position of the node
        offsetX: 100,
        offsetY: 100,
        // Size of the node
        width: 100,
        height: 100,
        style: {

```

```

        fill: '#6BA5D7',
        strokeColor: 'white'
    },
    //Sets the annotation for the node
    annotations: [{
        content: 'Annotation Text Wrapping',
        // Sets the style for the text wrapping
        style: {
            textWrapping: 'Wrap'
        }
    }]
  ]
}
export default {
  name: 'app'
  data() {
    return {
      width: "100%",
      height: "350px",
      nodes: nodes,
    }
  }
}
</script>
<style>
  @import "../node_modules/@syncfusion/ej2-vue-
  diagrams/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/diagram/labels/Wrap-cs1" %}

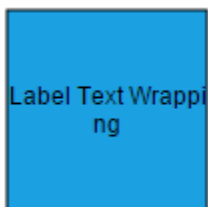
| Value | Description | Image |

| ----- | ----- | ----- |



| No Wrap | Text will not be wrapped. |

| Wrap | Text-wrapping occurs, when the text overflows beyond the available node width. |



| WrapWithOverflow (Default) | Text-wrapping occurs, when the text overflows beyond the available node width. However, the text may overflow beyond the node width in the case of a very long word. |



### Text overflow

The label's [TextOverflow](#) property is used control whether to display the overflowed content in node or not.

### APP.VUE

```
<template>
  <div id="app">
    <ejs-diagram id="diagram" :width='width' :height='height'
:nodes='nodes' ></ejs-diagram>
  </div>
</template>
<script>
  import Vue from 'vue';
  import { DiagramPlugin } from '@syncfusion/ej2-vue-diagrams';
  Vue.use(DiagramPlugin);
  let nodes = [{
    id: 'node1',
    // Position of the node
    offsetX: 100,
    offsetY: 100,
    // Size of the node
    width: 100,
    height: 100,
    style: {
      fill: '#6BA5D7',
      strokeColor: 'white'
    },
    // Sets the annotation for the node
    annotations: [{
      content: 'Annotation Text',
      // Sets the style for the text to be displayed
      style: {
        textOverflow: 'Ellipsis'
      }
    }]
  }]
  export default {
    name: 'app'
    data() {
      return {
        width: "100%",
        height: "350px",
        nodes: nodes,
      }
    }
  }
</script>
```

```

    }
  }
</script>
<style>
  @import "../../node_modules/@syncfusion/ej2-vue-
  diagrams/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/diagram/labels/Overflow-cs1" %}

### Appearance

- You can change the font style of the annotations with the font specific properties (fontSize, fontFamily, color). The following code illustrates how to customize the appearance of the annotation.
- The label's [bold](#), [italic](#), and [textDecoration](#) properties are used to style the label's text.
- The label's [fill](#), [strokeColor](#), and [strokeWidth](#) properties are used to define the background color and border color of the annotation and the [opacity](#) property is used to define the transparency of the annotations.
- The [visible](#) property of the annotation enables or disables the visibility of annotation.

### APP.VUE

```

<template>
  <div id="app">
    <ejs-diagram id="diagram" :width='width' :height='height'
    :nodes='nodes' ></ejs-diagram>
  </div>
</template>
<script>
  import Vue from 'vue';
  import { DiagramPlugin } from '@syncfusion/ej2-vue-diagrams';
  Vue.use(DiagramPlugin);
  let nodes = [{
    id: 'node1',
    // Position of the node
    offsetX: 100,
    offsetY: 100,
    // Size of the node
    width: 100,
    height: 100,
    style: {
      fill: '#6BA5D7',
      strokeColor: 'white'
    },
    // Sets the annotation for the node
    annotations: [{
      content: 'Annotation Text',
      // Sets the style for the text to be displayed
      style: {
        color: 'black',
        bold: true,
        italic: true,
        fontSize: '12',

```

```

        fontFamily: 'TimesNewRoman'
    }
  ]]
  ]]
  export default {
    name: 'app'
    data() {
      return {
        width: "100%",
        height: "350px",
        nodes: nodes,
      }
    }
  }
</script>
<style>
  @import "../../node_modules/@syncfusion/ej2-vue-
  diagrams/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/diagram/labels/Appear-cs1" %}

The fill, border, and opacity appearances of the text can also be customized with appearance specific properties of annotation. The following code illustrates how to customize background, opacity, and border of the annotation.

#### APP.VUE

```

<template>
  <div id="app">
    <ejs-diagram id="diagram" :width='width' :height='height'
    :nodes='nodes' ></ejs-diagram>
  </div>
</template>
<script>
  import Vue from 'vue';
  import { DiagramPlugin } from '@syncfusion/ej2-vue-diagrams';
  Vue.use(DiagramPlugin);
  let nodes = [{
    id: 'node1',
    // Position of the node
    offsetX: 100,
    offsetY: 100,
    // Size of the node
    width: 100,
    height: 100,
    style: {
      fill: '#6BA5D7',
      strokeColor: 'white'
    },
    // Sets the annotation for the node
    annotations: [{
      content: 'Annotation Text',
      style: {
        color: 'black',
        fill: 'white',

```

```

        opacity: 0.7,
        strokeColor: 'black',
        strokeWidth: 2
      }
    ]
  ]
}
export default {
  name: 'app'
  data() {
    return {
      width: "100%",
      height: "350px",
      nodes: nodes,
    }
  }
}
</script>
<style>
  @import "../node_modules/@syncfusion/ej2-vue-
  diagrams/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/diagram/labels/Opacity-cs1" %}

### Interaction

Diagram allows annotation to be interacted by selecting, dragging, rotating, and resizing. Annotation interaction is disabled, by default. You can enable annotation interaction with the [constraints](#) property of annotation. You can also curtail the services of interaction by enabling either selecting, dragging, rotating, or resizing individually with the respective constraints property of annotation. The following code illustrates how to enable annotation interaction.

### APP.VUE

```

<template>
  <div id="app">
    <ejs-diagram id="diagram" :width='width' :height='height'
    :nodes='nodes' ></ejs-diagram>
  </div>
</template>
<script>
  import Vue from 'vue';
  import { DiagramPlugin, AnnotationConstraints } from '@syncfusion/ej2-
  vue-diagrams';
  Vue.use(DiagramPlugin);
  let nodes = [{
    id: 'node1',
    // Position of the node
    offsetX: 100,
    offsetY: 100,
    // Size of the node
    width: 100,
    height: 100,
    style: {
      fill: '#6BA5D7',
      strokeColor: 'white'
    }
  },

```



```

// Sets the annotation for the node
annotations: [{
  content: 'Annotation Text',
  //Sets the constraints as Interaction
  constraints: AnnotationConstraints.Interaction
}]
}]
export default {
  name: 'app'
  data() {
    return {
      width: "100%",
      height: "350px",
      nodes: nodes,
    }
  }
}
</script>
<style>
  @import "../node_modules/@syncfusion/ej2-vue-
  diagrams/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/diagram/labels/Interaction-cs1" %}

### Edit

Diagram provides support to edit an annotation at runtime, either programmatically or interactively. By default, annotation is in view mode. But it can be brought to edit mode in two ways;

- Programmatically

By using [startTextEdit](#) method, edit the text through programmatically.

- Interactively
  1. By double-clicking the annotation.
  2. By selecting the item and pressing the F2 key.

Double-clicking any annotation will enables editing and the node enables first annotation editing. When the focus of editor is lost, the annotation for the node is updated.

When you double-click on the node/connector/diagram model, the [doubleClick](#) event gets triggered.

### Read-only annotations

Diagram allows to create read-only annotations. You have to set the read-only property of annotation to enable/disable the read-only [constraints](#). The following code illustrates how to enable read-only mode.

### APP.VUE

```

<template>
  <div id="app">
    <ejs-diagram id="diagram" :width='width' :height='height'
    :nodes='nodes' ></ejs-diagram>
  </div>
</template>

```

```

<script>
  import Vue from 'vue';
  import { DiagramPlugin, AnnotationConstraints } from '@syncfusion/ej2-vue-diagrams';
  Vue.use(DiagramPlugin);
  let nodes = [{
    id: 'node1',
    // Position of the node
    offsetX: 100,
    offsetY: 100,
    // Size of the node
    width: 100,
    height: 100,
    style: {
      fill: '#6BA5D7',
      strokeColor: 'white'
    },
    // Sets the annotation for the node
    annotations: [{
      content: 'Annotation Text',
      //Sets the constraints as Read only
      constraints: AnnotationConstraints.ReadOnly
    }]
  }]
  export default {
    name: 'app'
    data() {
      return {
        width: "100%",
        height: "350px",
        nodes: nodes,
      }
    }
  }
</script>
<style>
  @import "../node_modules/@syncfusion/ej2-vue-diagrams/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/diagram/labels/Read-cs1" %}

### Drag Limit

- The diagram control now supports defining the [dragLimit](#) to the label while dragging from the connector and also update the position to the nearest segment offset.
- You can set the value to dragLimit [left](#), [right](#), [top](#), and [bottom](#) properties which allow the dragging of connector labels to a certain limit based on the user defined values.
- By default, drag limit will be disabled for the connector. It can be enabled by setting connector constraints as drag.
- The following code illustrates how to set a dragLimit for connector annotations.

### APP.VUE

```

<template>
  <div id="app">
    <ejs-diagram id="diagram" :width='width' :height='height'
:connectors='connectors' ></ejs-diagram>
  </div>
</template>
<script>
  import Vue from 'vue';
  import { DiagramPlugin, AnnotationConstraints } from '@syncfusion/ej2-
vue-diagrams';
  Vue.use(DiagramPlugin);
  let connectors = [
    {
      id: 'connector2',
      type: 'Orthogonal',
      sourcePoint: { x: 300, y: 300 },
      targetPoint: { x: 400, y: 400 },
      annotations: [
        {
          content: 'connector1', offset: 0.5,
          //Enables drag constraints for a connector.
          constraints: AnnotationConstraints.Interaction |
AnnotationConstraints.Drag,
          //Set drag limit for a connector annotation.
          dragLimit: {left: 20, right: 20, top: 20, bottom: 20}
        }
      ],
    }
  ]
  export default {
    name: 'app'
    data() {
      return {
        width: "100%",
        height: "350px",
        connectors: connectors,
      }
    }
  }
</script>
<style>
  @import "../../node_modules/@syncfusion/ej2-vue-
diagrams/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/diagram/labels/Interaction-cs2" %}

### Multiple annotations

You can add any number of annotations to a node or connector. The following code illustrates how to add multiple annotations to a node.

#### APP.VUE

```

<template>
  <div id="app">

```

```

    <ejs-diagram id="diagram" :width='width' :height='height'
:nodes='nodes' ></ejs-diagram>
  </div>
</template>
<script>
  import Vue from 'vue';
  import { DiagramPlugin } from '@syncfusion/ej2-vue-diagrams';
  Vue.use(DiagramPlugin);
  let nodes = [{
    id: 'node1',
    // Position of the node
    offsetX: 100,
    offsetY: 100,
    // Size of the node
    width: 100,
    height: 100,
    style: {
      fill: '#6BA5D7',
      strokeColor: 'white'
    },
    // Sets the multiple annotation for the node
    annotations: [{
      content: 'Left',
      offset: {
        x: 0.12,
        y: 0.1
      }
    },
    {
      content: 'Center',
      offset: {
        x: 0.5,
        y: 0.5
      }
    },
    {
      content: 'Right',
      offset: {
        x: 0.82,
        y: 0.9
      }
    }
  ]
  }]
  export default {
    name: 'app'
    data() {
      return {
        width: "100%",
        height: "350px",
        nodes: nodes,
      }
    }
  }
</script>
<style>

```

```
@import "../../../node_modules/@syncfusion/ej2-vue-  
diagrams/styles/material.css";  
</style>
```

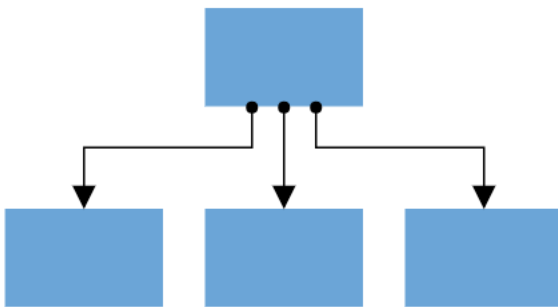
{% previewsample "page.domainurl/code-snippet/diagram/labels/Multiple-cs1" %}

### Constraints

The constraints property of annotation allows you to enable/disable certain annotation behaviors. For instance, you can disable annotation editing.

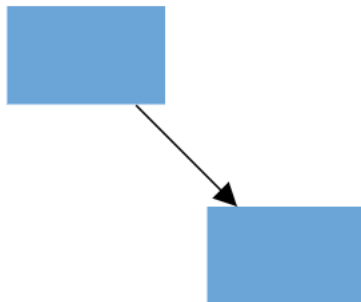
### Ports in Vue Diagram component

Diagram provides support to define custom ports for making connections.

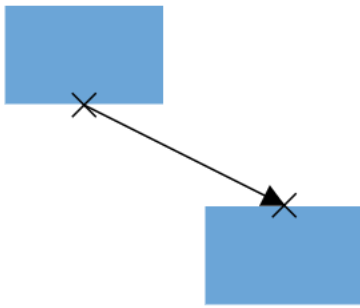


<!-- markdownlint-disable MD033 -->

When a connector is connected between two nodes, its end points are automatically docked to the node's nearest boundary as shown in the following image.



Ports act as the connection points of the node and allows to create connections with only those specific points as shown in the following image.



### Create Port

#### Add ports when initializing nodes

To add a connection port, define the port object and add it to node's ports collection. The `offset` property of port accepts an object of fractions and used to determine the position of ports. The following code illustrates how to add ports when initializing the node.

#### APP.VUE

```
<template>
  <div id="app">
    <ejs-diagram id="diagram" :width='width' :height='height'
:nodes='nodes' ></ejs-diagram>
  </div>
</template>
<script>
  import Vue from 'vue';
  import { DiagramPlugin, PortVisibility } from '@syncfusion/ej2-vue-
diagrams';
  Vue.use(DiagramPlugin);
  let nodes = [{
    // Position of the node
    offsetX: 250,
    offsetY: 250,
    // Size of the node
    width: 100,
    height: 100,
    style: {
      fill: '#6BA5D7',
      strokeColor: 'white'
    },
    // Initialize port collection
    ports: [{
      // Sets the position for the port
      offset: {
        x: 0.5,
        y: 0.5
      },
      visibility: PortVisibility.Visible
    }]
  }]
  export default {
    name: 'app'
    data() {
```

```

        return {
            width: "100%",
            height: "350px",
            nodes: nodes,
        }
    }
}
</script>
<style>
    @import "../node_modules/@syncfusion/ej2-vue-
    diagrams/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/diagram/ports/Remove-cs1" %}

### Add ports at runtime

Add ports at runtime by using the client-side method [addPorts](#). The following code illustrates how to add ports to node at runtime.

The port's ID property is used to define the unique ID for the port and its further used to find the port at runtime.

If ID is not set, then default ID is automatically set.

### APP.VUE

```

<template>
    <div id="app">
        <ejs-diagram id="diagram" :width='width' :height='height'
        :nodes='nodes' ></ejs-diagram>
    </div>
</template>
<script>
    import Vue from 'vue';
    import { DiagramPlugin , PointPortModel, PortVisibility} from
    '@syncfusion/ej2-vue-diagrams';
    Vue.use(DiagramPlugin);
    let nodes = [{
        // Position of the node
        offsetX: 250,
        offsetY: 250,
        // Size of the node
        width: 100,
        height: 100,
        style: {
            fill: '#6BA5D7',
            strokeColor: 'white'
        },
    },
    ];
    let port: PointPortModel[] = [{
        id: 'port1',
        offset: {
            x: 0,
            y: 0.5
        },
        visibility: PortVisibility.Visible
    } {

```

```

        id: 'port2',
        offset: {
            x: 1,
            y: 0.5
        },
        visibility: PortVisibility.Visible
    },
    {
        id: 'port3',
        offset: {
            x: 0.5,
            y: 0
        },
        visibility: PortVisibility.Visible
    },
    {
        id: 'port4',
        offset: {
            x: 0.5,
            y: 1
        },
        visibility: PortVisibility.Visible
    }
];
export default {
    name: 'app'
    data() {
        return {
            width: "100%",
            height: "350px",
            nodes: nodes,
        }
    }
    mounted: function() {
        let diagramInstance: Diagram;
        let diagramObj: any = document.getElementById("diagram");
        diagramInstance = diagramObj.ej2_instances[0];
        // Adds to the Diagram
        diagramInstance.addPorts(diagramInstance.nodes[0], port);
    }
}
</script>
<style>
    @import "../../../node_modules/@syncfusion/ej2-vue-
    diagrams/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/diagram/ports/Node-cs1" %}

### Remove ports at runtime

Remove ports at runtime by using client-side method [removePorts](#). Refer to the following example which shows how to remove ports at runtime.

### APP.VUE

```
<template>
```



```

<div id="app">
  <ejs-diagram id="diagram" :width='width' :height='height'
:nodes='nodes' ></ejs-diagram>
</div>
</template>
<script>
  import Vue from 'vue';
  import { DiagramPlugin, PortVisibility, PointPortModel } from
'@syncfusion/ej2-vue-diagrams';
  Vue.use(DiagramPlugin);
  let nodes = [{
    // Position of the node
    offsetX: 250,
    offsetY: 250,
    // Size of the node
    width: 100,
    height: 100,
    style: {
      fill: '#6BA5D7',
      strokeColor: 'white'
    },
    // Initialize port collection
    ports: [{
      id: 'port1',
      offset: {
        x: 0,
        y: 0.5
      },
      visibility: PortVisibility.Visible
    },
    {
      id: 'port2',
      offset: {
        x: 1,
        y: 0.5
      },
      visibility: PortVisibility.Visible
    },
    {
      id: 'port3',
      offset: {
        x: 0.5,
        y: 0
      },
      visibility: PortVisibility.Visible
    },
    {
      id: 'port4',
      offset: {
        x: 0.5,
        y: 1
      },
      visibility: PortVisibility.Visible
    }
  ]
  }]
  export default {

```

```

    name: 'app'
    data() {
      return {
        width: "100%",
        height: "350px",
        nodes: nodes,
      }
    }
    mounted: function() {
      let ports: PointPortModel[] = [{
        id: 'port1',
      }, {
        id: 'port2',
      }, {
        id: 'port3',
      }, {
        id: 'port4',
      }]
      let diagramInstance: Diagram;
      let diagramObj: any = document.getElementById("diagram");
      diagramInstance = diagramObj.ej2_instances[0];
      // Adds to the Diagram
      diagramInstance.removePorts(diagramInstance.nodes[0], ports);
    }
  }
</script>
<style>
  @import "../node_modules/@syncfusion/ej2-vue-
  diagrams/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/diagram/ports/ports-cs1" %}

### Update Port at runtime

You can change any port properties at runtime and update it through the client-side method [dataBind](#).

The following code example illustrates how to change the port properties.

### APP.VUE

```

<template>
  <div id="app">
    <ejs-diagram id="diagram" :width='width' :height='height'
    :nodes='nodes' ></ejs-diagram>
  </div>
</template>
<script>
  import Vue from 'vue';
  import { DiagramPlugin, PortVisibility } from '@syncfusion/ej2-vue-
  diagrams';
  Vue.use(DiagramPlugin);
  let nodes = [{
    // Position of the node
    offsetX: 250,
    offsetY: 250,
    // Size of the node
    width: 100,

```

```

    height: 100,
    style: {
      fill: '#6BA5D7',
      strokeColor: 'white'
    },
    // Initialize port collection
    ports: [{
      offset: {
        x: 0.5,
        y: 0.5
      },
      visibility: PortVisibility.Visible
    }]
  ]
}
export default {
  name: 'app'
  data() {
    return {
      width: "100%",
      height: "350px",
      nodes: nodes,
    }
  },
  mounted: function() {
    let diagramInstance: Diagram;
    let diagramObj: any = document.getElementById("diagram");
    diagramInstance = diagramObj.ej2_instances[0];
    diagramInstance.nodes[0].ports[0].offset = {
      x: 1,
      y: 1
    };
  }
}
</script>
<style>
  @import
  "https://ej2.syncfusion.com/vue/documentation/../../node_modules/@syncfusion/ej2-vue-diagrams/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/diagram/ports/ports-cs2" %}

### Appearance

- The shape of port can be changed by using its shape property. To explore the different types of port shapes, refer to Port Shapes. If you need to render a custom shape, then you can set shape as path and define path using path data property of port.
- The appearance of ports can be customized by using [strokeColor](#),

[strokeWidth](#), and [fill](#) properties of the port.

- Customize the port size by using the [width](#) and [height](#) properties of port.
- The ports [visibility](#) property allows you to define, when the port should be visible.

The following code illustrates how to change the appearance of port.

### **APP.VUE**

```
<template>
  <div id="app">
    <ejs-diagram id="diagram" :width='width' :height='height'
:nodes='nodes' ></ejs-diagram>
  </div>
</template>
<script>
  import Vue from 'vue';
  import { DiagramPlugin, PortVisibility } from '@syncfusion/ej2-vue-
diagrams';
  Vue.use(DiagramPlugin);
  let nodes = [{
    // Position of the node
    offsetX: 250,
    offsetY: 250,
    // Size of the node
    width: 100,
    height: 100,
    style: {
      fill: '#6BA5D7',
      strokeColor: 'white'
    },
  },
  // Initialize port collection
  ports: [{
    offset: {
      x: 1,
      y: 0.5
    },
    visibility: PortVisibility.Visible,
    //Set the style for the port
    style: {
      fill: 'red',
      strokeWidth: 2,
      strokeColor: 'black'
    },
    width: 12,
    height: 12,
    // Sets the shape of the port as Circle
    shape: 'Circle'
  }
  ]
  }
  export default {
    name: 'app'
    data() {
      return {
        width: "100%",
        height: "350px",
        nodes: nodes,
      }
    }
  }
</script>
<style>
```

```
@import "../../../node_modules/@syncfusion/ej2-vue-  
diagrams/styles/material.css";  
</style>
```

```
{% previewsample "page.domainurl/code-snippet/diagram/ports/ports-cs3" %}
```

### Offset

The offset property of port is used to align the port based on fractions. 0 represents top/left corner, 1 represents bottom/right corner, and 0.5 represents half of width/height.

### Constraints

The constraints property allows to enable/disable certain behaviors of ports. For more information about port constraints, refer to [Port Constraints](#).

### Constraints in Vue Diagram component

Constraints are used to enable/disable certain behaviors of the diagram, nodes and connectors. Constraints are provided as flagged enumerations, so that multiple behaviors can be enabled/disabled using Bitwise operators (&, |, ~, <<, etc.).

To know more about Bitwise operators, refer to [Bitwise Operations](#).

### Diagram constraints

Diagram constraints allow to enable or disable the following behaviors:

- Page editing
- Bridging
- Zoom and pan
- Undo/redo
- Tooltip

The following example illustrates how to disable page editing using the diagram constraints.

```
`javascript  
export default {  
  name: 'app'  
  data() {  
    return {  
      width: "100%",  
      height: "350px",  
      constraints: DiagramConstraints.Default & ~DiagramConstraints.PageEditable  
    }  
  }  
}
```

For more information about diagram constraints, refer to [DiagramConstraints](#).

### Node constraints

Node constraints allows to enable or disable the following behaviors of node. They are as follows:

- Selection
- Deletion
- Drag
- Resize
- Rotate
- Connect
- Shadow
- Tooltip

The following example illustrates how to disable rotation using the node constraints.

```
`javascript
let nodes = [{
  id: "node1",
  height: 60,
  offsetX: 300,
  offsetY: 80,
  constraints: NodeConstraints.Default & ~NodeConstraints.Rotate,
  annotations: [{
    content: "start"
  }]
}]

export default {
  name: 'app'
  data() {
    return {
      width: "100%",
      height: "350px",
      nodes: nodes,
    }
  }
}
```

For more information about node constraints, refer to [NodeConstraints](#).

### Connector constraints

Connector constraints allow to enable or disable certain behaviors of connectors.

- Selection
- Deletion
- Drag
- Segment editing
- Tooltip
- Bridging

The following code illustrates how to disable selection by using connector constraints.

```
`javascript
let connectors = [{
  id: 'connector1',
  type: 'Straight',
  sourcePoint: {
    x: 100,
    y: 100
  },
  targetPoint: {
    x: 200,
    y: 200
  },
  constraints: ConnectorConstraints.Default & ~ConnectorConstraints.Select
}]

export default {
  name: 'app'
  data() {
    return {
      width: "100%",
      height: "350px",
      connectors: connectors,
    }
  }
}
```

For more information about connector constraints, refer to [ConnectorConstraints](#).

### Port constraints

You can enable or disable certain behaviors of port. They are as follows:

- Connect
- ConnectOnDrag

The following code illustrates how to disable creating connections with a port.

```
`javascript
let nodes = [{
  id: 'connector1',
  type: 'Straight',
  sourcePoint: {
    x: 100,
    y: 100
  },
  targetPoint: {
    x: 200,
    y: 200
  },
  ports: [{
    constraints: PortConstraints.None
  }]
}]

export default {
  name: 'app'
  data() {
    return {
      width: "100%",
      height: "350px",
      nodes: nodes,
    }
  }
}
```



For more information about port constraints, refer to [PortConstraints](#).

#### Annotation constraints

You can enable or disable read-only mode for the annotations by using the annotation constraints.

The following code illustrates how to enable read-only mode for the annotations.

```
`javascript
let nodes = [{
  id: 'connector1',
  type: 'Straight',
  sourcePoint: {
    x: 100,
    y: 100
  },
  targetPoint: {
    x: 200,
    y: 200
  },
  annotations: [{
    id: 'anotation_1',
    content: 'annotation',
    constraints: AnnotationConstraints.ReadOnly,
  }]
}]

export default {
  name: 'app'
  data() {
    return {
      width: "100%",
      height: "350px",
      nodes: nodes,
    }
  }
}
```

For more details about annotation constraints, refer to [AnnotationConstraints](#).

### Selector constraints

Selector visually represents the selected elements with certain editable thumbs. The visibility of the thumbs can be controlled with selector constraints. The part of selector is categorized as follows:

- Resizer
- Rotator
- User handles

The following code illustrates how to hide rotator.

```
`javascript
export default {
  name: 'app'
  data() {
    return {
      width: "100%",
      height: "350px",
      selectedItems: {
        constraints: SelectorConstraints.All & ~SelectorConstraints.Rotate
      }
    }
  }
}
```

For more information about selector constraints, refer to [SelectorConstraints](#).

### Snap constraints

Snap constraints control the visibility of gridlines and enable/disable snapping. Snap constraints allow to set the following behaviors.

- Show only horizontal or vertical gridlines.
- Show both horizontal and vertical gridlines.
- Snap to either horizontal or vertical gridlines.
- Snap to both horizontal and vertical gridlines.

The following code illustrates how to show only horizontal gridlines.

```
`javascript
export default {
  name: 'app'
```

```
data() {  
  return {  
    width: "100%",  
    height: "350px",  
    snapSettings: {  
      constraints: SnapConstraints.ShowHorizontalLines  
    }  
  }  
}
```

For more information about snap constraints, refer to [SnapConstraints](#).

### Boundary constraints

Boundary constraints defines a boundary for the diagram inside which the interaction should be done. Boundary constraints allow to set the following behaviors.

- Infinite boundary
- Diagram sized boundary
- Page sized boundary

The following code illustrates how to limit the interaction done inside a diagram within a page.

```
`javascript  
export default {  
  name: 'app'  
  data() {  
    return {  
      width: "100%",  
      height: "350px",  
      pageSettings: {  
        boundaryConstraints: 'Page'  
      }  
    }  
  }  
}
```

For more information about selector constraints, refer to [BoundaryConstraints](#).

### Inherit behaviors

Some of the behaviors can be defined through both the specific object (node/connector) and diagram. When the behaviors are contradictorily defined through both, the actual behavior is set through inherit options.

The following code example illustrates how to inherit the line bridging behavior from the diagram model.

```
`javascript
Diagram.Inject(ConnectorBridging);
let connectors[] = [{
  id: 'connector1',
  type: 'Straight',
  sourcePoint: {
    x: 100,
    y: 100
  },
  targetPoint: {
    x: 200,
    y: 200
  },
  constraints: ConnectorConstraints.Default & ConnectorConstraints.InheritBridging
}];
export default {
  name: 'app'
  data() {
    return {
      width: "100%",
      height: "350px",
      connectors: connectors,
      constraints: DiagramConstraints.Default | DiagramConstraints.Bridging
    }
  }
}
```

### Bitwise operations

Bitwise operations are used to manipulate the flagged enumerations [enum]. In this section, Bitwise operations are illustrated by using node constraints. The same is applicable while working with node constraints, connector constraints, or port constraints.

#### Add operation

You can add or enable multiple values at a time by using Bitwise ‘|’ (OR) operator.

```
`javascript
```

```
node.constraints = NodeConstraints.Select | NodeConstraints.Rotate;
```

```
,
```

In the previous example, you can do both the selection and rotation operation.

#### Remove Operation

You can remove or disable values by using Bitwise ‘&~’ (XOR) operator.

```
`javascript
```

```
node.constraints = node.constraints & ~(NodeConstraints.Rotate);
```

```
,
```

In the previous example, rotation is disabled but other constraints are enabled.

#### Check Operation

You can check any value by using Bitwise ‘&’ (AND) operator.

```
`javascript
```

```
if ((node.constraints & (NodeConstraints.Rotate)) == (NodeConstraints.Rotate));
```

```
,
```

In the previous example, check whether the rotate constraints are enabled in a node. When node constraints have rotate constraints, the expression returns a rotate constraint.

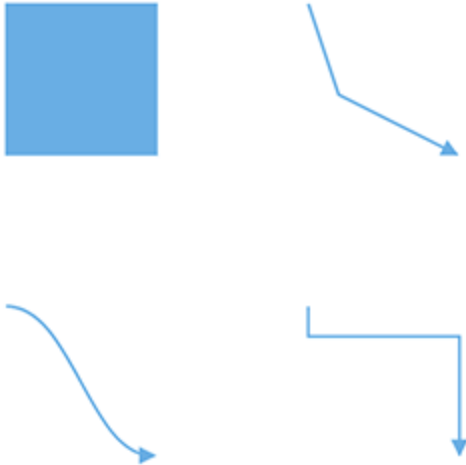
### Interaction in Vue Diagram component

#### Selection

Selector provides a visual representation of selected elements. It behaves like a container and allows to update the size, position, and rotation angle of the selected elements through interaction and by using program. Single or multiple elements can be selected at a time.

#### Single selection

An element can be selected by clicking that element. During single click, all previously selected items are cleared. The following image shows how the selected elements are visually represented.



- While selecting the diagram elements, the following events can be used to do your customization.
- When selecting/unselecting the diagram elements, the [selectionChange](#) event gets triggered.

### Selecting a group

When a child element of any group is clicked, its contained group is selected instead of the child element. With consecutive clicks on the selected element, selection is changed from top to bottom in the hierarchy of parent group to its children.

### Multiple selection

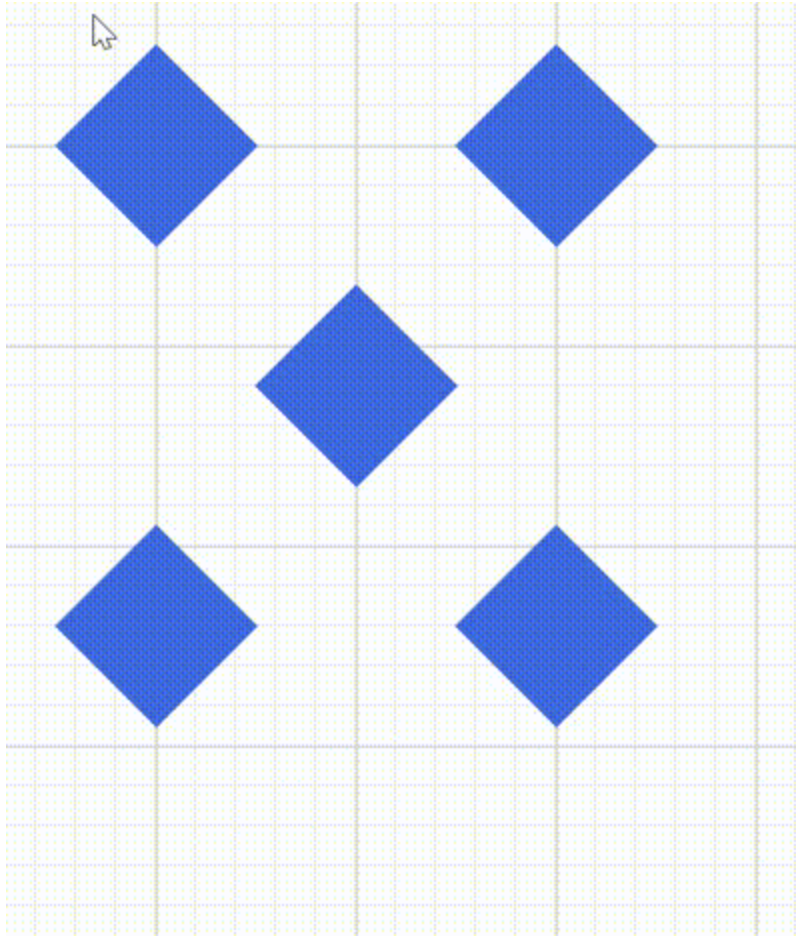
Multiple elements can be selected with the following ways:

- Ctrl+Click

During single click, any existing item in the selection list be cleared, and only the item clicked recently is there in the selection list. To avoid cleaning the old selected item, Ctrl key must be on hold when clicking.

- Selection rectangle/rubber band selection

Clicking and dragging the diagram area allows to create a rectangular region. The elements that are covered under the rectangular region are selected at the end.



#### Select/Unselect elements using program

The client-side methods [select](#) and [clearSelection](#) help to select or clear the selection of the elements at runtime. The following code example illustrates how to select or clear the selection of an item using program.

Get the current selected items from the [nodes](#) and [connectors](#) collection of the [selectedItems](#) property of the diagram model.

#### Select entire elements in diagram programmatically

The client-side method [selectAll](#) used to select all the elements such as nodes/connectors in the diagram. Refer to the following link which shows how to use [selectAll](#) method on the diagram.

#### Drag

- An object can be dragged by clicking and dragging it. When multiple elements are selected, dragging any one of the selected elements move every selected element.
- When you drag the elements in the diagram, the [positionChange](#) event gets triggered and to do customization in this event.



## Resize

- Selector is surrounded by eight thumbs. When dragging these thumbs, selected items can be resized.
- When one corner of the selector is dragged, opposite corner is in a static position.
- When a node is resized, the [sizeChange](#) event gets triggered.



Note: While dragging and resizing, the objects are snapped towards the nearest objects to make better alignments. For better alignments, refer to **Snapping**.

## Customize the resize-thumb

You can change the size of the node resize thumb and the connector end point handle by using the **handleSize** property. The appearance such as fill, stroke, and stroke width of the node resize thumb and connector end point handle can be customized by overriding the **e-diagram-resize-handle** and **e-diagram-endpoint-handle** classes respectively.

## APP.VUE

```
<template>
  <div id="app">
```



```

    <ejs-diagram id="diagram" :width='width' :height='height'
:nodes='nodes' :selectedItems='selectedItems' ></ejs-diagram>
  </div>
</template>
<script>
  import Vue from 'vue';
  import { DiagramPlugin, Diagram, NodeModel, SelectorConstraints } from
'@syncfusion/ej2-vue-diagrams';
  Vue.use(DiagramPlugin);
  let nodes = [{
    // Position of the node
    offsetX: 250,
    offsetY: 250,
    // Size of the node
    width: 100,
    height: 100,
    style: {
      fill: '#6BA5D7',
      strokeColor: 'white'
    },
    // Text(label) added to the node
  }]
  export default {
    name: 'app',
    data() {
      return {
        width: "100%",
        height: "350px",
        nodes: nodes,
        selectedItems: { handleSize : 40 }
      }
    }
  }
</script>
<style>
  @import "../../node_modules/@syncfusion/ej2-vue-
diagrams/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/diagram/interaction/resizeThumb-cs1" %}



### Rotate

- A rotate handler is placed above the selector. Clicking and dragging the handler in a circular direction lead to rotate the node.
- The node is rotated with reference to the static pivot point.
- Pivot thumb (thumb at the middle of the node) appears while rotating the node to represent the static point.



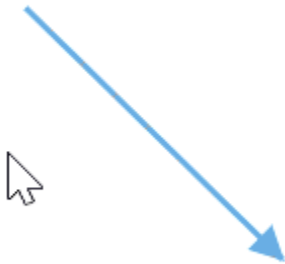
### Connection editing

- Each segment of a selected connector is editable with some specific handles/thumbs.

Note: For connector editing, you have to inject the [ConnectorEditing](#) module.

### End point handles

Source and target points of the selected connectors are represented with two handles. Clicking and dragging those handles help you to adjust the source and target points.



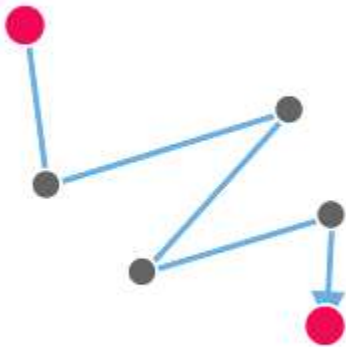
- If you drag the connector end points, then the following events can be used to do your customization.
- When the connector source point is changed, the [sourcePointChange](#) event gets triggered.
- When the connector target point is changed, the [targetPointChange](#) event gets triggered.
- When you connect connector with ports/node or disconnect from it, the [connectionChange](#) event gets triggered.

### Straight segment editing

- End point of each straight segment is represented by a thumb that enables to edit the segment.
- Any number of new segments can be inserted into a straight line by clicking, when Shift and Ctrl keys are pressed (Ctrl+Shift+Click).



- Straight segments can be removed by clicking the segment end point, when Ctrl and Shift keys are pressed (Ctrl+Shift+Click).



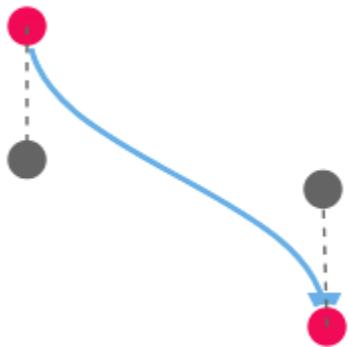
### Orthogonal thumbs

- Orthogonal thumbs allow you to adjust the length of adjacent segments by clicking and dragging it.
- When necessary, some segments are added or removed automatically, when dragging the segment. This is to maintain proper routing of orthogonality between segments.



### Bezier thumbs

- Bezier segments are annotated with two thumbs to represent the control points. Control points of the curve can be configured by clicking and dragging the control thumbs.



### Drag and drop nodes over other elements

Diagram provides support to drop a node/connector over another node/connector. The [drop](#) event is raised to notify that an element is dropped over another one and it is disabled, by default. It can be enabled with the `constraints` property.

### User handles

- User handles are used to add some frequently used commands around the selector. To create user handles, define and add them to the [userHandles](#) collection of the [selectedItems](#) property.
- The `name` property of user handle is used to define the name of the user handle and is further used to find the user handle at runtime and do any customization.

### Alignment

User handles can be aligned relative to the node boundaries. It has [margin](#), [offset](#), [side](#), [horizontalAlignment](#), and [verticalAlignment](#) settings. It is quite tricky when all four alignments are used together but gives more control over alignment.

### Offset

The [offset](#) property of [userHandles](#) is used to align the user handle based on fractions. 0 represents top/left corner, 1 represents bottom/right corner, and 0.5 represents half of width/height.

### Side

The [side](#) property of [userHandles](#) is used to align the user handle by using the [Top](#), [Bottom](#), [Left](#), and [Right](#) options.

### Horizontal and vertical alignments

The [horizontalAlignment](#) property of [userHandles](#) is used to set how the user handle is horizontally aligned at the position based on the [offset](#). The [verticalAlignment](#) property is used to set how user handle is vertically aligned at the position.

### Margin

Margin is an absolute value used to add some blank space in any one of its four sides. The [userHandles](#) can be displaced with the [margin](#) property.

### Notification for the mouse button clicked

The diagram component notifies the mouse button clicked. For example, whenever the right mouse button is clicked, the clicked button is notified as right. Mouse click is notified with,

Notification	Description
Left	When the left mouse button is clicked, left is notified
Middle	When the mouse wheel is clicked, middle is notified
Right	When the right mouse button is clicked, right is notified

```
<template>
```

```
<div id="app">
```

```
<ejs-diagram id="diagram" :width='width' :height='height' :nodes='nodes' :click="click"></ejs-diagram>
```

```
</div>
```

```
</template>
```

```
<script>
```

```
import Vue from 'vue';
```

```
import { DiagramPlugin, DiagramTools } from '@syncfusion/ej2-vue-diagrams';
```

```
Vue.use(DiagramPlugin);
```

```
let nodes = [{
```

```
// Position of the node
```

```
offsetX: 250,
offsetY: 250,
// Size of the node
width: 100,
height: 100,
style: {
  fill: '#6BA5D7',
  strokeColor: 'white'
},
// Text(label) added to the node
}]
export default {
  name: 'app',
  data() {
    return {
      width: "100%",
      height: "700px",
      nodes: nodes,
      click: (args) => {
        //Obtains the button clicked
        let clickNotify = args.button
      }
    }
  },
}
```

</script>

,

### Appearance

The appearance of the user handle can be customized by using the [size](#), [borderColor](#), [backgroundColor](#), [visible](#), [pathData](#), and [pathColor](#) properties of the [userHandles](#).

### APP.VUE

```
<template>
  <div id="app">
```

```

    <ejs-diagram id="diagram" :width='width' :height='height'
:nodes='nodes' :getCustomTool='getCustomTool'
:selectedItems='selectedItems'></ejs-diagram>
  </div>
</template>
<script>
  import Vue from 'vue';
  import { DiagramPlugin, NodeModel
, MoveTool, BasicShapeModel, UserHandleModel, randomId,
SelectorConstraints, cloneObject } from '@syncfusion/ej2-vue-diagrams';
  Vue.use(DiagramPlugin);
  let diagramInstance: Diagram;
  let shape: BasicShapeModel = {
    type: 'Basic',
    shape: 'Rectangle'
  };
  function getTool(action: string): ToolBase {
    let tool: ToolBase | any;
    if (action === "clone") {
      tool = new CloneTool(diagramInstance.commandHandler);
    }
    return tool;
  }
  class CloneTool extends MoveTool {
    public mouseDown(args: MouseEventArgs): void {
      let newObject: any;
      if (diagramInstance.selectedItems.nodes.length > 0) {
        newObject =
cloneObject(diagramInstance.selectedItems.nodes[0]) as NodeModel;
      } else {
        newObject =
cloneObject(diagramInstance.selectedItems.connectors[0]) as ConnectorModel;
      }
      newObject.id += randomId();
      diagramInstance.paste([newObject]);
      args.source = diagramInstance.nodes[diagramInstance.nodes.length
- 1] as IElement;
      args.sourceWrapper = args.source.wrapper;
      super.mouseDown(args);
      this.inAction = true;
    }
  }
  let nodes = [{
    id: 'node',
    offsetX: 100,
    offsetY: 100,
    shape: shape
  }]
  let handles: UserHandleModel[] = [{
    name: 'clone',
    pathData: 'M60.3,18H27.5c-3,0-5.5,2.4-5.5,5.5v38.2h5.5V23.5h32.7V18z
M68.5,28.9h-30c-3,0-5.5,2.4-5.5,5.5v38.2c0,3,2.4,5.5,5.5,5.5h30c3,0,5.5-
2.4,5.5-5.5V34.4C73.9,31.4,71.5,28.9,68.5,28.9z M68.5,72.5h-
30V34.4h30V72.5z',
    visible: true,
    offset: 0,
    side: 'Bottom',
  }]

```



```

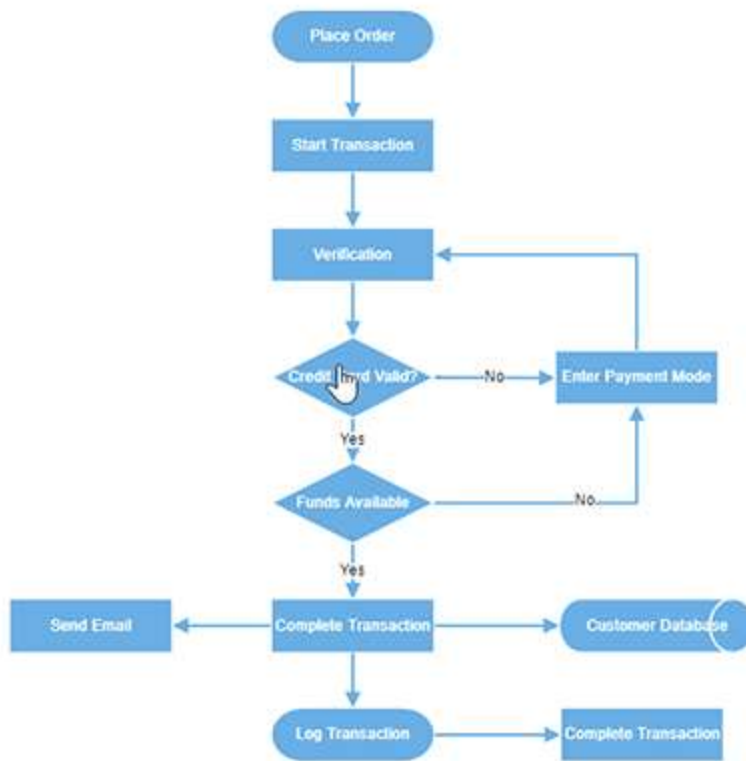
        margin: {
            top: 0,
            bottom: 0,
            left: 0,
            right: 0
        }
    }
    });
    export default {
        name: 'app'
        data() {
            return {
                width: "100%",
                height: "350px",
                nodes: nodes,
                selectedItems: {
                    constraints: SelectorConstraints.UserHandle,
                    userHandles: handles
                },
                getCustomTool: getTool
            }
        }
        mounted: function() {
            let obj: any = document.getElementById("diagram");
            diagramInstance = obj.ej2_instances[0];
        }
    }
</script>
<style>
    @import
    "https://ej2.syncfusion.com/vue/documentation/../../node_modules/@syncfusion/ej2-vue-diagrams/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/diagram/interaction/UserHandle-cs1" %}

### Zoom pan

- When a large diagram is loaded, only certain portion of the diagram is visible. The remaining portions are clipped. Clipped portions can be explored by scrolling the scrollbars or panning the diagram.
- Diagram can be zoomed in or out by using Ctrl + mouse wheel.
- When the diagram is zoomed or panned, the [scrollChange](#) event gets triggered.



### Zoom pan status

Diagram provides the support to notify the pan status of the zoom pan tool. When ever the diagram is panning the [scrollChange](#) event is triggered and hence the pan status can be obtained. The pan status is notified with Start, Progress, and Completed.

Pan Status	Description
Start	When the mouse is clicked and dragged the status is notified as start.
Progress	When the mouse is in motion the status is notified as progress.
Completed	When panning is stopped the status is notified with completed.

```

<template>
<div id="app">
  <ejs-diagram id="diagram" :width='width' :height='height' :nodes='nodes' :tool='tools'
  :scrollChange="scrollChange"></ejs-diagram>
</div>
</template>
<script>
import Vue from 'vue';

```

```
import { DiagramPlugin, DiagramTools } from '@syncfusion/ej2-vue-diagrams';
Vue.use(DiagramPlugin);
let nodes = [{
  // Position of the node
  offsetX: 250,
  offsetY: 250,
  // Size of the node
  width: 100,
  height: 100,
  style: {
    fill: '#6BA5D7',
    strokeColor: 'white'
  },
  // Text(label) added to the node
}]
export default {
  name: 'app',
  data() {
    return {
      width: "100%",
      height: "700px",
      nodes: nodes,
      tools: DiagramTools.ZoomPan,
      scrollChange: (args) => {
        //Obtains the pan status
        let panStatus = args.panState
      },
    }
  },
}
```

</script>

,

## Keyboard

Diagram provides support to interact with the elements with key gestures. By default, some in-built commands are bound with a relevant set of key combinations.

The following table illustrates those commands with the associated key values.

Shortcut Key	Command	Description
----- ----- -----		
Ctrl + A	<code>selectAll</code>	Select all nodes/connectors in the diagram.
Ctrl + C	<code>copy</code>	Copy the diagram selected elements.
Ctrl + V	<code>paste</code>	Pastes the copied elements.
Ctrl + X	<code>cut</code>	Cuts the selected elements.
Ctrl + Z	<code>undo</code>	Reverses the last editing action performed on the diagram.
Ctrl + Y	<code>redo</code>	Restores the last editing action when no other actions have occurred since the last undo on the diagram.
Delete	<code>delete</code>	Deletes the selected elements.
Ctrl/Shift + Click on object		Multiple selection (Selector binds all selected nodes/connectors).
Up Arrow	<code>nudge("up")</code>   <code>nudgeUp</code>	Moves the selected elements towards up by one pixel.
Down Arrow	<code>nudge("down")</code>   <code>nudgeDown</code>	Moves the selected elements towards down by one pixel.
Left Arrow	<code>nudge("left")</code>   <code>nudgeLeft</code>	Moves the selected elements towards left by one pixel.
Right Arrow	<code>nudge("right")</code>   <code>nudgeRight</code>	Moves the selected elements towards right by one pixel.
Ctrl + MouseWheel	<code>zoom</code>	Zoom (Zoom in/Zoom out the diagram).
F2	<code>startLabelEditing</code>	Starts to edit the label of selected element.
Esc	<code>endLabelEditing</code>	Sets the label mode as view and stops editing.

## See Also

- [How to create diagram nodes using drawing tools](#)
- [How to create diagram connectors using drawing tools](#)
- [How to disable the diagram interaction](#)
- [How to control the diagram history](#)
- [How to create overview control to the diagram](#)

## Tools in Vue Diagram component

### Drawing tools

Drawing tool allows you to draw any kind of node/connector during runtime by clicking and dragging on the diagram page.

## Shapes

To draw a shape, set the JSON of that shape to the drawType property of the diagram and activate the drawing tool by using the [tool](#) property. The following code example illustrates how to draw a rectangle at runtime.

### APP.VUE

```
<template>
  <div id="app">
    <ejs-diagram id="diagram" :width='width' :height='height'
      :getNodeDefaults='getNodeDefaults'></ejs-diagram>
    </div>
  </template>
<script>
  import Vue from 'vue';
  import { DiagramPlugin, DiagramTools, BasicShapeModel, NodeModel } from
  '@syncfusion/ej2-vue-diagrams';
  Vue.use(DiagramPlugin);
  export default {
    name: 'app'
    data() {
      return {
        width: "100%",
        height: "350px",
        getNodeDefaults: (obj) => {
          obj.height = 15;
          obj.width = 15;
          obj.borderWidth = 1;
          obj.style = {
            fill: '#6BA5D7',
            strokeWidth: 2,
            strokeColor: '#6BA5D7'
          };
          return obj;
        },
      }
    }
    mounted: function() {
      let diagramInstance: Diagram;
      let diagramObj: any = document.getElementById("diagram");
      diagramInstance = diagramObj.ej2_instances[0];
      let drawingshape: BasicShapeModel = { type: 'Basic', shape:
      'Rectangle' };
      let node: NodeModel = {
        shape: drawingshape
      };
      diagramInstance.drawingObject = node;
      //To draw an object once, activate draw once
      diagramInstance.tool = DiagramTools.DrawOnce;
      diagramInstance.dataBind();
    }
  }
</script>
<style>
  @import "../node_modules/@syncfusion/ej2-vue-
  diagrams/styles/material.css";
</style>
```

```
{% previewsample "page.domainurl/code-snippet/diagram/Tools/shape-cs1" %}
```

## Path

The following code example illustrates how to draw a path.

### APP.VUE

```
<template>
  <div id="app">
    <ejs-diagram id="diagram" :width='width' :height='height'
      :getNodeDefaults='getNodeDefaults'></ejs-diagram>
    </div>
  </template>
<script>
  import Vue from 'vue';
  import { DiagramPlugin, DiagramTools, BasicShapeModel, NodeModel } from
  '@syncfusion/ej2-vue-diagrams';
  Vue.use(DiagramPlugin);
  export default {
    name: 'app'
    data() {
      return {
        width: "100%",
        height: "350px",
        getNodeDefaults: (obj) => {
          obj.height = 15;
          obj.width = 15;
          obj.borderWidth = 1;
          obj.style = {
            fill: '#6BA5D7',
            strokeWidth: 2,
            strokeColor: '#6BA5D7'
          };
          return obj;
        },
      }
    }
    mounted: function() {
      let diagramInstance: Diagram;
      let diagramObj: any = document.getElementById("diagram");
      diagramInstance = diagramObj.ej2_instances[0];
      let node: NodeModel = {
        id: "Path",
        style: { fill: "#f8e172" },
        annotations: [{
          content: "Path"
        }],
        shape: {
          type: 'Path',
          data: 'M13.560 67.524 L 21.941 41.731 L 0.000 25.790 L
27.120 25.790 L 35.501 0.000 L 43.882 25.790 L 71.000 25.790 L 49.061 41.731
L 57.441 67.524 L 35.501 51.583 z'
        } as PathModel
      };
      diagramInstance.drawingObject = node;
    }
  }
}
```

```

        //To draw an object once, activate draw once
        diagramInstance.tool = DiagramTools.DrawOnce;
        diagramInstance.dataBind();
    }
}
</script>
<style>
    @import "../node_modules/@syncfusion/ej2-vue-
    diagrams/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/diagram/Tools/path-cs1" %}

### Connectors

To draw connectors, set the JSON of the connector to the drawType property. The drawing [tool](#) can be activated by using the tool property. The following code example illustrates how to draw a straight line connector.

### APP.VUE

```

<template>
    <div id="app">
        <ejs-diagram id="diagram" :width='width' :height='height'
        :getNodeDefaults='getNodeDefaults'></ejs-diagram>
    </div>
</template>
<script>
    import Vue from 'vue';
    import { DiagramPlugin, DiagramTools, BasicShapeModel, ConnectorModel }
    from '@syncfusion/ej2-vue-diagrams';
    Vue.use(DiagramPlugin);
    export default {
        name: 'app'
        data() {
            return {
                width: "100%",
                height: "350px",
                getNodeDefaults: (obj) => {
                    obj.height = 15;
                    obj.width = 15;
                    obj.borderWidth = 1;
                    obj.style = {
                        fill: '#6BA5D7',
                        strokeWidth: 2,
                        strokeColor: '#6BA5D7'
                    };
                    return obj;
                },
            },
        },
        mounted: function() {
            let diagramInstance: Diagram;
            let diagramObj: any = document.getElementById("diagram");
            diagramInstance = diagramObj.ej2_instances[0];
            let connectors: ConnectorModel = {
                id: 'connector1',

```

```

        type: 'Straight',
        segments: [{ type: "polyline" }]
    }
    diagramInstance.drawingObject = connectors;
    //To draw an object once, activate draw once
    diagramInstance.tool = DiagramTools.DrawOnce;
    diagramInstance.dataBind();
}
}
</script>
<style>
    @import "../node_modules/@syncfusion/ej2-vue-
    diagrams/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/diagram/Tools/connector-cs1" %}

### Text

Diagram allows you to create a textNode, when you click on the diagram page. The following code illustrates how to draw a text.

### APP.VUE

```

<template>
    <div id="app">
        <ejs-diagram id="diagram" :width='width' :height='height'
        :getNodeDefaults='getNodeDefaults'></ejs-diagram>
    </div>
</template>
<script>
    import Vue from 'vue';
    import { DiagramPlugin
    , DiagramTools, BasicShapeModel, TextModel, NodeModel } from '@syncfusion/ej2-
    vue-diagrams';
    Vue.use(DiagramPlugin);
    export default {
        name: 'app'
        data() {
            return {
                width: "100%",
                height: "350px",
                getNodeDefaults: (obj) => {
                    obj.height = 15;
                    obj.width = 15;
                    obj.borderWidth = 1;
                    obj.style = {
                        fill: '#6BA5D7',
                        strokeWidth: 2,
                        strokeColor: '#6BA5D7'
                    };
                    return obj;
                },
            }
        }
        mounted: function() {
            let diagramInstance: Diagram;

```



```

    let diagramObj: any = document.getElementById("diagram");
    diagramInstance = diagramObj.ej2_instances[0];
    let node: NodeModel = {
        shape: {
            type: 'Text',
        } as TextModel
    };
    diagramInstance.drawingObject = node;
    //To draw an object once, activate draw once
    diagramInstance.tool = DiagramTools.DrawOnce;
    diagramInstance.dataBind();
}
}
</script>
<style>
    @import "../node_modules/@syncfusion/ej2-vue-
    diagrams/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/diagram/Tools/text-cs1" %}

Once you activate the TextTool, perform label editing of a node/connector.

### Polygon shape

Diagram allows to create the polygon shape by clicking and moving the mouse at runtime on the diagram page.

The following code illustrates how to draw a polygon shape.

### APP.VUE

```

<template>
    <div id="app">
        <ejs-diagram id="diagram" :width='width' :height='height'
        :getNodeDefaults='getNodeDefaults'></ejs-diagram>
    </div>
</template>
<script>
    import Vue from 'vue';
    import { DiagramPlugin
    ,DiagramTools,BasicShapeModel,BasicShapeModel,NodeModel} from
    '@syncfusion/ej2-vue-diagrams';
    Vue.use(DiagramPlugin);
    export default {
        name: 'app'
        data() {
            return {
                width: "100%",
                height: "350px",
                getNodeDefaults: (obj) => {
                    obj.height = 15;
                    obj.width = 15;
                    obj.borderWidth = 1;
                    obj.style = {
                        fill: '#6BA5D7',
                        strokeWidth: 2,
                        strokeColor: '#6BA5D7'
                    }
                }
            }
        }
    }

```

```

        };
        return obj;
    },
    }
}
mounted: function() {
    let diagramInstance: Diagram;
    let diagramObj: any = document.getElementById("diagram");
    diagramInstance = diagramObj.ej2_instances[0];
    let drawingshape: BasicShapeModel = { type: 'Basic', shape:
'Polygon' };
    //JSON to create a polygon
    let node: NodeModel = {
        shape: drawingshape
    };
    diagramInstance.drawingObject = node;
    //To draw an object once, activate draw once
    diagramInstance.tool = DiagramTools.DrawOnce;
    diagramInstance.dataBind();
}
}
</script>
<style>
    @import "../node_modules/@syncfusion/ej2-vue-
diagrams/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/diagram/Tools/polygon-cs1" %}

### Polyline Connector

Diagram allows to create the polyline segments with straight lines and angled vertices at the control points by clicking and moving the mouse at runtime on the diagram page.

The following code illustrates how to draw a polyline connector.

### APP.VUE

```

<template>
    <div id="app">
        <ejs-diagram id="diagram" :width='width' :height='height'
:getNodeDefaults='getNodeDefaults'></ejs-diagram>
    </div>
</template>
<script>
    import Vue from 'vue';
    import { DiagramPlugin, DiagramTools, ConnectorModel } from
'@syncfusion/ej2-vue-diagrams';
    Vue.use(DiagramPlugin);
    export default {
        name: 'app'
        data() {
            return {
                width: "100%",
                height: "350px",
                getNodeDefaults: (obj) => {
                    obj.height = 15;

```

```

        obj.width = 15;
        obj.borderWidth = 1;
        obj.style = {
            fill: '#6BA5D7',
            strokeWidth: 2,
            strokeColor: '#6BA5D7'
        };
        return obj;
    },
}
}
mounted: function() {
    let diagramInstance: Diagram;
    let diagramObj: any = document.getElementById("diagram");
    diagramInstance = diagramObj.ej2_instances[0];
    //JSON to create a polyline
    let connector: ConnectorModel = { id: 'connector1', type:
'Polyline' };
    diagramInstance.drawingObject = connector;
    //To draw an object once, activate draw once
    diagramInstance.tool = DiagramTools.DrawOnce;
    diagramInstance.dataBind();
}
}
</script>
<style>
    @import "../node_modules/@syncfusion/ej2-vue-
diagrams/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/diagram/Tools/polyline-cs1" %}

### Tool selection

There are some functionalities that can be achieved by clicking and dragging on the diagram surface. They are as follows.

- Draw selection rectangle: MultipleSelect tool
- Pan the diagram: Zoom pan
- Draw nodes/connectors: DrawOnce/DrawOnce

As all the three behaviors are completely different, you can achieve only one behavior at a time based on the tool that you choose. When more than one of those tools are applied, a tool is activated based on the precedence given in the following table.

Precedence	Tools	Description
1st	ContinuesDraw	Allows you to draw the nodes or connectors continuously. Once it is activated, you cannot perform any other interaction in the diagram.
2nd	DrawOnce	Allows you to draw a single node or connector. Once you complete the DrawOnce action, SingleSelect, and MultipleSelect tools are automatically enabled.

|3rd|ZoomPan|Allows you to pan the diagram. When you enable both the SingleSelect and ZoomPan tools, you can perform the basic interaction as the cursor hovers node/connector. Panning is enabled when cursor hovers the diagram.|

|4th|MultipleSelect|Allows you to select multiple nodes and connectors. When you enable both the MultipleSelect and ZoomPan tools, cursor hovers the diagram. When panning is enabled, you cannot select multiple nodes.|

|5th|SingleSelect|Allows you to select individual nodes or connectors.|

|6th|None|Disables all tools.|

Set the desired [tool](#) to the tool property of the diagram model. The following code illustrates how to enable Zoom pan in the diagram

### APP.VUE

```
<template>
  <div id="app">
    <ejs-diagram id="diagram" :width='width' :height='height'
    :getNodeDefaults='getNodeDefaults'></ejs-diagram>
  </div>
</template>
<script>
  import Vue from 'vue';
  import { DiagramPlugin, DiagramTools, BasicShapeModel, ConnectorModel }
  from '@syncfusion/ej2-vue-diagrams';
  Vue.use(DiagramPlugin);
  export default {
    name: 'app'
    data() {
      return {
        width: "100%",
        height: "350px",
        getNodeDefaults: (obj) => {
          obj.height = 15;
          obj.width = 15;
          obj.borderWidth = 1;
          obj.style = {
            fill: '#6BA5D7',
            strokeWidth: 2,
            strokeColor: '#6BA5D7'
          };
          return obj;
        },
      }
    }
    mounted: function() {
      let diagramInstance: Diagram;
      let diagramObj: any = document.getElementById("diagram");
      diagramInstance = diagramObj.ej2_instances[0];
      let connectors: ConnectorModel = {
        id: 'connector1',
        type: 'Straight',
        segments: [{ type: "polyline" }]
      }
      diagramInstance.drawingObject = connectors;
    }
  }
}
```

```

        //To draw an object once, activate draw once
        diagramInstance.tool = DiagramTools.DrawOnce | DiagramTools.ZoomPan;
        diagramInstance.dataBind();
    }
}
</script>
<style>
    @import "../../node_modules/@syncfusion/ej2-vue-
    diagrams/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/diagram/Tools/connector-cs2" %}

## Events

[elementDraw](#) event is triggered when node or connector is drawn using drawing tool.

## APP.VUE

```

<template>
    <div id="app">
        <ejs-diagram id="diagram" :width='width' :height='height'
        :getNodeDefaults='getNodeDefaults' :elementDraw="elementDraw"></ejs-diagram>
    </div>
</template>
<script>
    import Vue from 'vue';
    import { DiagramPlugin, DiagramTools, BasicShapeModel, ConnectorModel,
    IElementDrawEventArgs } from '@syncfusion/ej2-vue-diagrams';
    Vue.use(DiagramPlugin);
    export default {
        name: 'app'
        data() {
            return {
                width: "100%",
                height: "350px",
                getNodeDefaults: (obj) => {
                    obj.height = 15;
                    obj.width = 15;
                    obj.borderWidth = 1;
                    obj.style = {
                        fill: '#6BA5D7',
                        strokeWidth: 2,
                        strokeColor: '#6BA5D7'
                    };
                },
                return obj;
            },
            elementDraw: (args) => {
                {
                    console.log("Event Triggered")
                }
            }
        }
        mounted: function() {
            let diagramInstance: Diagram;
            let diagramObj: any = document.getElementById("diagram");
            diagramInstance = diagramObj.ej2_instances[0];

```

```

    let connectors: ConnectorModel = {
      id: 'connector1',
      type: 'Straight',
      segments: [{ type: "Straight" }]
    }
    diagramInstance.drawingObject = connectors;
    //To draw an object once, activate draw once
    diagramInstance.tool = DiagramTools.DrawOnce;
    diagramInstance.dataBind();
  }
}
</script>
<style>
  @import "../node_modules/@syncfusion/ej2-vue-
  diagrams/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/diagram/Tools/connector-cs3" %}

### Freehand Drawing

Diagram has support for free-hand drawing to draw anything on the diagram page independently. Free-hand drawing will be enabled by using the drawingObject property and setting its value to Freehand.

The following code illustrates how to draw a freehand drawing.

### APP.VUE

```

<template>
  <div id="app">
    <ejs-diagram id="diagram" :width='width' :height='height'
    :getNodeDefaults='getNodeDefaults'></ejs-diagram>
  </div>
</template>
<script>
  import Vue from 'vue';
  import { DiagramPlugin, DiagramTools, BasicShapeModel, ConnectorModel }
  from '@syncfusion/ej2-vue-diagrams';
  Vue.use(DiagramPlugin);
  export default {
    name: 'app'
    data() {
      return {
        width: "100%",
        height: "350px",
        getNodeDefaults: (obj) => {
          obj.height = 15;
          obj.width = 15;
          obj.borderWidth = 1;
          obj.style = {
            fill: '#6BA5D7',
            strokeWidth: 2,
            strokeColor: '#6BA5D7'
          };
          return obj;
        },
      },
    }
  }

```

```

    }
    mounted: function() {
      let diagramInstance: Diagram;
      let diagramObj: any = document.getElementById("diagram");
      diagramInstance = diagramObj.ej2_instances[0];
      let connectors: ConnectorModel = {
        id: 'connector1',
        type: 'Freehand'
      }
      diagramInstance.drawingObject = connectors;
      //To draw an object once, activate draw once
      diagramInstance.tool = DiagramTools.DrawOnce;
      diagramInstance.dataBind();
    }
  }
</script>
<style>
  @import "../node_modules/@syncfusion/ej2-vue-
  diagrams/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/diagram/Tools/connector-cs4" %}

### Grid lines in Vue Diagram component

Gridlines are the pattern of lines drawn behind the diagram elements. It provides a visual guidance while dragging or arranging the objects on the diagram surface.

The model's [snapSettings](#) property is used to customize the gridlines and control the snapping behavior in the diagram.

#### Customize the gridlines visibility

The [snapSettings.snapConstraints](#) enables you to show/hide the gridlines. The following code example illustrates how to show or hide gridlines.

If you need to enable snapping, then inject snapping module into the diagram.

#### APP.VUE

```

<template>
  <div id="app">
    <ejs-diagram id="diagram" :width='width' :height='height'
    :snapSettings='snapSettings' ></ejs-diagram>
  </div>
</template>
<script>
  import Vue from 'vue';
  import {
    DiagramPlugin, SnapSettingsModel, SnapConstraints, Snapping, Diagram } from
    '@syncfusion/ej2-vue-diagrams';
  Vue.use(DiagramPlugin);
  Diagram.Inject(Snapping);
  let snapSettings: SnapSettingsModel = {
    // Display both Horizontal and Vertical gridlines
    constraints: SnapConstraints.ShowLines };
  export default {
    name: 'app'
  }

```

```

        data() {
            return {
                width: "100%",
                height: "350px",
                snapSettings: snapSettings
            }
        }
    }
</script>
<style>
    @import "../node_modules/@syncfusion/ej2-vue-
    diagrams/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/diagram/gridlines/gridlines-cs1" %}

### Appearance

The appearance of the gridlines can be customized by using a set of predefined properties.

- The [horizontalGridLines](#) and the [verticalGridLines](#) properties allow to customize the appearance of the horizontal and vertical gridlines respectively.
- The horizontal gridlines [lineColor](#) and [lineDashArray](#) properties are used to customizes the line color and line style of the horizontal gridlines.
- The vertical gridlines [lineColor](#) and [lineDashArray](#) properties are used to customizes the line color and line style of the vertical gridlines.

The following code example illustrates how to customize the appearance of gridlines.

### APP.VUE

```

<template>
    <div id="app">
        <ejs-diagram id="diagram" :width='width' :height='height'
        :snapSettings='snapSettings' ></ejs-diagram>
    </div>
</template>
<script>
    import Vue from 'vue';
    import {
        DiagramPlugin, SnapSettingsModel, SnapConstraints, Snapping, Diagram } from
        '@syncfusion/ej2-vue-diagrams';
    Vue.use(DiagramPlugin);
    Diagram.Inject(Snapping);
    let snapSettings: SnapSettingsModel = {
        // Define the Constraints for gridlines and snapping
        constraints: SnapConstraints.ShowLines,
        // Defines the horizontalGridlines for SnapSettings
        horizontalGridlines: {
            // Sets the line color of gridlines
            lineColor: 'blue',
            // Defines the lineDashArray of gridlines
            lineDashArray: '2 2'
        },
        // Defines the verticalGridlines for SnapSettings
        verticalGridlines: {

```



```

        lineColor: 'blue',
        lineDashArray: '2 2'
    });
    export default {
        name: 'app'
        data() {
            return {
                width: "100%",
                height: "350px",
                snapSettings:snapSettings
            }
        }
    }
</script>
<style>
    @import "../node_modules/@syncfusion/ej2-vue-
    diagrams/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/diagram/gridlines/gridlinesAppearance-cs1" %}

### Line Intervals

Thickness and the space between gridlines can be customized by using horizontal gridlines's [linesInterval](#) and vertical gridlines's [linesInterval](#) properties. In the lines interval collections, values at the odd places are referred as the thickness of lines and values at the even places are referred as the space between gridlines.

The following code example illustrates how to customize the thickness of lines and the line intervals.

### APP.VUE

```

<template>
    <div id="app">
        <ejs-diagram id="diagram" :width='width' :height='height'
        :snapSettings='snapSettings' ></ejs-diagram>
    </div>
</template>
<script>
    import Vue from 'vue';
    import {
    DiagramPlugin, SnapSettingsModel, SnapConstraints, Snapping, Diagram } from
    '@syncfusion/ej2-vue-diagrams';
    Vue.use(DiagramPlugin);
    Diagram.Inject(Snapping);
    let snapSettings: SnapSettingsModel = {
        constraints: SnapConstraints.ShowLines,
        horizontalGridlines: {
            // Sets the lineIntervals of Gridlines
            lineIntervals: [1.25, 14, 0.25, 15, 0.25, 15, 0.25, 15, 0.25, 15],
            lineColor: 'blue',
            lineDashArray: '2 2'
        },
        verticalGridlines: {
            // Sets the lineIntervals of Gridlines
            lineIntervals: [1.25, 14, 0.25, 15, 0.25, 15, 0.25, 15, 0.25, 15],
            lineColor: 'blue',

```

```

        lineDashArray: '2 2'
    });
    export default {
      name: 'app'
      data() {
        return {
          width: "100%",
          height: "350px",
          snapSettings:snapSettings
        }
      }
    }
  </script>
  <style>
    @import "../node_modules/@syncfusion/ej2-vue-
    diagrams/styles/material.css";
  </style>

```

{% previewsample "page.domainurl/code-snippet/diagram/gridlines/gridlineIntervals-cs1" %}

## Snapping

### Snap To Lines

This feature allows the diagram objects to snap to the nearest intersection of gridlines while being dragged or resized. This feature enables easier alignment during layout or design.

Snapping to gridlines can be enabled/disabled with the [snapSettings.snapConstraints](#). The following code example illustrates how to enable/disable the snapping to gridlines.

### APP.VUE

```

<template>
  <div id="app">
    <ejs-diagram id="diagram" :width='width' :height='height'
    :nodes='nodes' :snapSettings='snapSettings' ></ejs-diagram>
  </div>
</template>
<script>
  import Vue from 'vue';
  import {
    DiagramPlugin, SnapSettingsModel, SnapConstraints, Snapping, Diagram } from
    '@syncfusion/ej2-vue-diagrams';
  Vue.use(DiagramPlugin);
  Diagram.Inject(Snapping);
  let nodes = [{
    id: 'node1',
    style:{fill:'#6BA5D7',strokeColor:'#6BA5D7'},
    width: 100,
    height: 100,
    offsetX: 100,
    offsetY: 100,
  }]
  let snapSettings: SnapSettingsModel = {
    // Enables the object to snap with both horizontal and Vertical
    gridlines
    constraints: SnapConstraints.SnapToLines | SnapConstraints.ShowLines
  }

```

```

export default {
  name: 'app'
  data() {
    return {
      width: "100%",
      height: "350px",
      nodes: nodes,
      snapSettings: snapSettings
    }
  }
}
</script>
<style>
  @import "../node_modules/@syncfusion/ej2-vue-
  diagrams/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/diagram/gridlines/snapping-cs1" %}

### Customization of Snap Intervals

By default, the objects are snapped towards the nearest gridline. The gridline or position towards where the diagram object snaps can be customized with the horizontal gridlines's [snapInterval](#) and the vertical gridlines's [snapInterval](#) properties.

### APP.VUE

```

<template>
  <div id="app">
    <ejs-diagram id="diagram" :width='width' :height='height'
    :nodes='nodes'
    :snapSettings='snapSettings' ></ejs-diagram>
  </div>
</template>
<script>
  import Vue from 'vue';
  import {
    DiagramPlugin, SnapSettingsModel, SnapConstraints, Snapping, Diagram } from
    '@syncfusion/ej2-vue-diagrams';
  Vue.use(DiagramPlugin);
  Diagram.Inject(Snapping);
  let nodes = [{
    id: 'node1',
    width: 100,
    style: { fill: '#6BA5D7', strokeColor: '#6BA5D7' },
    height: 100,
    offsetX: 100,
    offsetY: 100
  }]
  let snapSettings: SnapSettingsModel = {
    horizontalGridlines: {
      // Defines the snap interval for object
      snapIntervals: [10]
    },
    verticalGridlines: {
      snapIntervals: [10]
    }
  },

```

```

    constraints: SnapConstraints.All
  }
  export default {
    name: 'app'
    data() {
      return {
        width: "100%",
        height: "350px",
        nodes: nodes,
        snapSettings:snapSettings
      }
    }
  }
</script>
<style>
  @import "../node_modules/@syncfusion/ej2-vue-
  diagrams/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/diagram/gridlines/snapintervals-cs1" %}

### Snap To Objects

The snap-to-object provides visual cues to assist with aligning and spacing Diagram elements. A node can be snapped with its neighboring objects based on certain alignments. Such alignments are visually represented as smart guides.

The [snapObjectDistance](#) property allows you to define minimum distance between the selected object and the nearest object.

The [snapAngle](#) property allows you to define the snap angle by which the object needs to be rotated.

[snapConstraints](#) property allow you to enable or disable the certain features of the snapping , refer to [snapConstraints](#)

The [snapLineColor](#) property allows you to define the color of the snapline.

### APP.VUE

```

<template>
  <div id="app">
    <ejs-diagram id="diagram" :width='width' :height='height'
    :nodes='nodes'
    :snapSettings='snapSettings' ></ejs-diagram>
  </div>
</template>
<script>
  import Vue from 'vue';
  import {
  DiagramPlugin, SnapSettingsModel, SnapConstraints, Snapping, Diagram } from
  '@syncfusion/ej2-vue-diagrams';
  Vue.use(DiagramPlugin);
  Diagram.Inject(Snapping);
  let nodes = [{
    id: 'node1',
    style:{fill:'#6BA5D7',strokeColor:'#6BA5D7'},
    width: 100,
    height: 100,

```

```

        offsetX: 100,
        offsetY: 100
      },
      {
        id: 'node2',
        style: { fill: '#6BA5D7', strokeColor: '#6BA5D7' },
        width: 100,
        height: 100,
        offsetX: 300,
        offsetY: 100
      }
    ]
  }
  let snapSettings: SnapSettingsModel = {
    // Enable snap to object constraint
    constraints: SnapConstraints.SnapToObject | SnapConstraints.ShowLines,
    // Sets the Snap object distance
    snapObjectDistance: 10,
    // Snap Angle for object
    snapAngle: 10,
    // Set the Snapline color
    snapLineColor: 'red'
  }
  export default {
    name: 'app'
    data() {
      return {
        width: "100%",
        height: "350px",
        snapSettings: snapSettings,
        nodes: nodes,
      }
    }
  }
</script>
<style>
  @import "../../../node_modules/@syncfusion/ej2-vue-diagrams/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/diagram/gridlines/snapobjects-cs1" %}

## Page settings in Vue Diagram component

Page settings enable to customize the appearance, width, and height of the diagram page.

### Page size and appearance

- The size and appearance of the diagram pages can be customized with the page settings property.
- The [width](#) and [height](#) properties of page settings define the size of the page and based on the size, the [orientation](#) will be set for the page. In addition to that, the appearance of the page can be customized with [source](#) and set of appearance specific properties.
- The [color](#) property is used to customize the background color and border color of the page.
- The [margin](#) property is used to define the page margin.
- To explore those properties, refer to [Page Settings](#).

**APP.VUE**

```

<template>
  <div id="app">
    <ejs-diagram id="diagram" :width='width' :height='height'
:nodes='nodes' :getNodeDefaults='getNodeDefaults'
:connectors='connectors'
:pageSettings='pageSettings' ></ejs-diagram>
  </div>
</template>
<script>
  import Vue from 'vue';
  import { DiagramPlugin, PageSettings } from '@syncfusion/ej2-vue-
diagrams';
  Vue.use(DiagramPlugin);
  let connectors = [{
    id: 'connector1',
    style: { strokeColor: '#6BA5D7', fill: '#6BA5D7', strokeWidth: 2 },
    targetDecorator: { style: { fill: '#6BA5D7', strokeColor: '#6BA5D7'
} },
    sourceID: 'node1',
    targetID: 'node2',
  }]
  let nodes = [{
    id: 'node1',
    width: 100,
    height: 100,
    offsetX: 100,
    offsetY: 100,
    annotations: [{
      content: 'Node1'
    }]
  },
  {
    id: 'node2',
    width: 100,
    height: 100,
    offsetX: 300,
    offsetY: 350,
    annotations: [{
      content: 'Node3'
    }]
  }]
  export default {
    name: 'app'
    data() {
      return {
        width: "100%",
        height: "350px",
        nodes: nodes,
        getNodeDefaults: (node) => {
          node.height = 100;
          node.width = 100;
          node.style.fill = '#6BA5D7';
          node.style.strokeColor = 'white';
          return node;
        },

```

```

        connectors: connectors,
        pageSettings: {
          // Sets the PageOrientation for the diagram to page
          orientation: 'Landscape',
          // Sets the Page Break for diagram
          showPageBreaks: true,
          // Defines the background color and image of diagram
          background: {
            color: 'grey'
          },
          // Sets the width for the Page
          width: 300,
          // Sets the height for the Page
          height: 300,
          // Sets the space to be left between an annotation and its
          parent node/connector
          margin: {
            left: 10,
            top: 10,
            bottom: 10
          },
        },
      },
    },
  ],
},
</script>
<style>
  @import "../node_modules/@syncfusion/ej2-vue-
  diagrams/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/diagram/page-settings/pagesettings-cs1" %}

### Set background image

Stretch and align the background image anywhere over the diagram area. The [source](#) property of [background](#) allows you to set the path of the image. The [scale](#) and the [align](#) properties help to stretch/align the background images.

The following code illustrates how to stretch and align the background image.

### APP.VUE

```

<template>
  <div id="app">
    <ejs-diagram id="diagram" :width='width' :height='height'
      :pageSettings='pageSettings' ></ejs-diagram>
  </div>
</template>
<script>
  import Vue from 'vue';
  import { DiagramPlugin, PageSettings } from '@syncfusion/ej2-vue-
  diagrams';
  Vue.use(DiagramPlugin);
  export default {
    name: 'app'
    data() {

```

```

        return {
            width: "100%",
            height: "350px",
            pageSettings: {
                orientation: 'Landscape',
                showPageBreaks: true,
                // Defines the background Image source
                background: {
                    source:
'https://www.w3schools.com/images/w3schools_green.jpg',
                    // Defines the scale values for the background image
                    scale: 'Meet',
                    // Defines the align values for the background image
                    align: 'XMinYMin'
                },
                width: 300,
                height: 300,
                margin: {
                    left: 10,
                    top: 10,
                    bottom: 10
                },
            },
        }
    }
}
</script>
<style>
    @import "../node_modules/@syncfusion/ej2-vue-
diagrams/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/diagram/page-settings/BGImage-cs1" %}

### Multiple page and page breaks

When multiple page is enabled, the size of the page dynamically increases or decreases in multiples of page width and height and completely fits diagram within the page boundaries. Page breaks is used as a visual guide to see how pages are split into multiple pages.

The [multiplePage](#) and [showPageBreak](#) properties of page settings allow you to enable/disable multiple pages and page breaks respectively.

The following code illustrates how to enable multiple page and page break lines.

### APP.VUE

```

<template>
    <div id="app">
        <ejs-diagram id="diagram" :width='width' :height='height'
:nodes='nodes' :getNodeDefaults='getNodeDefaults'
:connectors='connectors'
:pageSettings='pageSettings' ></ejs-diagram>
    </div>
</template>
<script>
    import Vue from 'vue';

```



```

import { DiagramPlugin, PageSettings } from '@syncfusion/ej2-vue-
diagrams';
Vue.use(DiagramPlugin);
let connectors = [{
  id: 'connector1',
  style: { strokeColor: '#6BA5D7', fill: '#6BA5D7', strokeWidth: 2 },
  targetDecorator: { style: { fill: '#6BA5D7', strokeColor: '#6BA5D7'
} }},
  sourceID: 'node1',
  targetID: 'node2',
}]
let nodes = [{
  id: 'node1',
  width: 100,
  height: 100,
  offsetX: 100,
  offsetY: 100,
  annotations: [{
    content: 'Node1'
  }]
},
{
  id: 'node2',
  width: 100,
  height: 100,
  offsetX: 300,
  offsetY: 350,
  annotations: [{
    content: 'Node3'
  }]
}]
export default {
  name: 'app'
  data() {
    return {
      width: "100%",
      height: "350px",
      nodes: nodes,
      getNodeDefaults: (node) => {
        node.height = 100;
        node.width = 100;
        node.style.fill = '#6BA5D7';
        node.style.strokeColor = 'white';
        return node;
      },
      connectors: connectors,
      pageSettings: {
        orientation: 'Landscape',
        // Sets the Multiple page for diagram
        multiplePage: true,
        // Sets the Page Break for diagram
        showPageBreaks: true,
        width: 300,
        height: 300,
        margin: {
          left: 10,
          top: 10,

```

```

        bottom: 10
      },
    }
  }
}
</script>
<style>
  @import "../node_modules/@syncfusion/ej2-vue-
  diagrams/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/diagram/page-settings/multiplepage-cs1" %}

### Boundary constraints

The diagram provides support to restrict/customize the interactive region, out of which the elements cannot be dragged, resized, or rotated. The [boundaryConstraints](#) property of page settings allows you to customize the interactive region. To explore the boundary constraints, refer to [Boundary Constraints](#).

The following code example illustrates how to define boundary constraints with respect to the page.

### APP.VUE

```

<template>
  <div id="app">
    <ejs-diagram id="diagram" :width='width' :height='height'
    :nodes='nodes' :getNodeDefaults='getNodeDefaults'
    :connectors='connectors'
    :pageSettings='pageSettings' ></ejs-diagram>
  </div>
</template>
<script>
  import Vue from 'vue';
  import { DiagramPlugin, PageSettings } from '@syncfusion/ej2-vue-
  diagrams';
  Vue.use(DiagramPlugin);
  let connectors = [{
    id: 'connector1',
    style: { strokeColor: '#6BA5D7', fill: '#6BA5D7', strokeWidth: 2 },
    targetDecorator: { style: { fill: '#6BA5D7', strokeColor: '#6BA5D7'
    } },
    sourcePoint: {
      x: 300,
      y: 100
    },
    targetPoint: {
      x: 400,
      y: 100
    }
  } ],
  let nodes = [
    {
      id: 'node1',
      width: 150,
      height: 100,
      offsetX: 100,

```

```

        offsetY: 100,
      },
      {
        id: 'node2',
        width: 80,
        height: 130,
        offsetX: 200,
        offsetY: 200,
      }
    ]
  },
  export default {
    name: 'app'
    data() {
      return {
        width: "100%",
        height: "350px",
        nodes: nodes,
        getNodeDefaults: (node) => {
          node.height = 100;
          node.width = 100;
          node.style.fill = '#6BA5D7';
          node.style.strokeColor = 'white';
          return node;
        },
        connectors: connectors,
        pageSettings: {
          // Sets the BoundaryConstraints to page
          boundaryConstraints: 'Page',
          background: {
            color: 'grey'
          },
          width: 400,
          height: 400,
          showPageBreaks: true,
        }
      }
    }
  }
</script>
<style>
  @import "../node_modules/@syncfusion/ej2-vue-
  diagrams/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/diagram/page-settings/boundaryconstraints-cs1" %}

### Scroll settings in Vue Diagram component

The diagram can be scrolled by using the vertical and horizontal scrollbars. In addition to the scrollbars, mousewheel can be used to scroll the diagram.

Diagram's [scrollSettings](#) enable you to read the current scroll status, view port size, current zoom, and zoom factor. It also allows you to scroll the diagram programmatically.

### Get current scroll status

Scroll settings allow you to read the scroll status, [viewPortWidth](#), [viewPortHeight](#), and [currentZoom](#) with a set of properties. To explore those properties, see [Scroll Settings](#).

### Define scroll status

Diagram allows you to pan the diagram before loading, so that any desired region of a large diagram is made to view. You can programmatically pan the diagram with the [horizontalOffset](#) and [verticalOffset](#) properties of scroll settings. The following code illustrates how to set pan the diagram programmatically.

In the following example, the vertical scroll bar is scrolled down by 50px and horizontal scroll bar is scrolled to right by 100px.

#### APP.VUE

```
<template>
  <div id="app">
    <ejs-diagram id="diagram" :width='width' :height='height'
scrollSettings='scrollSettings'></ejs-diagram>
  </div>
</template>
<script>
  import Vue from 'vue';
  import { DiagramPlugin } from '@syncfusion/ej2-vue-diagrams';
  Vue.use(DiagramPlugin);
  export default {
    name: 'app'
    data() {
      return {
        width: "100%",
        height: "350px",
        scrollSettings: {
          horizontalOffset: 100,
          verticalOffset: 50
        }
      }
    }
  }
</script>
<style>
  @import "../../node_modules/@syncfusion/ej2-vue-
diagrams/styles/material.css";
</style>
```

{% previewsample "page.domainurl/code-snippet/diagram/scroll-settings/scroll-cs1" %}

### Update scroll status

You can programmatically change the scroll offsets at runtime by using the client-side method `update`. The following code illustrates how to change the scroll offsets and zoom factor at runtime.

#### APP.VUE

```
<template>
  <div id="app">
    <ejs-diagram id="diagram" :width='width' :height='height'
scrollSettings='scrollSettings'></ejs-diagram>
  </div>
```

```

</template>
<script>
  import Vue from 'vue';
  import { DiagramPlugin } from '@syncfusion/ej2-vue-diagrams';
  Vue.use(DiagramPlugin);
  export default {
    name: 'app'
    data() {
      return {
        width: "100%",
        height: "350px",
      }
    }
    mounted: function() {
      let diagramInstance: Diagram;
      let diagramObj: any = document.getElementById("diagram");
      diagramInstance = diagramObj.ej2_instances[0];
      //Updates scroll settings
      diagramInstance.scrollSettings.horizontalOffset = 200;
      diagramInstance.scrollSettings.verticalOffset = 30
      diagramInstance.dataBind();
    }
  }
</script>
<style>
  @import "../../node_modules/@syncfusion/ej2-vue-diagrams/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/diagram/scroll-settings/scrollSettings-cs1" %}

### AutoScroll

Autoscroll feature automatically scrolls the diagram, whenever the node or connector is moved beyond the boundary of the diagram. So that, it is always visible during dragging, resizing, and multiple selection operations. Autoscroll is automatically triggered when any one of the following is done towards the edges of the diagram.

- Node dragging, resizing
- Connection editing
- Rubber band selection
- Label dragging

The diagram client-side event [ScrollChange](#) gets triggered when the autoscroll (scrollbars) is changed and you can do your own customization in this event.

The autoscroll behavior in your diagram can be enabled/disabled by using the [canAutoScroll](#) property of the diagram. The following code example illustrates how to set autoscroll.

### APP.VUE

```

<template>
  <div id="app">

```

```

    <ejs-diagram id="diagram" :width='width' :height='height'
:nodes='nodes' :connectors= 'connectors' :scrollSettings='scrollSettings'
:getNodeDefaults='getNodeDefaults'></ejs-diagram>
  </div>
</template>
<script>
  import Vue from 'vue';
  import { DiagramPlugin } from '@syncfusion/ej2-vue-diagrams';
  Vue.use(DiagramPlugin);
  let nodes = [{
    id: 'Start',
    width: 140,
    height: 50,
    offsetX: 300,
    offsetY: 50,
    annotations: [{
      id: 'label1',
      content: 'Start'
    }],
    shape: {
      type: 'Flow',
      shape: 'Terminator'
    }
  }];
  let connectors: ConnectorModel[] = [{
    id: 'connector1', sourcePoint: { x: 300, y: 100 }, targetPoint: { x:
450, y: 200 },
    style: {
      strokeColor: '#6BA5D7',
      strokeWidth: 2
    },
    targetDecorator: {
      style: {
        fill: '#6BA5D7',
        strokeColor: '#6BA5D7'
      }
    }
  }];
  export default {
    name: 'app'
    data() {
      return {
        width: "100%",
        height: "350px",
        nodes: nodes,
        connectors: connectors,
        scrollSettings: {
          canAutoScroll: true,
          scrollLimit: 'Infinity',
        },
        getNodeDefaults: (node) => {
          node.height = 100;
          node.width = 100;
          node.style.fill = '#6BA5D7';
          node.style.strokeColor = 'white';
          return node;
        }
      }
    }
  }

```

```

    }
  }
}
</script>
<style>
  @import "../node_modules/@syncfusion/ej2-vue-
diagrams/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/diagram/scroll-settings/scrollSettings-cs2" %}

### Autoscroll border

The autoscroll border is used to specify the maximum distance between the object and diagram edge to trigger autoscroll. The default value is set as 15 for all sides (left, right, top, and bottom) and it can be changed by using the [autoScrollBorder](#) property of page settings. The following code example illustrates how to set autoscroll border.

### APP.VUE

```

<template>
  <div id="app">
    <ejs-diagram id="diagram" :width='width' :height='height'
:nodes='nodes' :scrollSettings='scrollSettings'
:getNodeDefaults='getNodeDefaults'></ejs-diagram>
  </div>
</template>
<script>
  import Vue from 'vue';
  import { DiagramPlugin } from '@syncfusion/ej2-vue-diagrams';
  Vue.use(DiagramPlugin);
  let nodes = [{
    id: 'Start',
    width: 140,
    height: 50,
    offsetX: 300,
    offsetY: 50,
    annotations: [{
      id: 'label1',
      content: 'Start'
    }],
    shape: {
      type: 'Flow',
      shape: 'Terminator'
    }
  }]
  export default {
    name: 'app'
    data() {
      return {
        width: "100%",
        height: "350px",
        nodes: nodes,
        scrollSettings: {
          canAutoScroll: true,
          scrollLimit: 'Infinity',
          autoScrollBorder: {

```

```

        left: 100,
        right: 100,
        top: 100,
        bottom: 100
      },
    },
    getNodeDefaults: (node) => {
      node.height = 100;
      node.width = 100;
      node.style.fill = '#6BA5D7';
      node.style.strokeColor = 'white';
      return node;
    }
  }
}
</script>
<style>
  @import "../node_modules/@syncfusion/ej2-vue-
  diagrams/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/diagram/scroll-settings/autoScroll-cs1" %}

### Scroll limit

The scroll limit allows you to define the scrollable region of the diagram. It includes the following options:

- Allows to scroll in all directions without any restriction.
- Allows to scroll within the diagram content.
- Allows to scroll within the specified scrollable area.
- The [scrollLimit](#) property of scroll settings helps to limit the scrolling.

The scrollSettings [scrollableArea](#) allow to extend the scrollable region that is based on the scroll limit.

The following code example illustrates how to specify the scroll limit.

### APP.VUE

```

<template>
  <div id="app">
    <ejs-diagram id="diagram" :width='width' :height='height'
    :nodes='nodes' :scrollSettings='scrollSettings'
    :getNodeDefaults='getNodeDefaults'></ejs-diagram>
  </div>
</template>
<script>
  import Vue from 'vue';
  import { DiagramPlugin } from '@syncfusion/ej2-vue-diagrams';
  Vue.use(DiagramPlugin);
  let nodes = [{
    id: 'Start',
    width: 140,
    height: 50,
    offsetX: 300,

```



```

        offsetY: 50,
        annotations: [{
            id: 'label1',
            content: 'Start'
        }],
        shape: {
            type: 'Flow',
            shape: 'Terminator'
        }
    }
}
export default {
    name: 'app'
    data() {
        return {
            width: "100%",
            height: "350px",
            nodes: nodes,
            // set the autoScrollBorder
            scrollSettings: {
                canAutoScroll: true,
                //Sets the scroll limit
                scrollLimit: 'infinity'
            },
            getNodeDefaults: (node) => {
                node.height = 100;
                node.width = 100;
                node.style.fill = '#6BA5D7';
                node.style.strokeColor = 'white';
                return node;
            }
        }
    }
}
</script>
<style>
    @import "../../node_modules/@syncfusion/ej2-vue-
    diagrams/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/diagram/scroll-settings/scrollLimit-cs1" %}

### Scroll padding

[padding](#) property of scroll settings allows you to extend the scrollable region that is based on the scroll limit.

The following code example illustrates how to set scroll padding to diagram region.

### APP.VUE

```

<template>
    <div id="app">
        <ejs-diagram id="diagram" :width='width' :height='height'
        :nodes='nodes' :scrollSettings='scrollSettings' ></ejs-diagram>
    </div>
</template>
<script>

```

```

import Vue from 'vue';
import { DiagramPlugin } from '@syncfusion/ej2-vue-diagrams';
Vue.use(DiagramPlugin);
let nodes = [{
  id: 'Start',
  width: 100, height: 100,
  offsetX: 350, offsetY: 350,
  shape: {
    type: 'Flow',
    shape: 'Terminator'
  }
}]
export default {
  name: 'app'
  data() {
    return {
      width: "100%",
      height: "350px",
      nodes: nodes,
      // set the autoScrollBorder
      scrollSettings: {
        canAutoScroll: true,
        scrollLimit: 'Infinity',
        //Sets the scroll limit
        padding: { right: 50, bottom: 50 }
      },
    }
  }
}
</script>
<style>
  @import "../../node_modules/@syncfusion/ej2-vue-diagrams/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/diagram/scroll-settings/scrollLimit-cs2" %}

### Scrollable Area

Scrolling beyond any particular rectangular area can be restricted by using the [scrollableArea](#) property of scroll settings. To restrict scrolling beyond any custom region, set the [scrollLimit](#) as "limited". The following code example illustrates how to customize scrollable area.

### APP.VUE

```

<template>
  <div id="app">
    <ejs-diagram id="diagram" :width='width' :height='height'
      :nodes='nodes' :scrollSettings='scrollSettings'
      :getNodeDefaults='getNodeDefaults'></ejs-diagram>
    </div>
  </template>
<script>
  import Vue from 'vue';
  import { DiagramPlugin } from '@syncfusion/ej2-vue-diagrams';
  Vue.use(DiagramPlugin);
  let nodes = [{

```

```

    id: 'Start',
    width: 140,
    height: 50,
    offsetX: 300,
    offsetY: 50,
    annotations: [{
      id: 'label1',
      content: 'Start'
    }],
    shape: {
      type: 'Flow',
      shape: 'Terminator'
    }
  ]
}
export default {
  name: 'app'
  data() {
    return {
      width: "100%",
      height: "350px",
      nodes: nodes,
      // set the autoScrollBorder
      scrollSettings: {
        canAutoScroll: true,
        //Sets the scroll limit
        scrollLimit: 'infinity',
        //Sets the scrollable Area
        scrollableArea: {
          x: 0,
          y: 0,
          width: 500,
          height: 500
        }
      },
      getNodeDefaults: (node) => {
        node.height = 100;
        node.width = 100;
        node.style.fill = '#6BA5D7';
        node.style.strokeColor = 'white';
        return node;
      }
    }
  }
}
</script>
<style>
  @import "../node_modules/@syncfusion/ej2-vue-
  diagrams/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/diagram/scroll-settings/scrollArea-cs1" %}

#### UpdateViewPort

The [updateViewPort](#) method is used to update the diagram page and view size at runtime.

## Data binding in Vue Diagram component

- Diagram can be populated with the **nodes** and **connectors** based on the information provided from an external data source.
- Diagram exposes its specific data-related properties allowing you to specify the data source fields from where the node information has to be retrieved from.
- The [dataManager](#) property is used to define the data source either as a collection of objects or as an instance of **DataManager** that needs to be populated in the diagram.
- The [ID](#) property is used to define the unique field of each JSON data.
- The [parentId](#) property is used to defines the parent field which builds the relationship between ID and parent field.
- The [root](#) property is used to define the root node for the diagram populated from the data source.
- To explore those properties, see [DataSourceSettings](#).
- Diagram supports two types of data binding. They are:
  1. Local data
  2. Remote data

### Local data

Diagram can be populated based on the user defined JSON data (Local Data) by mapping the relevant data source fields.

To map the user defined JSON data with diagram, configure the fields of [dataSourceSettings](#). The following code example illustrates how to bind local data with the diagram.

### APP.VUE

```
<template>
  <div id="app">
    <ejs-diagram id="diagram" :width='width' :height='height'
    :tool='tool' :getNodeDefaults='getNodeDefaults' :snapSettings='snapSettings'
    :getConnectorDefaults='getConnectorDefaults' :layout='layout'
    :dataSourceSettings='dataSourceSettings' ></ejs-diagram>
  </div>
</template>
<script>
  import Vue from 'vue';
  import { DiagramPlugin, Diagram, HierarchicalTree,
  DataBinding, DiagramTools, NodeModel } from '@syncfusion/ej2-vue-diagrams';
  import { DataManager } from "@syncfusion/ej2-data";
  Vue.use(DiagramPlugin);
  Diagram.Inject(DataBinding, HierarchicalTree);
  export let species = [{
    'Name': 'Species',
    'fillColor': '#3DD94A'
  },
  {
    'Name': 'Plants',
    'Category': 'Species'
  },
  {
    'Name': 'Fungi',
    'Category': 'Species'
  }
];
```

```
},
{
  'Name': 'Lichens',
  'Category': 'Species'
},
{
  'Name': 'Animals',
  'Category': 'Species'
},
{
  'Name': 'Mosses',
  'Category': 'Plants'
},
{
  'Name': 'Ferns',
  'Category': 'Plants'
},
{
  'Name': 'Gymnosperms',
  'Category': 'Plants'
},
{
  'Name': 'Dicotyledens',
  'Category': 'Plants'
},
{
  'Name': 'Monocotyledens',
  'Category': 'Plants'
},
{
  'Name': 'Invertebrates',
  'Category': 'Animals'
},
{
  'Name': 'Vertebrates',
  'Category': 'Animals'
},
{
  'Name': 'Insects',
  'Category': 'Invertebrates'
},
{
  'Name': 'Molluscs',
  'Category': 'Invertebrates'
},
{
  'Name': 'Crustaceans',
  'Category': 'Invertebrates'
},
{
  'Name': 'Others',
  'Category': 'Invertebrates'
},
{
  'Name': 'Fish',
  'Category': 'Vertebrates'
},
},
```

```
{
  'Name': 'Amphibians',
  'Category': 'Vertebrates'
},
{
  'Name': 'Reptiles',
  'Category': 'Vertebrates'
},
{
  'Name': 'Birds',
  'Category': 'Vertebrates'
},
{
  'Name': 'Mammals',
  'Category': 'Vertebrates'
}
]];
export default {
  name: 'app',
  data() {
    return {
      width: "100%",
      height: "350px",
      getNodeDefaults: (obj) => {
        //Initialize shape
        obj.shape = {
          type: 'Basic',
          shape: 'Rectangle'
        };
        obj.style = {
          strokeWidth: 1
        };
        obj.width = 95;
        obj.height = 30;
      },
      getConnectorDefaults: (connector) => {
        connector.type = 'Orthogonal';
        connector.style.strokeColor = '#4d4d4d';
        connector.targetDecorator.shape = 'None';
      },
      snapSettings: {
        constraints: 0
      },
      tool: DiagramTools.ZoomPan,
      layout: {
        type: 'HierarchicalTree',
        horizontalSpacing: 15,
        verticalSpacing: 50,
        margin: {
          top: 10,
          left: 10,
          right: 10,
          bottom: 0
        }
      },
      dataSourceSettings: {
        id: 'Name',
        parentId: 'Category',

```

```

        dataManager: new DataManager(species),
        //binds the external data with node
        doBinding: (nodeModel: NodeModel, data: DataInfo,
diagram: Diagram) => {
            nodeModel.annotations = [{
                /* tslint:disable:no-string-literal */
                content: data['Name'],
                margin: {
                    top: 10,
                    left: 10,
                    right: 10,
                    bottom: 0
                },
                style: {
                    color: 'black'
                }
            }];
            /* tslint:disable:no-string-literal */
            nodeModel.style = {
                fill: '#ffeec7',
                strokeColor: '#f5d897',
                strokeWidth: 1
            };
        },
    },
    },
    },
    provide: {
        diagram: [DataBinding, HierarchicalTree]
    },
}
</script>
<style>
    @import
    "https://ej2.syncfusion.com/vue/documentation/../../node_modules/@syncfusion/ej
2-vue-diagrams/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/diagram/data-binding/LocalBinding-cs1" %}

### Remote data

You can bind the diagram with remote data by using `[dataManager]`.

It uses two different classes: `DataManager` for processing and `Query` for serving data. `DataManager` communicates with data source and `Query` generates data queries that are read by the [dataManager](#).

To learn more about data manager, refer to [Data Manager](#).

To bind remote data to the diagram, configure the fields of [dataSourceSettings](#). The following code illustrates how to bind remote data to the diagram.

### APP.VUE

```

<template>
  <div id="app">
    <ejs-diagram id="diagram" :width='width' :height='height'
    :tool='tool' :getNodeDefaults='getNodeDefaults' :snapSettings='snapSettings'

```

```

:getConnectorDefaults='getConnectorDefaults' :layout='layout'
:dataSourceSettings='dataSourceSettings' ></ejs-diagram>
</div>
</template>
<script>
  import Vue from 'vue';
  import
{DiagramPlugin,Diagram,NodeModel,Node,Connector,DataBinding,HierarchicalTree
,TreeInfo,DiagramTools} from '@syncfusion/ej2-vue-diagrams';
  import { DataManager,Query } from "@syncfusion/ej2-data";
  Vue.use(DiagramPlugin);
  Diagram.Inject(DataBinding, HierarchicalTree);
  export default {
    name: 'app',
    data() {
      return {
        width: "100%",
        height: "350px",
        layout: {
          type: 'HierarchicalTree', margin: { left: 0, right: 0, top: 100,
bottom: 0 },
          verticalSpacing: 40,
          getLayoutInfo: (node: Node, options: TreeInfo) => {
            if (options.level === 3) {
              node.style.fill = '#3c418d';
            }
            if (options.level === 2) {
              node.style.fill = '#108d8d';
              options.type = 'Center';
              options.orientation = 'Horizontal';
            }
            if (options.level === 1) {
              node.style.fill = '#822b86';
            }
          }
        },
        //Sets the default values of nodes
        getNodeDefaults: (obj: Node) => {
          obj.width = 80;
          obj.height = 40;
          //Initialize shape
          obj.shape = { type: 'Basic', shape: 'Rectangle' };
          obj.style = { fill: '#048785', strokeColor: 'Transparent' };
        },
        //Sets the default values of connector
        getConnectorDefaults: (connector: Connector) => {
          connector.type = 'Orthogonal';
          connector.style.strokeColor = '#048785';
          connector.targetDecorator.shape = 'None';
        },
        dataSourceSettings: {
          id: "Id",
          parentId: "ParentId",
          dataSource: new DataManager(
            {
              url:
                "https://services.syncfusion.com/vue/production/api/RemoteData",

```



```

        crossDomain: true
      },
    ),
    //binds the external data with node
    doBinding: (nodeModel, data, diagram) => {
      nodeModel.annotations = [
        {
          /* tslint:disable:no-string-literal */
          content: data["Label"],
          style: { color: "white" }
        }
      ];
    },
    //Disables all interactions except zoom/pan
    tool: DiagramTools.ZoomPan,
    snapSettings: { constraints: 0 }
  },
  provide: {
    diagram: [DataBinding, HierarchicalTree]
  },
}
</script>
<style>
  @import
  "https://ej2.syncfusion.com/vue/documentation/../../node_modules/@syncfusion/ej2-vue-diagrams/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/diagram/data-binding/RemoteBinding-cs1" %}

## CRUD

This feature allows you to read the data source and perform add or edit or delete the data in data source at runtime.

### Read DataSource

- This feature allows you to define the nodes and connectors collection in the data source and connectionDataSource respectively.
- You can set the data collection in the model's dataSourceSettings [dataManager](#) property. The nodes will be generated based on the data specified in the data source.
- You can set the connector collection in the model's dataSourceSettings [connectionDataSource](#) property.
- The dataSourceSettings connectionDataSource [dataManager](#) property is used to set the data source for the connection data source items.
- If you have a data (data will be set in the dataSource property) with parent relationship in the database and also defined the connector in the connectionDataSource simultaneously, then the connectors set in the connectionDataSource will be considered as a priority to render the connector.
- The dataSourceSettings [crudAction's read](#) property specifies the method, which is used to read the data source and its populate the nodes in the diagram.

- The connectionDataSource crudAction's [read](#) specifies the method, which is used to read the data source and its populates the connectors in the diagram.
- The dataSourceSettings's [id](#) and connectionDataSource's [id](#) properties are used to define the unique field of each JSON data.
- The connectionDataSource's [sourceID](#) and [targetID](#) properties are used to set the sourceID and targetID for connection data source item.
- The connectionDataSource's [sourcePointX](#), [sourcePointY](#), [targetPointX](#), and [targetPointY](#) properties are used to define the sourcePoint and targetPoint values for connector from data source.
- The dataSourceSettings crudAction's [customFields](#) property is used to maintain the additional information for nodes.
- Similarly, connectionDataSource's crudAction's [customFields](#) is used to maintain the additional information for connectors.

### How to perform Editing at runtime

- The dataSourceSettings crudAction object allows you to define the method, which is used to get the changes done in the data source defined for shapes from the client-side to the server-side.
- Similarly, the connectionDataSource crudAction object allows you to define the method, which is used to get the changes done in the data source defined for connectors from the client-side to the server-side.

### InsertData

- The dataSourceSettings crudAction's [create](#) property specifies the method, which is used to get the nodes added from the client-side to the server-side.
- The connectionDataSource crudAction's [create](#) specifies the method, which is used to get the connectors added from the client-side to the server-side.
- The following code example illustrates how to send the newly added or inserted data from the client to server-side.

```
`javascript
export default {
  name: 'app'
  data() {
    return {
      width: "100%",
      height: "350px",
      connectors: connectors,
      dataSourceSettings: {
        crudAction: {
          //Url which triggers the server side AddNodes method
```

```

create: 'https://ej2services.syncfusion.com/development/web-services/api/Crud/AddNodes',
},
connectionDataSource: {
  crudAction: {
    //Url which triggers the server side AddConnectors method
    create: 'https://ej2services.syncfusion.com/development/web-services/api/Crud/AddConnectors',
  }
}
}
}
}
}
}
mounted: function() {
  let diagramInstance: Diagram;
  let diagramObj: any = document.getElementById("diagram");
  diagramInstance = diagramObj.ej2_instances[0];
  //Sends the inserted nodes/connectors from client side to the server side through the URL which is
  //specified in server side.
  diagramInstance.insertData();
}
}
,

```

### UpdateData

- The dataSourceSettings crudAction's [update](#) property specifies the method, which is used to get the modified nodes from the client-side to the server-side.
- The connectionDataSource crudAction's [update](#) specifies the method, which is used to get the modified connectors from the client-side to the server-side.
- The following code example illustrates how to send the updated data from the client to the server side.

```

`javascript
export default {
  name: 'app'
  data() {
    return {
      width: "100%",

```

```

height: "350px",
connectors: connectors,
dataSourceSettings: {
  crudAction: {
    //Url which triggers the server side UpdateNodes method
    update: 'https://ej2services.syncfusion.com/development/web-services/api/Crud/UpdateNodes',
  },
  connectionDataSource: {
    crudAction: {
      //Url which triggers the server side UpdateConnectors method
      update: 'https://ej2services.syncfusion.com/development/web-services/api/Crud/UpdateConnectors',
    }
  }
}
}
}
}
}
}
}
mounted: function() {
  let diagramInstance: Diagram;
  let diagramObj: any = document.getElementById("diagram");
  diagramInstance = diagramObj.ej2_instances[0];
  //Sends the updated nodes/connectors from client side to the server side through the URL which is
  //specified in server side.
  diagramInstance.updateData();
}
}
`

```

### DeleteData

- The dataSourceSettings crudAction's [destroy](#) property specifies the method, which is used to get the deleted nodes from the client-side to the server-side.
- The connectionDataSource crudAction's [destroy](#) specifies the method, which is used to get the deleted connectors from the client-side to the server-side.

```

`javascript
export default {
  name: 'app'
}

```

```
data() {
  return {
    width: "100%",
    height: "350px",
    connectors: connectors,
    dataSourceSettings: {
      crudAction: {
        //Url which triggers the server side DeleteNodes method
        destroy: 'https://ej2services.syncfusion.com/development/web-services/api/Crud/DeleteNodes',
      },
      connectionDataSource: {
        crudAction: {
          //Url which triggers the server side DeleteConnectors method
          destroy: 'https://ej2services.syncfusion.com/development/web-services/api/Crud/DeleteConnectors',
        }
      }
    }
  }
},
mounted: function() {
  let diagramInstance: Diagram;
  let diagramObj: any = document.getElementById("diagram");
  diagramInstance = diagramObj.ej2_instances[0];
  //Sends the deleted nodes/connectors from client side to the server side through the URL which is
  //specified in server side.
  diagramInstance.removeData();
}
```

#### See Also

- [How to arrange the diagram nodes and connectors using varies layout](#)

## Automatic layout in Vue Diagram component

Diagram provides support to auto-arrange the nodes in the diagram area that is referred as **Layout**. It includes the following layout modes:

### Layout modes

- Hierarchical layout
- Organization chart
- Radial tree
- Symmetric layout
- Mind Map layout
- Complex hierarchical tree layout

### Hierarchical layout

The hierarchical tree layout arranges nodes in a tree-like structure, where the nodes in the hierarchical layout may have multiple parents. There is no need to specify the layout root. To arrange the nodes in a hierarchical structure, specify the layout [type](#) as hierarchical tree. The following example shows how to arrange the nodes in a hierarchical structure.

Note: If you want to use hierarchical tree layout in diagram, you need to inject HierarchicalTree in the diagram.

### APP.VUE

```
<template>
  <div id="app">
    <ejs-diagram id="diagram" :width='width' :height='height'
      :getNodeDefaults='getNodeDefaults'
      :getConnectorDefaults='getConnectorDefaults' :layout='layout'
      :dataSourceSettings='dataSourceSettings' ></ejs-diagram>
    </div>
  </template>
<script>
  import Vue from 'vue';
  import { DiagramPlugin, Diagram, HierarchicalTree, HierarchicalTree,
    DataBinding, DiagramTools, NodeModel } from '@syncfusion/ej2-vue-diagrams';
  import { DataManager, Query } from "@syncfusion/ej2-data";
  Diagram.Inject(DataBinding, HierarchicalTree);
  Vue.use(DiagramPlugin);
  export let data = [{
    Name: "Steve-Ceo"
  },
  {
    Name: "Kevin-Manager",
    ReportingPerson: "Steve-Ceo"
  },
  {
    Name: "Peter-Manager",
    ReportingPerson: "Steve-Ceo"
  },
  {
    Name: "John- Manager",
    ReportingPerson: "Peter-Manager"
  },
  ],
```

```

{
  Name: "Mary-CSE ",
  ReportingPerson: "Peter-Manager"
},
{
  Name: "Jim-CSE ",
  ReportingPerson: "Kevin-Manager"
},
{
  Name: "Martin-CSE",
  ReportingPerson: "Kevin-Manager"
}
]];
let items: DataManager = new DataManager(data as JSON[], new
Query().take(7));
export default {
  name: 'app',
  data() {
    return {
      width: "100%",
      height: "350px",
      //Uses layout to auto-arrange nodes on the Diagram page
      layout: {
        //Sets layout type
        type: 'HierarchicalTree'
      },
      dataSourceSettings: {
        id: 'Name',
        parentId: 'ReportingPerson',
        dataManager: items
      },
      getNodeDefaults: (obj) => {
        obj.shape = {
          type: 'Text',
          content: (obj.data as {
            Name: 'string'
          }).Name
        };
        obj.style = {
          fill: 'None',
          strokeColor: 'none',
          strokeWidth: 2,
          bold: true,
          color: 'white'
        };
        obj.borderColor = 'white';
        obj.width = 100;
        obj.height = 40;
        obj.backgroundColor = '#6BA5D7';
        obj.borderWidth = 1;
        (obj.shape as TextModel).margin = {
          left: 5,
          right: 5,
          top: 5,
          bottom: 5
        };
        return obj;
      },
    },
  },

```

```

        getConnectorDefaults: (connector) => {
            connector.style = {
                strokeColor: '#6BA5D7',
                strokeWidth: 2
            };
            connector.targetDecorator.style.fill = '#6BA5D7';

            connector.targetDecorator.style.strokeColor = '#6BA5D7';
            connector.type = 'Orthogonal';
            return connector;
        },
    },
    provide: {
        diagram: [DataBinding, HierarchicalTree]
    },
}
</script>
<style>
    @import
    "https://ej2.syncfusion.com/vue/documentation/../../node_modules/@syncfusion/ej2-vue-diagrams/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/diagram/automatic-layout/HierarchicalLayout-cs1" %}

### Radial tree layout

The radial tree layout arranges nodes on a virtual concentric circle around a root node. Sub-trees formed by the branching of child nodes are located radially around the child nodes. This arrangement result in an ever-expanding concentric arrangement with radial proximity to the root node indicating the node level in the hierarchy. The layout [root](#) property can be used to define the root node of the layout. When no root node is set, the algorithm automatically considers one of the diagram nodes as the root node.

To arrange nodes in a radial tree structure, set the [type](#) of the layout as **RadialTree**. The following code illustrates how to arrange the nodes in a radial tree structure.

Note: If you want to use radial tree layout in diagram, you need to inject DataBinding and RadialTree in the diagram.

### APP.VUE

```

<template>
    <div id="app">
        <ejs-diagram id="diagram" :width='width' :height='height'
        :getNodeDefaults='getNodeDefaults' :snapSettings='snapSettings'
        :getConnectorDefaults='getConnectorDefaults' :layout='layout'
        :dataSourceSettings='dataSourceSettings' ></ejs-diagram>
    </div>
</template>
<script>
    import Vue from 'vue';
    import { DiagramPlugin, Diagram, HierarchicalTree, RadialTree,
    DataBinding, DiagramTools, NodeModel } from '@syncfusion/ej2-vue-diagrams';
    import { DataManager, Query } from "@syncfusion/ej2-data";

```



```
Diagram.Inject(DataBinding, RadialTree);
Vue.use(DiagramPlugin);
export let data = [{
  "Id": 1,
  "Name": "Ana Trujillo",
  "Designation": "Project Manager",
  "RatingColor": "#68C2DE"
},
{
  "Id": 2,
  "Name": "Lino Rodri",
  "Designation": "Project Manager",
  "RatingColor": "#68C2DE",
  "ReportingPerson": 1
},
{
  "Id": 3,
  "Name": "Philip Cramer",
  "Designation": "Project Manager",
  "RatingColor": "#68C2DE",
  "ReportingPerson": 1
},
{
  "Id": 4,
  "Name": "Pedro Afonso",
  "Designation": "Project Manager",
  "RatingColor": "#68C2DE",
  "ReportingPerson": 1
},
{
  "Id": 5,
  "Name": "Anto Moreno",
  "Designation": "Project Lead",
  "RatingColor": "#93B85A",
  "ReportingPerson": 1
},
{
  "Id": 6,
  "Name": "Elizabeth Roel",
  "Designation": "Project Lead",
  "RatingColor": "#93B85A",
  "ReportingPerson": 1
},
{
  "Id": 7,
  "Name": "Aria Cruz",
  "Designation": "Project Lead",
  "RatingColor": "#93B85A",
  "ReportingPerson": 1
},
{
  "Id": 8,
  "Name": "Eduardo Roel",
  "Designation": "Project Lead",
  "RatingColor": "#93B85A",
  "ReportingPerson": 1
},
}
```

```

    {
      "Id": 9,
      "Name": "Howard Snyder",
      "Designation": "Project Lead",
      "RatingColor": "#68C2DE",
      "ReportingPerson": 1
    },
    {
      "Id": 10,
      "Name": "Daniel Tonini",
      "Designation": "Project Lead",
      "RatingColor": "#93B85A",
      "ReportingPerson": 1
    },
    {
      "Id": 11,
      "Name": "Nardo Batista",
      "Designation": "Project Lead",
      "RatingColor": "#68C2DE",
      "ReportingPerson": 1
    }
  ];
  let items: DataManager = new DataManager(data as JSON[], new
  Query().take(5));
  export default {
    name: 'app',
    data() {
      return {
        width: "100%",
        height: "590px",
        snapSettings: {
          constraints: 0
        },
        //Uses layout to auto-arrange nodes on the Diagram page
        layout: {
          //set the type as Radial Tree
          type: 'RadialTree',
          root: 'parent'
        },
        //Configures data source for Diagram
        dataSourceSettings: {
          id: 'Id',
          parentId: 'ReportingPerson',
          dataManager: items
        },
        //Sets the default properties for nodes and connectors
        getNodeDefaults: (obj: Node, diagram: Diagram) => {
          obj.height = 15;
          obj.width = 15;
          obj.borderWidth = 1;
          obj.style = {
            fill: '#6BA5D7',
            strokeWidth: 2,
            strokeColor: '#6BA5D7'
          };
          return obj;
        }
      },
    },
  },

```

```

        getConnectorDefaults: (connector: ConnectorModel, diagram:
Diagram) => {
            connector.style = {
                strokeColor: '#6BA5D7',
                strokeWidth: 2
            };
            connector.targetDecorator.style.fill = '#6BA5D7';

            connector.targetDecorator.style.strokeColor = '#6BA5D7';
            connector.targetDecorator.shape = 'None';
            connector.type = 'Straight';
            return connector;
        },
    },
}
</script>
<style>
    @import "../../node_modules/@syncfusion/ej2-vue-
diagrams/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/diagram/automatic-layout/RadialTreeLayout-cs1" %}

### Organizational Chart

An organizational chart is a diagram that displays the structure of an organization and relationships. To create an organizational chart, the [type](#) of layout should be set as an **OrganizationalChart**.

The following code example illustrates how to create an organizational chart.

#### APP.VUE

```

<template>
    <div id="app">
        <ejs-diagram id="diagram" :width='width' :height='height'
:getNodeDefaults='getNodeDefaults'
:getConnectorDefaults='getConnectorDefaults' :layout='layout'
:dataSourceSettings='dataSourceSettings' ></ejs-diagram>
    </div>
</template>
<script>
    import Vue from 'vue';
    import { DiagramPlugin, Diagram, HierarchicalTree,
DataBinding, DiagramTools, NodeModel } from '@syncfusion/ej2-vue-diagrams';
    import { DataManager, Query } from "@syncfusion/ej2-data";
    Diagram.Inject(DataBinding, HierarchicalTree);
    Vue.use(DiagramPlugin);
    export let data = [{
        Id: "parent",
        Role: "Project Management"
    },
    {
        Id: 1,
        Role: "R&D Team",
        Team: "parent"
    },
    ],

```

```
{
  Id: 3,
  Role: "Philosophy",
  Team: "1"
},
{
  Id: 4,
  Role: "Organization",
  Team: "1"
},
{
  Id: 5,
  Role: "Technology",
  Team: "1"
},
{
  Id: 7,
  Role: "Funding",
  Team: "1"
},
{
  Id: 8,
  Role: "Resource Allocation",
  Team: "1"
},
{
  Id: 9,
  Role: "Targeting",
  Team: "1"
},
{
  Id: 11,
  Role: "Evaluation",
  Team: "1"
},
{
  Id: 156,
  Role: "HR Team",
  Team: "parent"
},
{
  Id: 13,
  Role: "Recruitment",
  Team: "156"
},
{
  Id: 113,
  Role: "Training",
  Team: "12"
},
{
  Id: 112,
  Role: "Employee Relation",
  Team: "156"
},
{
  Id: 14,
```

```

    Role: "Record Keeping",
    Team: "12"
  },
  {
    Id: 15,
    Role: "Compensations & Benefits",
    Team: "12"
  },
  {
    Id: 16,
    Role: "Compliances",
    Team: "12"
  },
  {
    Id: 17,
    Role: "Production & Sales Team",
    Team: "parent"
  },
  {
    Id: 119,
    Role: "Design",
    Team: "17"
  },
  {
    Id: 19,
    Role: "Operation",
    Team: "17"
  },
  {
    Id: 20,
    Role: "Support",
    Team: "17"
  },
  {
    Id: 21,
    Role: "Quality Assurance",
    Team: "17"
  },
  {
    Id: 23,
    Role: "Customer Interaction",
    Team: "17"
  },
  {
    Id: 24,
    Role: "Support and Maintenance",
    Team: "17"
  },
  {
    Id: 25,
    Role: "Task Coordination",
    Team: "17"
  }
];
let items: DataManager = new DataManager(data as JSON[], new
Query().take(5));
export default {

```

```

name: 'app',
data() {
  return {
    width: "100%",
    height: "590px",
    //Uses layout to auto-arrange nodes on the Diagram page
    layout: {
      //set the type as Organizational Chart
      type: 'OrganizationalChart'
    }, //Configures data source for Diagram
    dataSourceSettings: {
      id: 'Id',
      parentId: 'Team',
      dataManager: items
    }, //Sets the default properties for nodes and connectors
    getNodeDefaults: (obj: NodeModel) => {
      obj.shape = {
        type: 'Text',
        content: (obj.data as {
          Role: 'string'
        }).Role
      };
      obj.style = {
        fill: 'None',
        strokeColor: 'none',
        strokeWidth: 2,
        bold: true,
        color: 'white'
      };
      obj.borderColor = 'white';
      obj.backgroundColor = '#6BA5D7';
      obj.borderWidth = 1;
      obj.width = 75;
      obj.height = 40;
      (obj.shape as TextModel).margin = {
        left: 5,
        right: 5,
        top: 5,
        bottom: 5
      };
      return obj;
    },
    getConnectorDefaults: (connector: ConnectorModel, diagram:
Diagram) => {
      connector.style = {
        strokeColor: '#6BA5D7',
        strokeWidth: 2
      };
      connector.targetDecorator.style.fill = '#6BA5D7';
      connector.targetDecorator.style.strokeColor = '#6BA5D7';
      connector.type = 'Orthogonal';
      return connector;
    }
  },
}

```

```

</script>
<style>
  @import "../node_modules/@syncfusion/ej2-vue-
  diagrams/styles/material.css";
</style>

```

```

{% previewsample "page.domainurl/code-snippet/diagram/automatic-layout/OrganizationalChart-cs1"
%}

```

Organizational chart layout starts parsing from root and iterate through all its child elements. The `getLayoutInfo` method provides necessary information of a node's children and the way to arrange (direction, orientation, offsets, etc.) them. The arrangements can be customized by overriding this function as explained.

### GetLayoutInfo

Set chart orientations, chart types, and offset to be left between parent and child nodes by overriding the method, `diagram.layout.getLayoutInfo`. The `getLayoutInfo` method is called to configure every subtree of the organizational chart. It takes the following arguments.

- node: Parent node to that options are to be customized.
- options: Object to set the customizable properties.

### APP.VUE

```

<template>
  <div id="app">
    <ejs-diagram id="diagram" :width='width' :height='height'
    :getNodeDefaults='getNodeDefaults'
    :snapSettings='snapSettings':getConnectorDefaults='getConnectorDefaults'
    :layout='layout' :dataSourceSettings='dataSourceSettings' ></ejs-diagram>
  </div>
</template>
<script>
  import Vue from 'vue';
  import { DiagramPlugin,Diagram, HierarchicalTree,
  DataBinding,DiagramTools,NodeModel } from '@syncfusion/ej2-vue-diagrams';
  import { DataManager,Query } from "@syncfusion/ej2-data";
  Diagram.Inject(DataBinding, HierarchicalTree);
  Vue.use(DiagramPlugin);
  export let data = [{
    Id: 1,
    Role: "General Manager"
  },
  {
    Id: 2,
    Role: "Assistant Manager",
    Team: 1
  },
  {
    Id: 3,
    Role: "Human Resource Manager",
    Team: 1
  },
  {

```

```

    Id: 4,
    Role: "Design Manager",
    Team: 1
  },
  {
    Id: 5,
    Role: "Operation Manager",
    Team: 1
  },
  {
    Id: 6,
    Role: "Marketing Manager",
    Team: 1
  }
];
let items: DataManager = new DataManager(data as JSON[], new
Query().take(7));
export default {
  name: 'app',
  data() {
    return {
      width: "100%",
      height: "590px",
      //Uses layout to auto-arrange nodes on the Diagram page
      snapSettings: {
        constraints: 0
      }, //Uses layout to auto-arrange nodes on the Diagram page
      layout: {
        //Sets layout type
        type: 'OrganizationalChart',
        getLayoutInfo: (node: Node, options: TreeInfo) => {
          if (!options.hasSubTree) {
            options.type = 'Center';
            options.orientation = 'Horizontal';
          }
        }
      }, //Configures data source for Diagram
      dataSourceSettings: {
        id: 'Id',
        parentId: 'Team',
        dataManager: items
      },
      //Sets the default properties for nodes and connectors
      getNodeDefaults: (obj: Node, diagram: Diagram) => {
        obj.width = 150;
        obj.height = 50;
        obj.style.fill = '#6BA5D7';
        obj.annotations = [{
          content: obj.data['Role'],
          style: {
            color: 'white'
          }
        }];
      },
      return obj;
    },
    getConnectorDefaults: (connector: ConnectorModel, diagram:
Diagram) => {
      connector.style = {

```



```

        strokeColor: '#6BA5D7',
        strokeWidth: 2
    };
    connector.targetDecorator.style.fill = '#6BA5D7';

    connector.targetDecorator.style.strokeColor = '#6BA5D7';
    connector.targetDecorator.shape = 'None';
    connector.targetDecorator.shape = 'None';
    connector.type = 'Orthogonal';
    return connector;
    }
    },
    }
}
</script>
<style>
    @import "../node_modules/@syncfusion/ej2-vue-
    diagrams/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/diagram/automatic-layout/GetLayoutInfo-cs1" %}

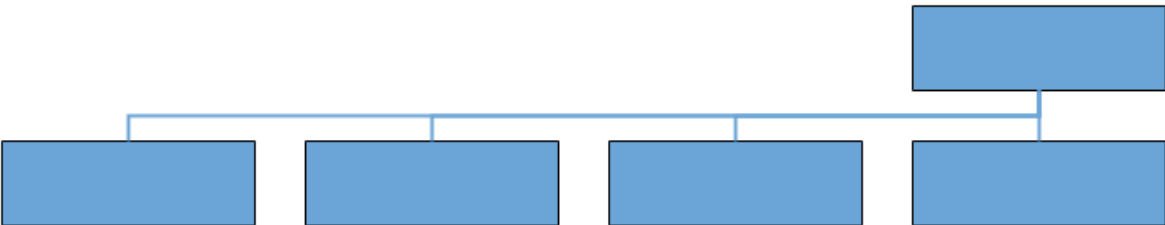
The following table illustrates the properties that “options” argument takes.

Property	Description
Default Value	
-----	-----
-----	-----
-----	-----
options.assistants	By default, the collection is empty. When any of the child nodes have to be set as <b>Assistant</b> , you can remove from children collection and have to insert into assistants collection.
Empty array	
options.orientation	Gets or sets the organizational chart orientation.
SubTreeOrientation.Vertical	
options.type	Gets or sets the chart organizational chart type.
For horizontal chart orientation:SubTreeAlignments.Center and for vertical chart orientation:SubTreeAlignments.Alternate	
options.offset	Offset is the horizontal space to be left between parent and child nodes.
20 pixels applicable only for vertical chart orientations.	
options.hasSubTree	Gets whether the node contains subtrees.
Boolean	
options.level	Gets the depth of the node from layout root.
Number	
options.enableRouting	By default, connections are routed based on the chart type and orientations. This property gets or sets whether default routing is to be enabled or disabled.
	true

| options.rows | Sets the number of rows on which the child nodes will be arranged. Applicable only for balanced type horizontal tree. | Number |

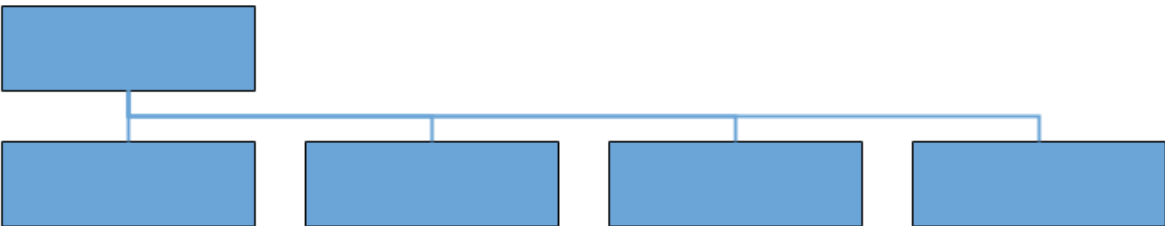
The following table illustrates the different chart orientations and chart types.

Orientation	Type	Description
Example		
-----	-----	-----
-----	-----	
Horizontal	Left	Arranges the child nodes horizontally at the left side of the parent.



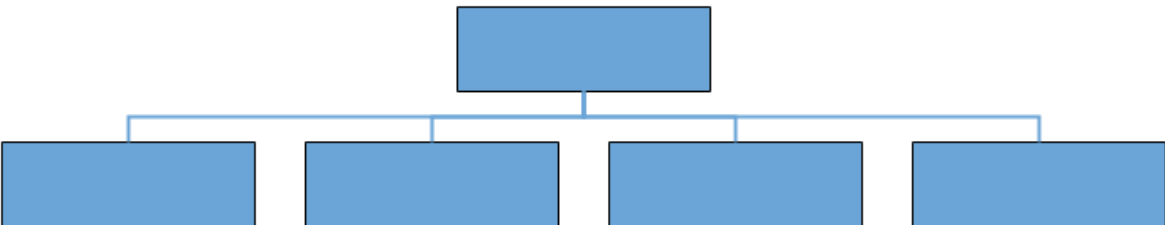
|

| Right | Arranges the child nodes horizontally at the right side of the parent. |



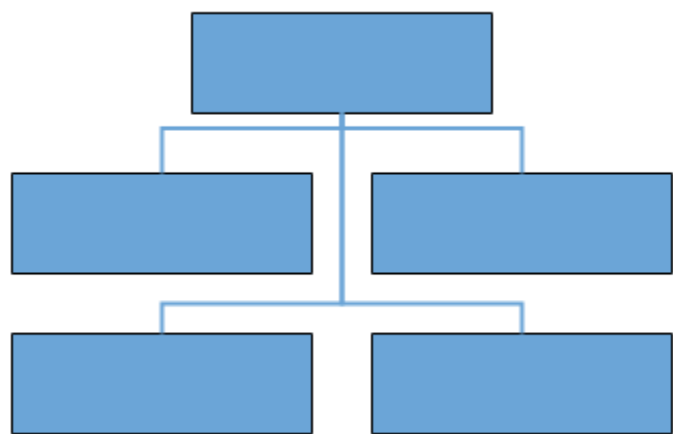
|

| Center | Arranges the children like standard tree layout orientation. |

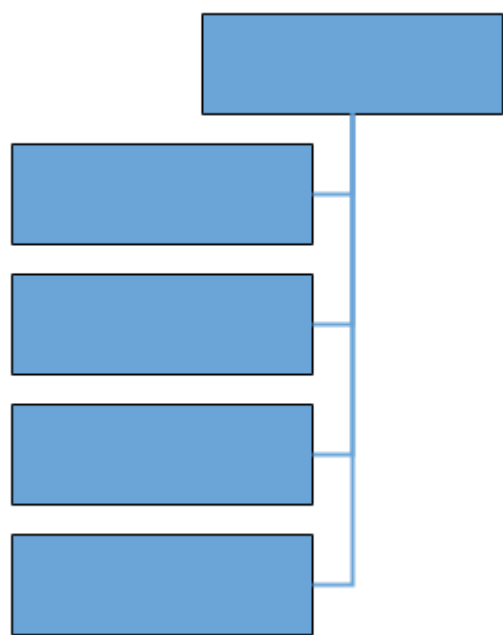


|

| Balanced | Arranges the leaf level child nodes in multiple rows. |

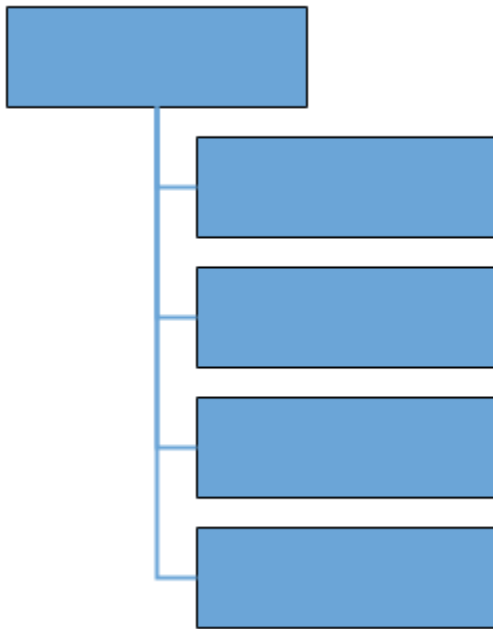


| Vertical | Left | Arranges the children vertically at the left



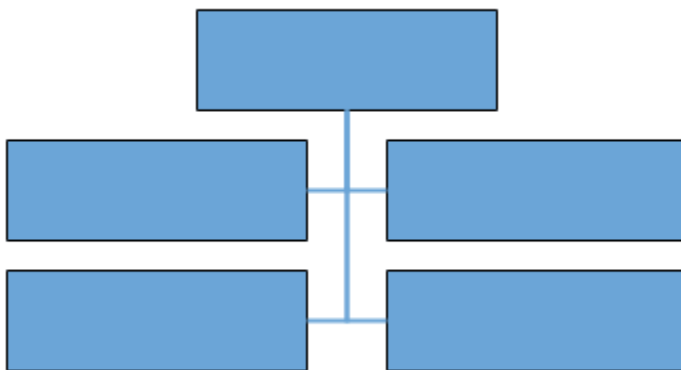
side of the parent. |

| Right | Arranges the children vertically at the right side of the parent. |



|

| Alternate | Arranges the children vertically at both left and right sides of the parent. |



|

The following code example illustrates how to set the vertical right arrangement to the leaf level trees.

#### **APP.VUE**

```
<template>
  <div id="app">
    <ejs-diagram id="diagram" :width='width' :height='height'
      :getNodeDefaults='getNodeDefaults'
      :snapSettings='snapSettings':getConnectorDefaults='getConnectorDefaults'
      :layout='layout' :dataSourceSettings='dataSourceSettings' ></ejs-diagram>
    </div>
  </template>
  <script>
    import Vue from 'vue';
```

```
import { DiagramPlugin, Diagram, HierarchicalTree,
DataBinding, TreeInfo, NodeModel } from '@syncfusion/ej2-vue-diagrams';
import { DataManager, Query } from '@syncfusion/ej2-data';
Diagram.Inject(DataBinding, HierarchicalTree);
Vue.use(DiagramPlugin);
export let data = [{
  Id: "parent",
  Role: "Board"
},
{
  Id: "1",
  Role: "General Manager",
  Manager: "parent"
},
{
  Id: "2",
  Role: "Human Resource Manager",
  Manager: "1"
},
{
  Id: "3",
  Role: "Trainers",
  Manager: "2"
},
{
  Id: "4",
  Role: "Recruiting Team",
  Manager: "2"
},
{
  Id: "6",
  Role: "Design Manager",
  Manager: "1"
},
{
  Id: "7",
  Role: "Design Supervisor",
  Manager: "6"
},
{
  Id: "8",
  Role: "Development Supervisor",
  Manager: "6"
},
{
  Id: "9",
  Role: "Drafting Supervisor",
  Manager: "6"
},
{
  Id: "10",
  Role: "Marketing Manager",
  Manager: "1"
},
{
  Id: "11",
  Role: "Oversea sales Manager",
```

```

    Manager: "10"
  },
  {
    Id: "12",
    Role: "Petroleum Manager",
    Manager: "10"
  },
  {
    Id: "13",
    Role: "Service Dept. Manager",
    Manager: "10"
  }
];
let items: DataManager = new DataManager(data as JSON[], new
Query().take(7));
export default {
  name: 'app',
  data() {
    return {
      width: "100%",
      height: "590px",
      //Uses layout to auto-arrange nodes on the Diagram page
      snapSettings: {
        constraints: 0
      }, //Uses layout to auto-arrange nodes on the Diagram page
      layout: {
        //Sets layout type
        type: 'OrganizationalChart',
        //Defines getLayoutInfo
        getLayoutInfo: (node: Node, options: TreeInfo) => {
          if (node.data['Role'] === 'General Manager') {
            options.assistants.push(options.children[0]);
            options.children.splice(0, 1);
          }
          if (!options.hasSubTree) {
            options.type = 'Right';
            options.orientation = 'Vertical';
          }
        }
      }, //Configures data source for Diagram
      dataSourceSettings: {
        id: 'Id',
        parentId: 'Manager',
        dataManager: items
      }, //Sets the default properties for nodes and connectors
      getNodeDefaults: (obj: Node, diagram: Diagram) => {
        obj.width = 150;
        obj.height = 50;
        obj.borderColor = 'white';
        obj.style.fill = '#6BA5D7';
        obj.borderWidth = 1;
        obj.annotations = [{
          content: obj.data['Role'],
          style: {
            color: 'white'
          }
        }
      ]
    };
    return obj;
  }
};

```

```

    },
    getConnectorDefaults: (connector: ConnectorModel, diagram:
Diagram) => {
        connector.style = {
            strokeColor: '#6BA5D7',
            strokeWidth: 2
        };
        connector.targetDecorator.style.fill = '#6BA5D7';
        connector.targetDecorator.style.strokeColor = '#6BA5D7';
        connector.targetDecorator.shape = 'None';
        connector.type = 'Orthogonal';
        return connector;
    }
},
}
}
</script>
<style>
    @import "../node_modules/@syncfusion/ej2-vue-
diagrams/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/diagram/automatic-layout/illustration-cs1" %}

### Assistant

Assistants are child item that have a different relationship with the parent node. They are laid out in a dedicated part of the tree. A node can be specified as an assistant of its parent by adding it to the `assistants` property of the argument "options".

The following code example illustrates how to add assistants to layout.

### APP.VUE

```

<template>
    <div id="app">
        <ejs-diagram id="diagram" :width='width' :height='height'
:getNodeDefaults='getNodeDefaults'
:snapSettings='snapSettings':getConnectorDefaults='getConnectorDefaults'
:layout='layout' :dataSourceSettings='dataSourceSettings' ></ejs-diagram>
    </div>
</template>
<script>
    import Vue from 'vue';
    import { DiagramPlugin,Diagram, HierarchicalTree,
DataBinding,TreeInfo,NodeModel } from '@syncfusion/ej2-vue-diagrams';
    import { DataManager,Query } from "@syncfusion/ej2-data";
    Diagram.Inject(DataBinding, HierarchicalTree);
    Vue.use(DiagramPlugin);
    export let data = [{
        Id: 1,
        Role: "General Manager"
    },
    {
        Id: 2,
        Role: "Assistant Manager",
        Team: 1
    }
]

```

```

    },
    {
      Id: 3,
      Role: "Human Resource Manager",
      Team: 1
    },
    {
      Id: 4,
      Role: "Design Manager",
      Team: 1
    },
    {
      Id: 5,
      Role: "Operation Manager",
      Team: 1
    },
    {
      Id: 6,
      Role: "Marketing Manager",
      Team: 1
    }
  ];
  let items: DataManager = new DataManager(data as JSON[], new
  Query().take(7));
  export default {
    name: 'app',
    data() {
      return {
        width: "100%",
        height: "590px",
        snapSettings: {
          constraints: 0
        }, //Uses layout to auto-arrange nodes on the Diagram page
        layout: {
          //Sets layout type
          type: 'OrganizationalChart',
          // define the getLayoutInfo
          getLayoutInfo: (node: Node, options: TreeInfo) => {
            if (node.data['Role'] === 'General Manager') {
              options.assistants.push(options.children[0]);
              options.children.splice(0, 1);
            }
            if (!options.hasSubTree) {
              options.type = 'Center';
              options.orientation = 'Horizontal';
            }
          }
        }, //Initializes the node template.
        dataSourceSettings: {
          id: 'Id',
          parentId: 'Team',
          dataManager: items
        },
        //Sets the default properties for nodes and connectors
        getNodeDefaults: (obj: Node, diagram: Diagram) => {
          obj.width = 150;
          obj.height = 50;
          obj.borderColor = 'white';

```



```

        obj.style.fill = '#6BA5D7';
        obj.borderWidth = 1;
        obj.annotations = [{
            content: obj.data['Role'],
            style: {
                color: 'white'
            }
        }];
        return obj;
    },
    getConnectorDefaults: (connector: ConnectorModel, diagram:
Diagram) => {
        connector.style = {
            strokeColor: '#6BA5D7',
            strokeWidth: 2
        };
        connector.targetDecorator.style.fill = '#6BA5D7';

        connector.targetDecorator.style.strokeColor = '#6BA5D7';
        connector.targetDecorator.shape = 'None';
        connector.type = 'Orthogonal';
        return connector;
    }
},
    },
}
</script>
<style>
    @import "../node_modules/@syncfusion/ej2-vue-
diagrams/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/diagram/automatic-layout/Assistant-cs1" %}

### Symmetric layout

The symmetric layout has been formed using nodes position by closer together or pushing them further apart. This is repeated iteratively until the system comes to an equilibrium state.

The layout's [springLength](#) defined as how long edges should be, ideally. This will be the resting length for the springs. Edge attraction and vertex repulsion forces to be defined by using layout's [springFactor](#), the more sibling nodes repel each other. The relative positions do not change any more from one iteration to the next. The number of iterations can be specified by using layout's [maxIteration](#).

The following code illustrates how to arrange the nodes in a radial tree structure.

Note: If you want to use symmetric layout in diagram, you need to inject SymmetricLayout in the diagram.

### Mind Map layout

A mind map is a diagram that displays the nodes as a spider diagram organizes information around a central concept. To create mind map, the [type](#) of layout should be set as **MindMap**.

### Tree Orientation in layout

An [Orientation](#) of a `MindMapTreeLayout` is used to arrange the tree layout according to a specific direction. By default, the orientation is set to Horizontal. The following table outlines the various orientation types available:

Orientation Type	Description
Horizontal	Aligns the tree layout from left to right
Vertical	Aligns the tree layout from top to bottom

**Note:** If you want to use mind map layout in diagram, you need to inject MindMap in the diagram.

The following code example illustrates how to create an mindmap layout.

#### APP.VUE

```
<template>
  <div id="app">
    <ejs-diagram id="diagram" :width='width' :height='height'
      :getNodeDefaults='getNodeDefaults'
      :getConnectorDefaults='getConnectorDefaults' :layout='layout'
      :dataSourceSettings='dataSourceSettings' ></ejs-diagram>
    </div>
  </template>
<script>
  import Vue from 'vue';
  import { DiagramPlugin, Diagram, MindMap, DataBinding, TreeInfo, NodeModel
} from '@syncfusion/ej2-vue-diagrams';
  import { DataManager, Query } from "@syncfusion/ej2-data";
  Diagram.Inject(DataBinding, MindMap);
  Vue.use(DiagramPlugin);
  export let data = [{
    id: 1,
    Label: 'StackPanel'
  },
  {
    id: 2,
    Label: 'Label',
    parentId: 1
  },
  {
    id: 3,
    Label: 'ListBox',
    parentId: 1
  },
  {
    id: 4,
    Label: 'StackPanel',
    parentId: 2
  },
  {
    id: 5,
    Label: 'Border',
    parentId: 2
  }
];
```

```

    },
    {
      id: 6,
      Label: 'Border',
      parentId: 4
    },
    {
      id: 7,
      Label: 'Button',
      parentId: 4
    },
    {
      id: 8,
      Label: 'ContentPresenter',
      parentId: 5
    },
    {
      id: 9,
      Label: 'Text Block',
      parentId: 5
    }
  ]];
  let items: DataManager = new DataManager(data as JSON[], new
  Query().take(7));
  export default {
    name: 'app',
    data() {
      return {
        width: "100%",
        height: "590px",
        layout: {
          //Sets layout type
          type: 'MindMap',
          orientation: 'Horizontal'
        }, //Configures data source for Diagram
        dataSourceSettings: {
          id: 'id',
          parentId: 'parentId',
          dataManager: items,
          root: String(1)
        }, //Sets the default properties for nodes and connectors
        getNodeDefaults: (obj: Node) => {
          obj.shape = {
            type: 'Text',
            content: (obj.data as {
              Label: 'string'
            }).Label,
          };
          obj.style = {
            fill: '#6BA5D7',
            strokeColor: 'none',
            strokeWidth: 2
          };
          obj.borderColor = 'white';
          obj.backgroundColor = '#6BA5D7';
          obj.borderWidth = 1;
          (obj.shape as TextModel).margin = {
            left: 5,

```

```

        right: 5,
        top: 5,
        bottom: 5
    };
    return obj;
},
getConnectorDefaults: (connector: ConnectorModel, diagram:
Diagram) => {
    connector.style = {
        strokeColor: '#6BA5D7',
        strokeWidth: 2
    };
    connector.targetDecorator.style.fill = '#6BA5D7';
    connector.targetDecorator.style.strokeColor = '#6BA5D7';
    connector.type = 'Orthogonal';
    return connector;
}
},
},
}
</script>
<style>
    @import "../node_modules/@syncfusion/ej2-vue-
diagrams/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/diagram/automatic-layout/Mind-cs1" %}

### Complex hierarchical tree

Complex hierarchical tree layout is the extended version of the hierarchical tree layout. The child had been two or more parents. To create a complex hierarchical tree, the [type](#) of layout should be set as **ComplexHierarchicalTree**.

Note: If you want to use Complex hierarchical layout in diagram, you need to inject **ComplexHierarchicalTree** in the diagram.

The following code example illustrates how to create a complex hierarchical tree.

### APP.VUE

```

<template>
    <div id="app">
        <ejs-diagram id="diagram" :width='width' :height='height'
        :nodes='nodes' :connectors='connectors'
        :getNodeDefaults='getNodeDefaults'
        :getConnectorDefaults='getConnectorDefaults' :layout='layout' ></ejs-
diagram>
    </div>
</template>
<script>
    import Vue from 'vue';
    import { DiagramPlugin,Diagram,DataBinding, ConnectorModel, NodeModel,
DiagramConstraints, ComplexHierarchicalTree } from '@syncfusion/ej2-vue-
diagrams';
    import { DataManager,Query } from "@syncfusion/ej2-data";
    Diagram.Inject(DataBinding, ComplexHierarchicalTree);

```

```
Vue.use(DiagramPlugin);
let nodes = [{
  id: 'node1',
  width: 70,
  height: 70,
  annotations: [{
    content: 'node1'
  }]
},
{
  id: 'node2',
  width: 70,
  height: 70,
  annotations: [{
    content: 'node2'
  }]
},
{
  id: 'node3',
  width: 70,
  height: 70,
  annotations: [{
    content: 'node3'
  }]
},
{
  id: 'node4',
  width: 70,
  height: 70,
  annotations: [{
    content: 'node4'
  }]
},
{
  id: 'node5',
  width: 70,
  height: 70,
  annotations: [{
    content: 'node5'
  }]
},
{
  id: 'node8',
  width: 70,
  height: 70,
  annotations: [{
    content: 'node8'
  }]
},
{
  id: 'node9',
  width: 70,
  height: 70,
  annotations: [{
    content: 'node9'
  }]
}]
```

```

let connectors = [
  {
    id: 'connectr',
    sourceID: 'node1',
    targetID: 'node4'
  },
  {
    id: 'connectr1',
    sourceID: 'node2',
    targetID: 'node4'
  },
  {
    id: 'connectr3',
    sourceID: 'node3',
    targetID: 'node4'
  },
  {
    id: 'connectr4',
    sourceID: 'node4',
    targetID: 'node5'
  }
]
export default {
  name: 'app',
  data() {
    return {
      width: "100%",
      height: "590px",
      nodes: nodes,
      connectors: connectors,
      getNodeDefaults: (obj: Node) => {
        obj.shape = {
          type: 'Text',
          style: {
            color: 'white';
          }
        };
        obj.style = {
          fill: '#6BA5D7',
          strokeColor: 'none',
          strokeWidth: 2
        };
        obj.borderColor = 'white';
        obj.backgroundColor = '#6BA5D7';
        obj.borderWidth = 1;
        (obj.shape as TextModel).margin = {
          left: 5,
          right: 5,
          top: 5,
          bottom: 5
        };
      },
      return obj;
    },
    getConnectorDefaults: (connector: ConnectorModel, diagram:
Diagram) => {
      connector.style = {
        strokeColor: '#6BA5D7',

```

```

        strokeWidth: 2
    };
    connector.targetDecorator.style.fill = '#6BA5D7';
    connector.targetDecorator.style.strokeColor = '#6BA5D7';
    connector.type = 'Orthogonal';
    return connector;
},
    layout: {
        //Sets layout type
        type: 'ComplexHierarchicalTree',
        orientation: 'TopToBottom'
    },
}
},
}
</script>
<style>
    @import "../node_modules/@syncfusion/ej2-vue-
    diagrams/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/diagram/automatic-layout/Complex-cs1" %}

### Line Distribution

Line distribution is used to arrange the connectors without overlapping in automatic layout. In some cases, the automatic layout connectors connecting to the nodes will be overlapped with one another. So user can decide whether the segment of each connector from a single parent node should be same point or different point. The [connectionPointOrigin](#) property of layout is used to enable or disable the line distribution in layout. By default connectionPointOrigin will be [SamePoint](#).

The following code example illustrates how to create a complex hierarchical tree with line distribution.

Note: If you want to use line distribution in diagram layout, you need to inject [LineDistribution](#) module in the diagram.

### APP.VUE

```

<template>
    <div id="app">
        <ejs-diagram id="diagram" :width='width' :height='height'
        :getNodeDefaults='getNodeDefaults'
        :getConnectorDefaults='getConnectorDefaults' :layout='layout'
        :dataSourceSettings='dataSourceSettings' ></ejs-diagram>
    </div>
</template>
<script>
    import Vue from 'vue';
    import { DiagramPlugin, Diagram, ComplexHierarchicalTree, DataBinding,
    LineDistribution, NodeModel, ConnectionPointOrigin, ConnectorModel } from
    '@syncfusion/ej2-vue-diagrams';
    import { DataManager, Query } from "@syncfusion/ej2-data";
    Diagram.Inject(DataBinding, ComplexHierarchicalTree, LineDistribution);
    Vue.use(DiagramPlugin);
    export let data = [
        { "Name": "node11" },
        { "Name": "node12", "ReportingPerson": ["node114"] },
    ]

```

```

    { "Name": "node13", "ReportingPerson": ["node12"] },
    { "Name": "node14", "ReportingPerson": ["node12"] },
    { "Name": "node15", "ReportingPerson": ["node12"] },
    { "Name": "node116", "ReportingPerson": ["node22", "node12"] },
    { "Name": "node16", "ReportingPerson": [] },
    { "Name": "node18", "ReportingPerson": [] },
    { "Name": "node21" },
    { "Name": "node22", "ReportingPerson": ["node114"] },
    { "Name": "node23", "ReportingPerson": ["node22"] },
    { "Name": "node24", "ReportingPerson": ["node22"] },
    { "Name": "node25", "ReportingPerson": ["node22"] },
    { "Name": "node26", "ReportingPerson": [] },
    { "Name": "node28", "ReportingPerson": [] },
    { "Name": "node31" },
    { "Name": "node114", "ReportingPerson": ["node11", "node21",
"node31"]}
];
let items: DataManager = new DataManager(data as JSON[], new
Query().take(7));
export default {
  name: 'app',
  data() {
    return {
      width: "100%",
      height: "590px",
      //Uses layout to auto-arrange nodes on the Diagram page
      layout: {
        type: 'ComplexHierarchicalTree',
        connectionPointOrigin:
ConnectionPointOrigin.DifferentPoint,
        horizontalSpacing: 40, verticalSpacing: 40,
horizontalAlignment: "Left", verticalAlignment: "Top",
        margin: { left: 0, right: 0, top: 0, bottom: 0 },
        orientation: 'TopToBottom'
      },
      dataSourceSettings: {
        id: 'Name',
        parentId: 'ReportingPerson',
        dataManager: items
      },
      getNodeDefaults: (obj) => {
        obj.width = 40; obj.height = 40;
        obj.shape = { type: 'Basic', shape: 'Rectangle',
cornerRadius: 7 };
        obj.style = { fill: '#6BA5D7', strokeColor: 'none',
strokeWidth: 2 };
        obj.borderWidth = 1;
        obj.backgroundColor = '#6BA5D7';
        return obj;
      },
      getConnectorDefaults: (connector) => {
        connector.type = 'Orthogonal';
        connector.cornerRadius = 7;
        connector.targetDecorator.height = 7;
        connector.targetDecorator.width = 7;
        connector.style = { strokeColor: '#6BA5D7', strokeWidth:
1 };

```



```

connector.targetDecorator.style.fill = '#6BA5D7';
connector.targetDecorator.style.strokeColor =
'#6BA5D7';

    return connector;
  },
},
},
},
mounted: function() {
  let diagramInstance: Diagram;
  let diagramObj: any = document.getElementById("diagram");
  diagramInstance = diagramObj.ej2_instances[0];
  diagramInstance.fitToPage({
    mode: 'Width',
  });
}
provide: {
  diagram: [DataBinding, ComplexHierarchicalTree,
LineDistribution]
},
}
</script>
<style>
  @import
  "https://ej2.syncfusion.com/vue/documentation/..../node_modules/@syncfusion/ej
2-vue-diagrams/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/diagram/automatic-layout/LineDistribution-cs1" %}

### Linear Arrangement

Linear arrangement is used to linearly arrange the child nodes in layout, which means the parent node is placed in the center corresponding to its children. When line distribution is enabled, linear arrangement is also activated by default. The [arrangement](#) property of layout is used to enable or disable the linear arrangement in layout. By default arrangement will be **Nonlinear**.

Note: If you want to use linear arrangement in diagram layout, you need to inject LineDistribution module in the diagram. Linear arrangement is applicable only for complex hierarchical tree layout.

The following code illustrates how to allow a linear arrangement in diagram layout.

```

export default {
  name: 'app',
  data() {
    return {
      width: "100%",
      height: "590px",
      layout: {
        type: 'ComplexHierarchicalTree',

```

```
//To arrange a child nodes in a linear manner
arrangement: ChildArrangement.Linear,
horizontalSpacing: 40, verticalSpacing: 40,
orientation: 'TopToBottom'
},
}
},
provide: {
  diagram: [DataBinding, ComplexHierarchicalTree, LineDistribution]
},
},
,
```

#### *Prevent connectors overlay*

The below constraints prevents the connector segments overlapping nodes with a complex hierarchical layout.

```
,
export default {
  name: 'app',
  data() {
    return {
      width: "100%",
      height: "590px",
      layout: {
        //this prevents connector segments overlapping
        enableRouting: true
      },
    },
  },
  provide: {
    diagram: [DataBinding, ComplexHierarchicalTree, LineDistribution]
  },
},
,
```

### Customize layout

Orientation, spacings, and position of the layout can be customized with a set of properties.

To explore layout properties, refer to [Layout Properties](#).

### Layout bounds

Diagram provides support to align the layout within any custom rectangular area. For more information about bounds, refer to [bounds](#).

### Layout alignment

The layout can be aligned anywhere over the layout bounds/viewport using the [horizontalAlignment](#) and [verticalAlignment](#) properties of the layout.

The following code illustrates how to align the layout at the top-left of the layout bounds.

### APP.VUE

```
<template>
  <div id="app">
    <ejs-diagram id="diagram" :width='width' :height='height'
      :getNodeDefaults='getNodeDefaults'
      :getConnectorDefaults='getConnectorDefaults' :layout='layout'
      :dataSourceSettings='dataSourceSettings' ></ejs-diagram>
    </div>
  </template>
<script>
  import Vue from 'vue';
  import { DiagramPlugin, HierarchicalTree, DataBinding, Diagram, Rect }
  from '@syncfusion/ej2-vue-diagrams';
  import { DataManager, Query } from "@syncfusion/ej2-data";
  Diagram.Inject(DataBinding, HierarchicalTree);
  Vue.use(DiagramPlugin);
  let data: object[] = [{
    Name: "Steve-Ceo"
  },
  {
    Name: "Kevin-Manager",
    ReportingPerson: "Steve-Ceo"
  },
  {
    Name: "Peter-Manager",
    ReportingPerson: "Kevin-Manager"
  },
  {
    Name: "John- Manager",
    ReportingPerson: "Peter-Manager"
  },
  {
    Name: "Mary-CSE ",
    ReportingPerson: "Peter-Manager"
  },
  ];
  let items: DataManager = new DataManager(data as JSON[], new
  Query().take(7));
  export default {
    name: 'app',
    data() {
```

```

    return {
      width: "100%",
      height: "350px",
      layout: {
        //Sets layout type
        type: 'HierarchicalTree',
        //set layout alignment
        bounds: new Rect(0, 0, 500, 500),
        horizontalSpacing: 25,
        verticalSpacing: 30,
        horizontalAlignment: 'Left',
        verticalAlignment: 'Top'
      },
      dataSourceSettings: {
        id: 'Name',
        parentId: 'ReportingPerson',
        dataManager: items
      },
      getNodeDefaults: (obj: Node) => {
        obj.shape = {
          type: 'Text',
          content: (obj.data as {
            Name: 'string'
          }).Name
        };
        obj.style = {
          fill: 'None',
          strokeColor: 'none',
          strokeWidth: 2,
          bold: true,
          color: 'white'
        };
        obj.width = 100;
        obj.height = 40;
        obj.borderColor = 'white';
        obj.backgroundColor = '#6BA5D7';
        obj.borderWidth = 1;
        return obj;
      },
      getConnectorDefaults: (connector, diagram) => {
        connector.style = {
          strokeColor: '#6BA5D7',
          strokeWidth: 2
        };
        connector.targetDecorator.style.fill = '#6BA5D7';
        connector.targetDecorator.style.strokeColor = '#6BA5D7';
        connector.type = 'Orthogonal';
        return connector;
      }
    },
    provide: {
      diagram: [DataBinding, HierarchicalTree]
    },
  }
</script>

```

```
<style>
  @import
  "https://ej2.syncfusion.com/vue/documentation/../../node_modules/@syncfusion/ej2-vue-diagrams/styles/material.css";
</style>
```

```
{% previewsample "page.domainurl/code-snippet/diagram/automatic-layout/Layout-cs1" %}
```

### Layout spacing

Layout provides support to add space horizontally and vertically between the nodes. The [horizontalSpacing](#) and [verticalSpacing](#) properties of the layout allows you to set the space between the nodes in horizontally and vertically.

### Layout margin

Layout provides support to add some blank space between the layout bounds/viewport and the layout. The [margin](#) property of the layout allows you to set the blank space.

The following code illustrates how to set the layout margin.

### APP.VUE

```
<template>
  <div id="app">
    <ejs-diagram id="diagram" :width='width' :height='height'
    :getNodeDefaults='getNodeDefaults'
    :getConnectorDefaults='getConnectorDefaults' :layout='layout'
    :dataSourceSettings='dataSourceSettings' ></ejs-diagram>
  </div>
</template>
<script>
  import Vue from 'vue';
  import { DiagramPlugin, HierarchicalTree, DataBinding, Diagram, Rect }
  from '@syncfusion/ej2-vue-diagrams';
  import { DataManager, Query } from '@syncfusion/ej2-data';
  Diagram.Inject(DataBinding, HierarchicalTree);
  Vue.use(DiagramPlugin);
  let data: object[] = [{
    Name: "Steve-Ceo"
  },
  {
    Name: "Kevin-Manager",
    ReportingPerson: "Steve-Ceo"
  },
  {
    Name: "Peter-Manager",
    ReportingPerson: "Kevin-Manager"
  },
  {
    Name: "John- Manager",
    ReportingPerson: "Peter-Manager"
  },
  {
    Name: "Mary-CSE ",
    ReportingPerson: "Peter-Manager"
  }
  ],];
```

```

let items: DataManager = new DataManager(data as JSON[], new
Query().take(7));
export default {
  name: 'app',
  data() {
    return {
      width: "100%",
      height: "350px",
      layout: {
        //Sets layout type
        type: 'HierarchicalTree',
        bounds: new Rect(0, 0, 500, 500),
        horizontalSpacing: 25,
        verticalSpacing: 30,
        horizontalAlignment: 'Left',
        verticalAlignment: 'Top'
      },
      dataSourceSettings: {
        id: 'Name',
        parentId: 'ReportingPerson',
        dataManager: items
      },
      getNodeDefaults: (obj: Node) => {
        obj.shape = {
          type: 'Text',
          content: (obj.data as {
            Name: 'string'
          }).Name
        };
        obj.style = {
          fill: 'None',
          strokeColor: 'none',
          strokeWidth: 2,
          bold: true,
          color: 'white'
        };
        obj.width = 100;
        obj.height = 40;
        obj.borderColor = 'white';
        obj.backgroundColor = '#6BA5D7';
        obj.borderWidth = 1;
        return obj;
      },
      getConnectorDefaults: (connector, diagram) => {
        connector.style = {
          strokeColor: '#6BA5D7',
          strokeWidth: 2
        };
        connector.targetDecorator.style.fill = '#6BA5D7';
        connector.targetDecorator.style.strokeColor = '#6BA5D7';
        connector.type = 'Orthogonal';
        return connector;
      }
    }
  },
  provide: {

```

```

        diagram: [DataBinding, HierarchicalTree]
    },
}
</script>
<style>
    @import
    "https://ej2.syncfusion.com/vue/documentation/../../node_modules/@syncfusion/ej2-vue-diagrams/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/diagram/automatic-layout/LayoutMargin-cs1" %}

### Layout orientation

The layout orientation can be used to arrange the layout based on the direction. There are different orientation types that are defined in the following table.

Orientation	Description
-----   -----	
TopToBottom	Aligns the layout from top to bottom. All the roots are placed at top of diagram.
LeftToRight	Aligns the layout from left to right. All the roots are placed at left of diagram.
BottomToTop	Aligns the layout from bottom to top. All the roots are placed at bottom of the diagram.
RightToLeft	Aligns the layout from right to left. All the roots are placed at right of the diagram.

Diagram provides support to customize the [orientation](#) of layout. You can set the desired orientation using `layout.orientation`.

Note: In the diagram the default orientation is `TopToBottom`.

The following code illustrates how to arrange the nodes in a `BottomToTop` orientation.

### APP.VUE

```

<template>
  <div id="app">
    <ejs-diagram id="diagram" :width='width' :height='height'
    :getNodeDefaults='getNodeDefaults'
    :getConnectorDefaults='getConnectorDefaults' :layout='layout'
    :dataSourceSettings='dataSourceSettings' ></ejs-diagram>
  </div>
</template>
<script>
  import Vue from 'vue';
  import { DiagramPlugin, HierarchicalTree, DataBinding, Diagram, Rect }
  from '@syncfusion/ej2-vue-diagrams';
  import { DataManager, Query } from '@syncfusion/ej2-data';
  Diagram.Inject(DataBinding, HierarchicalTree);
  Vue.use(DiagramPlugin);
  let data: object[] = [{
    Name: "Steve-Ceo"
  },
  {
    Name: "Kevin-Manager",

```

```

        ReportingPerson: "Steve-Ceo"
      },
      {
        Name: "Peter-Manager",
        ReportingPerson: "Kevin-Manager"
      },
      {
        Name: "John- Manager",
        ReportingPerson: "Peter-Manager"
      },
      {
        Name: "Mary-CSE ",
        ReportingPerson: "Peter-Manager"
      }
    ],
    let items: DataManager = new DataManager(data as JSON[], new
Query().take(7));
    export default {
      name: 'app',
      data() {
        return {
          width: "100%",
          height: "350px",
          layout: {
            //Sets layout type
            type: 'HierarchicalTree',
            bounds: new Rect(0, 0, 500, 500),
            horizontalSpacing: 25,
            verticalSpacing: 30,
            horizontalAlignment: 'Left',
            verticalAlignment: 'Top',
            orientation: 'BottomToTop'
          },
          dataSourceSettings: {
            id: 'Name',
            parentId: 'ReportingPerson',
            dataManager: items
          },
          getNodeDefaults: (obj: Node) => {
            obj.shape = {
              type: 'Text',
              content: (obj.data as {
                Name: 'string'
              }).Name
            };
            obj.style = {
              fill: 'None',
              strokeColor: 'none',
              strokeWidth: 2,
              bold: true,
              color: 'white'
            };
            obj.width = 100;
            obj.height = 40;
            obj.borderColor = 'white';
            obj.backgroundColor = '#6BA5D7';
            obj.borderWidth = 1;
            return obj;

```



```

        },
        getConnectorDefaults: (connector, diagram) => {
            connector.style = {
                strokeColor: '#6BA5D7',
                strokeWidth: 2
            };
            connector.targetDecorator.style.fill = '#6BA5D7';
        },
        connector.targetDecorator.style.strokeColor = '#6BA5D7';
        connector.type = 'Orthogonal';
        return connector;
    }
},
provide: {
    diagram: [DataBinding, HierarchicalTree]
},
}
</script>
<style>
    @import
    "https://ej2.syncfusion.com/vue/documentation/../../node_modules/@syncfusion/ej2-vue-diagrams/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/diagram/automatic-layout/LayoutOrientation-cs1" %}

#### Fixed node

Layout provides support to arrange the nodes with reference to the position of a fixed node and set it to the [fixedNode](#) of the layout property. This is helpful when you try to expand/collapse a node. It might be expected that the position of the double-clicked node should not be changed.

#### APP.VUE

```

<template>
    <div id="app">
        <ejs-diagram id="diagram" :width='width' :height='height'
        :getNodeDefaults='getNodeDefaults'
        :getConnectorDefaults='getConnectorDefaults' :layout='layout'
        :dataSourceSettings='dataSourceSettings' ></ejs-diagram>
    </div>
</template>
<script>
    import Vue from 'vue';
    import { DiagramPlugin, HierarchicalTree, DataBinding, Diagram, Rect }
    from '@syncfusion/ej2-vue-diagrams';
    import { DataManager, Query } from "@syncfusion/ej2-data";
    Diagram.Inject(DataBinding, HierarchicalTree);
    Vue.use(DiagramPlugin);
    let data: object[] = [{
        Name: "Steve-Ceo",
        //set the offsetX and offsetY for the parent node
        offsetX: 250,
        offsetY: 50
    },
    {

```

```

      Name: "Kevin-Manager",
      ReportingPerson: "Steve-Ceo"
    },
    {
      Name: "Peter-Manager",
      ReportingPerson: "Steve-Ceo"
    },
    {
      Name: "John- Manager",
      ReportingPerson: "Peter-Manager"
    },
    {
      Name: "Mary-CSE ",
      ReportingPerson: "Peter-Manager"
    },
    {
      Name: "Jim-CSE ",
      ReportingPerson: "Kevin-Manager"
    },
    {
      Name: "Martin-CSE",
      ReportingPerson: "Kevin-Manager"
    }
  ]];
  let items: DataManager = new DataManager(data as JSON[], new
  Query().take(7));
  export default {
    name: 'app',
    data() {
      return {
        width: "100%",
        height: "350px",
        layout: {
          type: 'HierarchicalTree',
          bounds: new Rect(0, 0, 500, 500),
          horizontalSpacing: 25,
          verticalSpacing: 30,
          horizontalAlignment: 'Left',
          verticalAlignment: 'Top'
        },
        dataSourceSettings: {
          id: 'Name',
          parentId: 'ReportingPerson',
          dataManager: items
        },
        getNodeDefaults: (obj: Node) => {
          obj.shape = {
            type: 'Text',
            content: (obj.data as {
              Name: 'string'
            }).Name
          };
          obj.style = {
            fill: 'None',
            strokeColor: 'none',
            strokeWidth: 2,
            bold: true,
            color: 'white'
          }
        }
      }
    }
  }

```

```

    };
    obj.width = 50;
    obj.height = 40;
    obj.borderColor = 'white';
    obj.backgroundColor = '#6BA5D7';
    obj.borderWidth = 1;
    (obj.shape as TextModel).margin = {
      left: 25,
      right: 25,
      top: 25,
      bottom: 25
    };
    return obj;
  },
  getConnectorDefaults: (connector, diagram) => {
    connector.style = {
      strokeColor: '#6BA5D7',
      strokeWidth: 2
    };
    connector.targetDecorator.style.fill = '#6BA5D7';

    connector.targetDecorator.style.strokeColor = '#6BA5D7';
    connector.type = 'Orthogonal';
    return connector;
  }
},
provide: {
  diagram: [DataBinding, HierarchicalTree]
},
}
</script>
<style>
  @import "../../node_modules/@syncfusion/ej2-vue-
  diagrams/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/diagram/automatic-layout/Fixed-cs1" %}

### [Expand and collapse](#)

Diagram allows to expand/collapse the subtrees of a layout. The node's `isExpanded` property allows you to expand/collapse its children. The following code example shows how to expand/collapse the children of a node.

### **APP.VUE**

```

<template>
  <div id="app">
    <ejs-diagram id="diagram" :width='width' :height='height'
    :getNodeDefaults='getNodeDefaults'
    :getConnectorDefaults='getConnectorDefaults' :layout='layout'
    :dataSourceSettings='dataSourceSettings' ></ejs-diagram>
  </div>
</template>
<script>
  import Vue from 'vue';

```

```

import { DiagramPlugin, HierarchicalTree,
DataBinding, Diagram, Rect, NodeModel,
    Orientation,
    VerticalAlignment,
    PathElement,
    HorizontalAlignment,
    LayoutAnimation,
    SelectorConstraints,
    DiagramNativeElement,
    IconShapeModel } from '@syncfusion/ej2-vue-diagrams';
import { DataManager, Query } from '@syncfusion/ej2-data';
Diagram.Inject(LayoutAnimation);
Vue.use(DiagramPlugin);
let data: object[] = [{
    'Id': 'parent1',
    'Name': 'Maria ',
    'Designation': 'Managing Director',
    'RatingColor': '#C34444'
},
{
    'Id': 'parent',
    'Name': ' sam',
    'Designation': 'Managing Director',
    'ReportingPerson': 'parent1',
    'RatingColor': '#C34444'
},
{
    'Id': 'parent3',
    'Name': ' sam geo',
    'Designation': 'Managing Director',
    'ReportingPerson': 'parent1',
    'RatingColor': '#C34444'
},
{
    'Id': '80',
    'Name': ' david',
    'Designation': 'Managing Director',
    'ReportingPerson': 'parent3',
    'RatingColor': '#C34444'
},
{
    'Id': '82',
    'Name': ' pirlo',
    'Designation': 'Managing Director',
    'ReportingPerson': 'parent',
    'RatingColor': '#C34444'
}
]];
let items: DataManager = new DataManager(data as JSON[], new
Query().take(7));
export default {
    name: 'app',
    data() {
        return {
            width: "100%",
            height: "350px",
            selectedItems: {
                constraints: ~SelectorConstraints.ResizeAll
            }
        }
    }
}

```

```

    },
    snapSettings: {
      constraints: 0
    },
    layout: {
      // set enableAnimation as true
      enableAnimation: true,
      type: 'OrganizationalChart',
      margin: {
        top: 20
      },
      // define the getLayoutInfo
      getLayoutInfo: (node: Node, tree: TreeInfo) => {
        if (!tree.hasSubTree) {
          tree.orientation = 'vertical';
          tree.type = 'alternate';
        }
      }
    },
    // define the dataSourceSettings
    dataSourceSettings: {
      id: 'Id',
      parentId: 'ReportingPerson',
      dataManager: items
    },
    // define the node defaults
    getNodeDefaults: (obj: Node, diagram: Diagram) => {
      obj.expandIcon = {
        height: 15,
        width: 15,
        shape: "Plus",
        fill: 'lightgray',
        offset: {
          x: .5,
          y: .85
        }
      }
      obj.collapseIcon.offset = {
        x: .5,
        y: .85
      }
      obj.collapseIcon.height = 15;
      obj.collapseIcon.width = 15;
      obj.collapseIcon.shape = "Minus";
      obj.height = 50;
      obj.borderColor = 'white';
      obj.backgroundColor = '#6BA5D7';
      obj.borderWidth = 1;
      obj.style = {
        fill: 'transparent',
        strokeWidth: 2
      };
      return obj;
    },
    // define the connector defaults
    getConnectorDefaults: (connector: ConnectorModel, diagram:
Diagram) => {
      connector.style = {
        strokeColor: '#6BA5D7',
        strokeWidth: 2
      }
    }
  }
}

```

```

        };
        connector.targetDecorator.style.fill = '#6BA5D7';

connector.targetDecorator.style.strokeColor = '#6BA5D7';
        connector.targetDecorator.shape = 'None';
        connector.type = 'Orthogonal';
        return connector;
    }
}
},
provide: {
    diagram: [DataBinding, HierarchicalTree]
},
}
</script>
<style>
    @import "../node_modules/@syncfusion/ej2-vue-
diagrams/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/diagram/automatic-layout/ExpandAndCollapse-cs1"  
 %}

In the previous example, while expanding/collapsing a node, it is set as fixed node in order to prevent it from repositioning.

#### [Refresh layout](#)

Diagram allows to refresh the layout at runtime. To refresh the layout, refer to Refresh layout.

#### [setNodeTemplate](#)

The setNodeTemplate function is provided for the purpose of customizing nodes. It will be called for each node on node initialization. In this function, the node style and its properties can be customized and can bind the custom JSON with node.

#### **APP.VUE**

```

<template>
    <div id="app">
        <ejs-diagram id="diagram" :width='width' :height='height'
:getNodeDefaults='getNodeDefaults'
:getConnectorDefaults='getConnectorDefaults' :layout='layout'
:dataSourceSettings='dataSourceSettings' ></ejs-diagram>
    </div>
</template>
<script>
    import Vue from 'vue';
    import { DiagramPlugin, HierarchicalTree,
DataBinding, Diagram, Rect, NodeModel,
    Orientation,
    VerticalAlignment,
    PathElement,
    HorizontalAlignment,
    LayoutAnimation
    DiagramNativeElement,
    IconShapeModel } from '@syncfusion/ej2-vue-diagrams';
    import { DataManager, Query } from '@syncfusion/ej2-data';

```

```

Diagram.Inject(LayoutAnimation);
Vue.use(DiagramPlugin);
let data: object[] = [{
  Name: "Steve-Ceo"
},
{
  Name: "Kevin-Manager",
  ReportingPerson: "Steve-Ceo",
  color: 'darkcyan'
},
{
  Name: "Peter-Manager",
  ReportingPerson: "Steve-Ceo",
  color: 'white'
},
{
  Name: "John- Manager",
  ReportingPerson: "Peter-Manager",
  color: 'darkcyan'
},
{
  Name: "Mary-CSE ",
  ReportingPerson: "Peter-Manager",
  color: 'white'
},
{
  Name: "Jim-CSE ",
  ReportingPerson: "Kevin-Manager",
  color: 'darkcyan'
},
{
  Name: "Martin-CSE",
  ReportingPerson: "Kevin-Manager",
  color: 'white'
}
];
let items: DataManager = new DataManager(data as JSON[], new
Query().take(7));
export default {
  name: 'app',
  data() {
    return {
      width: "100%",
      height: "350px",
      layout: {
        type: 'HierarchicalTree',
        bounds: new Rect(0, 0, 500, 500),
        horizontalSpacing: 25,
        verticalSpacing: 30,
        horizontalAlignment: 'Left',
        verticalAlignment: 'Top'
      },
      dataSourceSettings: {
        id: 'Name',
        parentId: 'ReportingPerson',
        dataManager: items
      },
      getNodeDefaults: (obj: Node) => {

```

```

        obj.shape = {
            type: 'Text',
            content: (obj.data as {
                Name: 'string'
            }).Name
        };
        obj.style = {
            fill: 'None',
            strokeColor: 'none',
            strokeWidth: 2,
            bold: true,
            color: 'white'
        };
        obj.width = 50;
        obj.height = 40;
        obj.borderColor = 'white';
        obj.backgroundColor = '#6BA5D7';
        obj.borderWidth = 1;
        (obj.shape as TextModel).margin = {
            left: 25,
            right: 25,
            top: 25,
            bottom: 25
        };
        return obj;
    },
    getConnectorDefaults: (connector: ConnectorModel, diagram:
Diagram) => {
        connector.style = {
            strokeColor: '#6BA5D7',
            strokeWidth: 2
        };
        connector.targetDecorator.style.fill = '#6BA5D7';
        connector.targetDecorator.style.strokeColor = '#6BA5D7';
        connector.targetDecorator.shape = 'None';
        connector.type = 'Orthogonal';
        return connector;
    },
    setNodeTemplate: function(obj, diagram) {
        obj.style.borderColor = (obj.data as {
            color: 'string'
        }).color;
    }
    },
    provide: {
        diagram: [DataBinding, HierarchicalTree]
    },
}
</script>
<style>
    @import "../node_modules/@syncfusion/ej2-vue-
diagrams/styles/material.css";
</style>

```



```
{% previewsample "page.domainurl/code-snippet/diagram/automatic-layout/setNodeTemplate-cs1" %}
```

### Accessibility in Vue Diagram component

Diagram provides built-in compliance with the [WAI-ARIA](#) specifications. WAI-ARIA Accessibility supports are achieved through the attributes like `aria-label`. It helps to provides information about elements in a document for assistive technology.

The accessibility compliance for the diagram component is outlined below.

Accessibility Criteria	Compatibility
<hr/>	
<a href="#">WCAG 2.2 Support</a>	<code>&lt;img src="https://cdn.syncfusion.com/content/images/landing-page/intermediate.png" alt="intermediate"&gt;</code>
<a href="#">Section 508 Support</a>	<code>&lt;img src="https://cdn.syncfusion.com/content/images/landing-page/intermediate.png" alt="intermediate"&gt;</code>
<a href="#">Screen Reader Support</a>	<code>&lt;img src="https://cdn.syncfusion.com/content/images/landing-page/yes.png" alt="Yes"&gt;</code>
<a href="#">Right-To-Left Support</a>	<code>&lt;img src="https://cdn.syncfusion.com/content/images/landing-page/no.png" alt="No"&gt;</code>
<a href="#">Color Contrast</a>	<code>&lt;img src="https://cdn.syncfusion.com/content/images/landing-page/yes.png" alt="Yes"&gt;</code>
<a href="#">Mobile Device Support</a>	<code>&lt;img src="https://cdn.syncfusion.com/content/images/landing-page/yes.png" alt="Yes"&gt;</code>
<a href="#">Keyboard Navigation Support</a>	<code>&lt;img src="https://cdn.syncfusion.com/content/images/landing-page/intermediate.png" alt="intermediate"&gt;</code>
<a href="#">Accessibility Checker Validation</a>	<code>&lt;img src="https://cdn.syncfusion.com/content/images/landing-page/yes.png" alt="Yes"&gt;</code>
<a href="#">Axe-core Accessibility Validation</a>	<code>&lt;img src="https://cdn.syncfusion.com/content/images/landing-page/yes.png" alt="Yes"&gt;</code>
<code>&lt;style&gt;</code>	
<code>.post .post-content img {</code>	
<code>display: inline-block;</code>	
<code>margin: 0.5em 0;</code>	
<code>}</code>	
<code>&lt;/style&gt;</code>	
<code>&lt;div&gt;&lt;img src="https://cdn.syncfusion.com/content/images/landing-page/yes.png" alt="Yes"&gt; - All features of the component meet the requirement.&lt;/div&gt;</code>	
<code>&lt;div&gt;&lt;img src="https://cdn.syncfusion.com/content/images/landing-page/intermediate.png" alt="Intermediate"&gt; - Some features of the component do not meet the requirement.&lt;/div&gt;</code>	

<div> - The component does not meet the requirement.</div>

### WAI-ARIA attributes

The Diagram component followed the [WAI-ARIA](#) patterns to meet the accessibility. The following ARIA attributes are used in the Diagram component:

| Attributes | Purpose |

| --- | --- |

| **aria-label** | Provides an accessible name for the Diagram Objects. |

### Aria-label

Attribute provides the text label with some default description for below elements in diagram.

<!-- markdownlint-disable MD033 -->

Element	Default description
ResizeNorthWest	Thumb to resize the selected object on the top-left corner.
ResizeNorthEast	Thumb to resize the selected object on the top-right side direction.
ResizeSouthWest	Thumb to resize the selected object on the bottom-left side direction.
ResizeSouthEast	Thumb to resize the selected object on the bottom-right side direction.
ResizeNorth	Thumb to resize the selected object on the top side direction.
ResizeSouth	Thumb to resize the selected object on the bottom side direction.
ResizeWest	Thumb to resize the selected object on the left side direction.
ResizeEast	Thumb to resize the selected object on the right side direction.
ConnectorSourceThumb	Thumb to move the source point of the connector.
ConnectorTargetThumb	Thumb to move the target point of the connector.
RotateThumb	Thumb to rotate the selected object.

### Mobile device support

Syncfusion Diagram component are more user-friendly and accessible to individuals using mobile devices, including those with disabilities. These are designed to be responsive, adaptable to various screen sizes and orientations, and touch-friendly.

### Screen Reader Support

The Diagram component supports and its information was dictated properly by the screen readers based on the ARIA attributes and content.

### Keyboard navigation support

Syncfusion Diagram component support keyboard navigation, allowing users who rely on alternate methods to effortlessly navigate and interact with the component.

### Keyboard interaction

The Diagram component followed the [keyboard interaction](#) guideline, making it easy for people who use assistive technologies (AT) and those who completely rely on keyboard navigation. The following keyboard shortcuts are supported by the Diagram component.

| **Command** | **Action** |

| --- | --- |

| Ctrl + A | Select All |

| Ctrl + X | Cut |

| Ctrl + C | Copy |

| Ctrl + V | Paste |

| Ctrl + Z | Undo |

| Ctrl + Y | Redo |

| Delete | Delete |

| Up Arrow | Move selected object to up |

| Down Arrow | Move selected object to down |

| Left Arrow | Move selected object to left |

| Right Arrow | Move selected object to right |

| Enter | Start Annotation Edit |

| Escape | End Annotation Edit |

### Ensuring accessibility

The Diagram component's accessibility levels are ensured through an [accessibility-checker](#) and [axe-core](#) software tools during automated testing.

The accessibility compliance of the Diagram component is shown in the following sample. Open the [sample](#) in a new window to evaluate the accessibility of the Diagram component with accessibility tools.

See also

- [Accessibility in Syncfusion Vue components](#)

### Commands in Vue Diagram component

<!-- markdownlint-disable MD010 -->

There are several commands available in the diagram as follows.

- Alignment commands
- Spacing commands
- Sizing commands
- Clipboard commands
- Grouping commands
- Z-order commands

- Zoom commands
- Nudge commands
- FitToPage commands
- Undo/Redo commands

### Align

Alignment commands enable you to align the selected or defined objects such as nodes and connectors with respect to the selection boundary. Refer to [align](#) commands which shows how to use align methods in the diagram.

<!-- markdownlint-disable MD033 -->

| Parameters | Description |

|:-----| :-----: |

| [`Alignment`

`Options`](https://ej2.syncfusion.com/vue/documentation/api/diagram/alignmentOptions#AlignmentOptions) |`

Defines the specific direction, with respect to which the objects to be aligned. The accepted values of the argument "alignment options" are as follows.

**Left** Aligns all the selected objects at the left of the selection boundary.

**Right** Aligns all the selected objects at the right of the selection boundary.

**Center** Aligns all the selected objects at the center of the selection boundary.

**Top** Aligns all the selected objects at the top of the selection boundary.

**Bottom** Aligns all the selected objects at the bottom of the selection boundary.

**Middle** Aligns all the selected objects at the middle of the selection boundary.

|

| **Objects** | <p align="left">Defines the objects to be aligned. This is an optional parameter. By default, all the nodes and connectors in the selected region of the diagram gets aligned.</p> |

| [`Alignment`

`Mode`](https://ej2.syncfusion.com/vue/documentation/api/diagram/alignmentMode#AlignmentMode)`

|

Defines the specific mode, with respect to which the objects to be aligned. This is an optional parameter. The default alignment mode is `'Object'`.

The accepted values of the argument "alignment mode" are as follows.

**Object** Aligns the objects based on the first object in the selected list.

**Selector** Aligns the objects based on the selection boundary.

|

**APP.VUE**

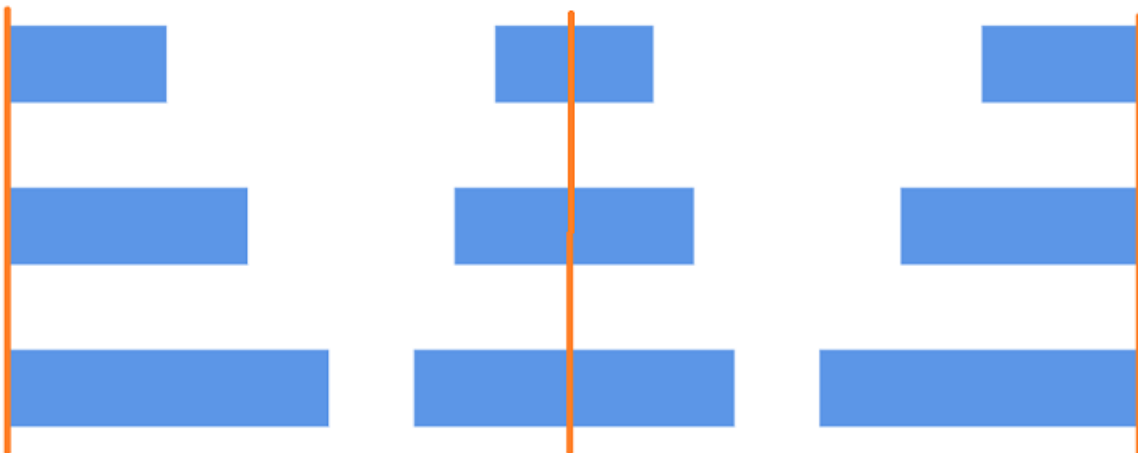
```
<template>
  <div id="app">
    <ejs-diagram id="diagram" :width='width' :height='height'
:nodes='nodes'></ejs-diagram>
  </div>
</template>
<script>
  import Vue from 'vue';
  import { DiagramPlugin ,NodeModel,ConnectorModel} from '@syncfusion/ej2-
vue-diagrams';
  Vue.use(DiagramPlugin);
  let nodes = [
    {
      id: 'node1',
      width: 90,
      height: 60,
      offsetX: 100,
      offsetY: 100,
      style: {
        fill: '#6BA5D7',
        strokeColor: 'white',
        strokeWidth: 1
      },
    },
    {
      id: 'node2',
      width: 100,
      height: 60,
      offsetX: 100,
      offsetY: 170,
      style: {
        fill: '#6BA5D7',
        strokeColor: 'white',
        strokeWidth: 1
      },
    },
    {
      id: 'node3',
      width: 140,
      height: 60,
      offsetX: 100,
      offsetY: 240,
      style: {
        fill: '#6BA5D7',
        strokeColor: 'white',
        strokeWidth: 1
      },
    },
  ];
  export default {
    name: 'app'
    data() {
      return {
        width: "100%",
        height: "350px",
      }
    }
  }
}
```

```

        nodes: nodes
    }
}
mounted: function() {
    let diagramInstance: Diagram;
    let diagramObj: any = document.getElementById("diagram");
    diagramInstance = diagramObj.ej2_instances[0];
    let selArray: (NodeModel | ConnectorModel)[] = [];
    selArray.push(diagramInstance.nodes[0],
diagramInstance.nodes[1], diagramInstance.nodes[2]);
    //Selects the nodes
    diagramInstance.select(selArray);
    //Sets direction as left
    diagramInstance.align('Left',
diagramInstance.selectedItems.nodes, 'Selector');
    diagramInstance.dataBind();
}
}
</script>
<style>
    @import "../node_modules/@syncfusion/ej2-vue-
diagrams/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/diagram/commands/align-cs1" %}



### Distribute

The [Distribute](#) commands enable to place the selected objects on the page at equal intervals from each other. The selected objects are equally spaced within the selection boundary.

The factor to distribute the shapes [DistributeOptions](#) are listed as follows:

- RightToLeft: Distributes the objects based on the distance between the right and left sides of the adjacent objects.
- Left: Distributes the objects based on the distance between the left sides of the adjacent objects.

- Right: Distributes the objects based on the distance between the right sides of the adjacent objects.
- Center: Distributes the objects based on the distance between the center of the adjacent objects.
- BottomToTop: Distributes the objects based on the distance between the bottom and top sides of the adjacent objects.
- Top: Distributes the objects based on the distance between the top sides of the adjacent objects.
- Bottom: Distributes the objects based on the distance between the bottom sides of the adjacent objects.
- Middle: Distributes the objects based on the distance between the vertical center of the adjacent objects.

The following code example illustrates how to execute the space commands.

#### APP.VUE

```
<template>
  <div id="app">
    <ejs-diagram id="diagram" :width='width' :height='height'
:nodes='nodes'></ejs-diagram>
  </div>
</template>
<script>
  import Vue from 'vue';
  import { DiagramPlugin ,NodeModel,ConnectorModel} from '@syncfusion/ej2-
vue-diagrams';
  Vue.use(DiagramPlugin);
  let nodes = [
    {
      id: 'node1',
      width: 90,
      height: 60,
      offsetX: 100,
      offsetY: 100,
      style: {
        fill: '#6BA5D7',
        strokeColor: 'white',
        strokeWidth: 1
      },
    },
    {
      id: 'node2',
      width: 90,
      height: 60,
      offsetX: 240,
      offsetY: 100,
      style: {
        fill: '#6BA5D7',
        strokeColor: 'white',
        strokeWidth: 1
      },
    },
    {
      id: 'node3',
```

```

        width: 90,
        height: 60,
        offsetX: 170,
        offsetY: 150,
        style: {
            fill: '#6BA5D7',
            strokeColor: 'white',
            strokeWidth: 1
        },
    },
];
export default {
    name: 'app'
    data() {
        return {
            width: "100%",
            height: "350px",
            nodes: nodes
        }
    }
    mounted: function() {
        let diagramInstance: Diagram;
        let diagramObj: any = document.getElementById("diagram");
        diagramInstance = diagramObj.ej2_instances[0];
        let selArray: (NodeModel | ConnectorModel)[] = [];
        selArray.push(diagramInstance.nodes[0],
        diagramInstance.nodes[1], diagramInstance.nodes[2]);
        //Selects the nodes
        diagramInstance.select(selArray);
        //Distributes space between the nodes
        diagramInstance.distribute('RightToLeft',
        diagramInstance.selectedItems.nodes);
    }
}
</script>
<style>
    @import "../../node_modules/@syncfusion/ej2-vue-
    diagrams/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/diagram/commands/distribute-cs1" %}





### Sizing

Sizing [sameSize](#) commands enable to equally size the selected nodes with respect to the first selected object.

[SizingOptions](#) are as follows:

- Width: Scales the width of the selected objects.
- Height: Scales the height of the selected objects.
- Size: Scales the selected objects both vertically and horizontally.

The following code example illustrates how to execute the size commands.

### APP.VUE

```
<template>
  <div id="app">
    <ejs-diagram id="diagram" :width='width' :height='height'
:nodes='nodes'></ejs-diagram>
  </div>
</template>
<script>
  import Vue from 'vue';
  import { DiagramPlugin ,NodeModel,ConnectorModel} from '@syncfusion/ej2-
vue-diagrams';
  Vue.use(DiagramPlugin);
  let nodes = [
    {
      id: 'node1',
      width: 90,
      height: 60,
      offsetX: 100,
      offsetY: 100,
      style: {
```

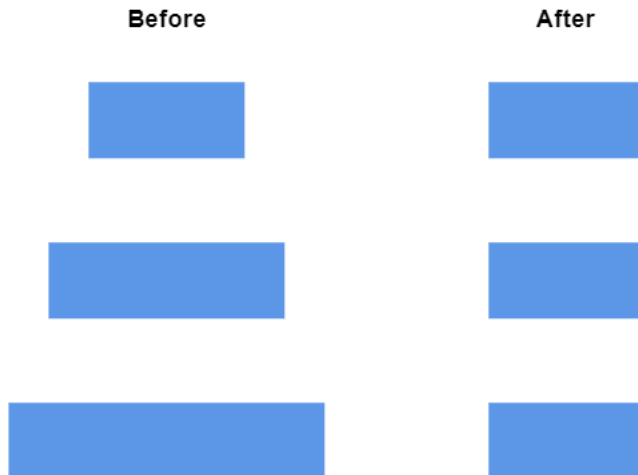
```

        fill: '#6BA5D7',
        strokeColor: 'white',
        strokeWidth: 1
      },
    },
  ],
  {
    id: 'node2',
    width: 100,
    height: 60,
    offsetX: 100,
    offsetY: 170,
    style: {
      fill: '#6BA5D7',
      strokeColor: 'white',
      strokeWidth: 1
    },
  },
],
{
  id: 'node3',
  width: 130,
  height: 60,
  offsetX: 100,
  offsetY: 230,
  style: {
    fill: '#6BA5D7',
    strokeColor: 'white',
    strokeWidth: 1
  },
},
]
];
export default {
  name: 'app'
  data() {
    return {
      width: "100%",
      height: "350px",
      nodes: nodes
    }
  }
  mounted: function() {
    let diagramInstance: Diagram;
    let diagramObj: any = document.getElementById("diagram");
    diagramInstance = diagramObj.ej2_instances[0];
    let selArray: (NodeModel | ConnectorModel)[] = [];
    selArray.push(diagramInstance.nodes[0],
    diagramInstance.nodes[1], diagramInstance.nodes[2]);
    //Selects the nodes
    diagramInstance.select(selArray);
    //Resizes the selected nodes with the same width
    diagramInstance.sameSize('Width',
    diagramInstance.selectedItems.nodes);
  }
}
</script>
<style>
  @import "../node_modules/@syncfusion/ej2-vue-
  diagrams/styles/material.css";

```

```
</style>
```

```
{% previewsample "page.domainurl/code-snippet/diagram/commands/sizing-cs1" %}
```



### Clipboard

Clipboard commands are used to cut, copy, or paste the selected elements. Refer to the following link which shows how to use clipboard methods in the diagram.

- Cuts the selected elements from the diagram to the diagram's clipboard, [cut](#).
- Copies the selected elements from the diagram to the diagram's clipboard, [copy](#).
- Pastes the diagram's clipboard data (nodes/connectors) into the diagram, [paste](#).

### APP.VUE

```
<template>
  <div id="app">
    <ejs-diagram id="diagram" :width='width' :height='height'
:nodes='nodes' :connectors='connectors' ></ejs-diagram>
  </div>
</template>
<script>
  import Vue from 'vue';
  import { DiagramPlugin ,NodeModel,ConnectorModel} from '@syncfusion/ej2-
vue-diagrams';
  Vue.use(DiagramPlugin);
  let nodes = [{
    id: 'node1',
    width: 90,
    height: 60,
    offsetX: 100,
    offsetY: 100,
    style: {
      fill: '#6BA5D7',
      strokeColor: 'white',
      strokeWidth: 1
    },
  },
```

```

    },
    {
      id: 'node2',
      width: 90,
      height: 60,
      offsetX: 240,
      offsetY: 100,
      style: {
        fill: '#6BA5D7',
        strokeColor: 'white',
        strokeWidth: 1
      },
    },
  ],
];

let connectors = [{
  id: 'connector1',
  sourceID: 'node1',
  targetID: 'node2',
  style: {
    strokeColor: '#6BA5D7',
    fill: '#6BA5D7',
    strokeWidth: 2,
    targetDecorator: {
      style: {
        fill: '#6BA5D7',
        strokeColor: '#6BA5D7'
      }
    }
  }
}]

export default {
  name: 'app'
  data() {
    return {
      width: "100%",
      height: "350px",
      nodes: nodes,
      connectors: connectors
    }
  }
  mounted: function() {
    let diagramInstance: Diagram;
    let diagramObj: any = document.getElementById("diagram");
    diagramInstance = diagramObj.ej2_instances[0];
    diagramInstance.select([diagramInstance.nodes[0],
diagramInstance.nodes[1], diagramInstance.connectors[0]]);
    //copies the selected nodes
    diagramInstance.copy();
    //pastes the copied objects
    diagramInstance.paste(diagramInstance.copy() as (NodeModel |
ConnectorModel) []);
  }
}
</script>
<style>
  @import "../node_modules/@syncfusion/ej2-vue-
diagrams/styles/material.css";

```

```
</style>
```

```
{% previewsample "page.domainurl/code-snippet/diagram/commands/clipboard-cs1" %}
```

### Grouping

**Grouping commands** are used to group/ungroup the selected elements on the diagram. Refer to the following link which shows how to use grouping commands in the diagram.

[Group](#) the selected nodes and connectors in the diagram.

[Ungroup](#) the selected nodes and connectors in the diagram.

The following code illustrates how to execute the grouping commands.

### APP.VUE

```
<template>
  <div id="app">
    <ejs-diagram id="diagram" :width='width' :height='height'
      :nodes='nodes' :connectors='connectors' ></ejs-diagram>
    </div>
  </template>
<script>
  import Vue from 'vue';
  import { DiagramPlugin ,NodeModel,ConnectorModel} from '@syncfusion/ej2-
vue-diagrams';
  Vue.use(DiagramPlugin);
  let nodes = [{
    id: 'node1',
    width: 100,
    height: 70,
    offsetX: 100,
    offsetY: 100,
    style: {
      fill: '#6BA5D7',
      strokeColor: 'white',
      strokeWidth: 1
    },
  },
  {
    id: 'node2',
    width: 100,
    height: 70,
    offsetX: 300,
    offsetY: 100,
    style: {
      fill: '#6BA5D7',
      strokeColor: 'white',
      strokeWidth: 1
    },
  },
  {
    id: 'node3',
    width: 100,
    height: 70,
    offsetX: 200,
    offsetY: 200,
```

```

        style: {
            fill: '#6BA5D7',
            strokeColor: 'white',
            strokeWidth: 1
        },
    },
    {
        id: 'group',
        children: ['node1', 'node2', 'connector1']
    },
    {
        id: 'group2',
        children: ['node3', 'group']
    }
];

let connectors = [{
    id: 'connector1',
    sourceID: 'node1',
    targetID: 'node2',
    style: {
        strokeColor: '#6BA5D7',
        fill: '#6BA5D7',
        strokeWidth: 2,
        targetDecorator: {
            style: {
                fill: '#6BA5D7',
                strokeColor: '#6BA5D7'
            }
        }
    }
}]

export default {
    name: 'app'
    data() {
        return {
            width: "100%",
            height: "350px",
            nodes: nodes,
            connectors: connectors
        }
    }
    mounted: function() {
        let diagramInstance: Diagram;
        let diagramObj: any = document.getElementById("diagram");
        diagramInstance = diagramObj.ej2_instances[0];
        diagramInstance.select([diagramInstance.nodes[0],
diagramInstance.nodes[1], diagramInstance.connectors[0]]);
        //Selects the diagram
        diagramInstance.selectAll();
        //Groups the selected elements.
        diagramInstance.group();
    }
}
</script>
<style>
    @import "../node_modules/@syncfusion/ej2-vue-
diagrams/styles/material.css";

```

```
</style>
```

```
{% previewsample "page.domainurl/code-snippet/diagram/commands/grouping-cs1" %}
```

### Z-Order command

**Z-Order commands** enable you to visually arrange the selected objects such as nodes and connectors on the page.

### *bringToFront* command

The [bringToFront](#) command visually brings the selected element to front over all the other overlapped elements. The following code illustrates how to execute the `bringToFront` command.

### APP.VUE

```
<template>
  <div id="app">
    <ejs-diagram id="diagram" :width='width' :height='height'
:nodes='nodes' ></ejs-diagram>
  </div>
</template>
<script>
  import Vue from 'vue';
  import { DiagramPlugin ,NodeModel,ConnectorModel} from '@syncfusion/ej2-
vue-diagrams';
  Vue.use(DiagramPlugin);
  let nodes = [{
    id: 'node1',
    width: 90,
    height: 60,
    offsetX: 100,
    offsetY: 100,
    style: {
      fill: '#6BA5D7',
      strokeColor: 'white',
      strokeWidth: 1
    },
  },
  {
    id: 'node2',
    width: 90,
    height: 60,
    offsetX: 240,
    offsetY: 100,
    style: {
      fill: '#6BA5D7',
      strokeColor: 'white',
      strokeWidth: 1
    },
  },
  {
    id: 'node3',
    width: 90,
    height: 60,
    offsetX: 160,
    offsetY: 90,
    style: {
```

```

        fill: '#6BA5D7',
        strokeColor: 'white',
        strokeWidth: 1
      },
    ],
  };
  export default {
    name: 'app'
    data() {
      return {
        width: "100%",
        height: "350px",
        nodes: nodes,
      }
    }
    mounted: function() {
      let diagramInstance: Diagram;
      let diagramObj: any = document.getElementById("diagram");
      diagramInstance = diagramObj.ej2_instances[0];
      let selArray: (NodeModel)[] = [];
      selArray.push(diagramInstance.nodes[2]);
      //Selects the nodes
      diagramInstance.select(selArray);
      //Brings to front
      diagramInstance.bringToFront();
    }
  }
</script>
<style>
  @import "../../node_modules/@syncfusion/ej2-vue-
  diagrams/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/diagram/commands/bringfront-cs1" %}

#### [sendToBack command](#)

The [sendToBack](#) command visually moves the selected element behind all the other overlapped elements. The following code illustrates how to execute the `sendToBack` command.

#### **APP.VUE**

```

<template>
  <div id="app">
    <ejs-diagram id="diagram" :width='width' :height='height'
    :nodes='nodes' ></ejs-diagram>
  </div>
</template>
<script>
  import Vue from 'vue';
  import { DiagramPlugin ,NodeModel,ConnectorModel} from '@syncfusion/ej2-
  vue-diagrams';
  Vue.use(DiagramPlugin);
  let nodes = [{
    id: 'node1',
    width: 90,
    height: 60,

```



```

        offsetX: 100,
        offsetY: 100,
        style: {
            fill: '#6BA5D7',
            strokeColor: 'white',
            strokeWidth: 1
        },
    },
    {
        id: 'node2',
        width: 90,
        height: 60,
        offsetX: 240,
        offsetY: 100,
        style: {
            fill: '#6BA5D7',
            strokeColor: 'white',
            strokeWidth: 1
        },
    },
    {
        id: 'node3',
        width: 90,
        height: 60,
        offsetX: 160,
        offsetY: 90,
        style: {
            fill: '#6BA5D7',
            strokeColor: 'white',
            strokeWidth: 1
        },
    },
    ]];
export default {
    name: 'app'
    data() {
        return {
            width: "100%",
            height: "350px",
            nodes: nodes,
        }
    }
    mounted: function() {
        let diagramInstance: Diagram;
        let diagramObj: any = document.getElementById("diagram");
        diagramInstance = diagramObj.ej2_instances[0];
        let selArray: (NodeModel)[] = [];
        selArray.push(diagramInstance.nodes[2]);
        //Selects the nodes
        diagramInstance.select(selArray);
        //Sends to back
        diagramInstance.sendToBack();
    }
}
</script>
<style>
    @import "../node_modules/@syncfusion/ej2-vue-
    diagrams/styles/material.css";

```

```
</style>
```

```
{% previewsample "page.domainurl/code-snippet/diagram/commands/sendback-cs1" %}
```

### *moveForward command*

The [moveForward](#) command visually moves the selected element over the nearest overlapping element. The following code illustrates how to execute the `moveForward` command.

### **APP.VUE**

```
<template>
  <div id="app">
    <ejs-diagram id="diagram" :width='width' :height='height'
:nodes='nodes' ></ejs-diagram>
  </div>
</template>
<script>
  import Vue from 'vue';
  import { DiagramPlugin ,NodeModel,ConnectorModel} from '@syncfusion/ej2-
vue-diagrams';
  Vue.use(DiagramPlugin);
  let nodes = [{
    id: 'node1',
    width: 90,
    height: 60,
    offsetX: 100,
    offsetY: 100,
    style: {
      fill: '#6BA5D7',
      strokeColor: 'white',
      strokeWidth: 1
    },
  },
  {
    id: 'node2',
    width: 90,
    height: 60,
    offsetX: 180,
    offsetY: 100,
    style: {
      fill: '#6BA5D7',
      strokeColor: 'white',
      strokeWidth: 1
    },
  },
  ];
  export default {
    name: 'app'
    data() {
      return {
        width: "100%",
        height: "350px",
        nodes: nodes,
      }
    }
    mounted: function() {
      let diagramInstance: Diagram;
```

```

        let diagramObj: any = document.getElementById("diagram");
        diagramInstance = diagramObj.ej2_instances[0];
        let selArray: (NodeModel)[] = [];
        selArray.push(diagramInstance.nodes[1]);
        //Selects the nodes
        diagramInstance.select(selArray);
        //Moves forward
        diagramInstance.moveForward();
    }
}
</script>
<style>
    @import "../../node_modules/@syncfusion/ej2-vue-
    diagrams/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/diagram/commands/moveforward-cs1" %}

#### [sendBackward command](#)

The [sendBackward](#) command visually moves the selected element behind the underlying element. The following code illustrates how to execute the `sendBackward` command.

#### **APP.VUE**

```

<template>
    <div id="app">
        <ejs-diagram id="diagram" :width='width' :height='height'
        :nodes='nodes' ></ejs-diagram>
    </div>
</template>
<script>
    import Vue from 'vue';
    import { DiagramPlugin ,NodeModel,ConnectorModel} from '@syncfusion/ej2-
    vue-diagrams';
    Vue.use(DiagramPlugin);
    let nodes = [{
        id: 'node1',
        width: 90,
        height: 60,
        offsetX: 100,
        offsetY: 100,
        style: {
            fill: '#6BA5D7',
            strokeColor: 'white',
            strokeWidth: 1
        },
    },
    {
        id: 'node2',
        width: 90,
        height: 60,
        offsetX: 180,
        offsetY: 100,
        style: {
            fill: '#6BA5D7',
            strokeColor: 'white',

```

```

        strokeWidth: 1
      },
    ]];
  export default {
    name: 'app'
    data() {
      return {
        width: "100%",
        height: "350px",
        nodes: nodes,
      }
    }
    mounted: function() {
      let diagramInstance: Diagram;
      let diagramObj: any = document.getElementById("diagram");
      diagramInstance = diagramObj.ej2_instances[0];
      let selArray: (NodeModel)[] = [];
      selArray.push(diagramInstance.nodes[1]);
      diagramInstance.select(selArray);
      //Sends backward
      diagramInstance.sendBackward();
    }
  }
</script>
<style>
  @import "../../node_modules/@syncfusion/ej2-vue-
  diagrams/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/diagram/commands/sendbackward-cs1" %}

### Zoom

The [zoom](#) command is used to zoom-in and zoom-out the diagram view.

The following code illustrates how to zoom-in/zoom out the diagram.

,

```

<template>
<div id="app">
  <ejs-diagram id="diagram" :width='width' :height='height' ></ejs-diagram>
</div>
</template>
<script>
import Vue from 'vue';
import { DiagramPlugin ,NodeModel,ConnectorModel} from '@syncfusion/ej2-vue-diagrams';
Vue.use(DiagramPlugin);
export default {
  name: 'app'

```

```

data() {
  return {
    width: "100%",
    height: "350px",
  }
}

mounted: function() {
  let diagramInstance: Diagram;
  let diagramObj: any = document.getElementById("diagram");
  diagramInstance = diagramObj.ej2_instances[0];
  diagramInstance.zoom(1.2, {
    x: 100,
    y: 100
  });
}
}

</script>
<style>
@import "../node_modules/@syncfusion/ej2-vue-diagrams/styles/material.css";
</style>

```

### Nudge command

The [nudge](#) commands move the selected elements towards up, down, left, or right by 1 pixel.

[NudgeDirection](#) nudge command moves the selected elements towards the specified direction by 1 pixel, by default.

The accepted values of the argument "direction" are as follows:

- Up: Moves the selected elements towards up by the specified delta value.
- Down: Moves the selected elements towards down by the specified delta value.
- Left: Moves the selected elements towards left by the specified delta value.
- Right: Moves the selected elements towards right by the specified delta value.

The following code illustrates how to execute nudge command.

```

<template>
<div id="app">

```

```

<ejs-diagram id="diagram" :width='width' :height='height' ></ejs-diagram>
</div>
</template>
<script>
import Vue from 'vue';
import { DiagramPlugin ,NodeModel,ConnectorModel} from '@syncfusion/ej2-vue-diagrams';
Vue.use(DiagramPlugin);
export default {
  name: 'app'
  data() {
    return {
      width: "100%",
      height: "350px",
    }
  }
  mounted: function() {
    let diagramInstance: Diagram;
    let diagramObj: any = document.getElementById("diagram");
    diagramInstance = diagramObj.ej2_instances[0];
    diagramInstance.nudge('Right');
  }
}
</script>
<style>
@import "../../node_modules/@syncfusion/ej2-vue-diagrams/styles/material.css";
</style>
`

```

### Nudge by using arrow keys

The corresponding arrow keys are used to move the selected elements towards up, down, left, or right direction by 1 pixel.



Nudge commands are particularly useful for accurate placement of elements.

### BringIntoView

The [bringIntoView](#) command brings the specified rectangular region into the viewport of the diagram.

The following code illustrates how to execute the `bringIntoView` command.

```
,  
  
<template>  
<div id="app">  
<ejs-diagram id="diagram" :width='width' :height='height' ></ejs-diagram>  
</div>  
</template>  
<script>  
import Vue from 'vue';  
import { DiagramPlugin ,NodeModel,ConnectorModel,Rect} from '@syncfusion/ej2-vue-diagrams';  
Vue.use(DiagramPlugin);  
export default {  
  name: 'app'  
  data() {  
    return {  
      width: "100%",  
      height: "350px",  
    }  
  }  
  mounted: function() {  
    let diagramInstance: Diagram;  
    let diagramObj: any = document.getElementById("diagram");  
    diagramInstance = diagramObj.ej2_instances[0];  
    //Brings the specified rectangular region of the Diagram content to the viewport of the page.  
    let bound: Rect = new Rect(200, 400, 500, 400);  
    diagramInstance.bringIntoView(bound);  
  }  
}  
</script>  
<style>  
@import "../node_modules/@syncfusion/ej2-vue-diagrams/styles/material.css";
```

</style>

,

### BringToCenter

The [bringToCenter](#) command brings the specified rectangular region of the diagram content to the center of the viewport.

The following code illustrates how to execute the `bringToCenter` command.

,

```
<template>
```

```
<div id="app">
```

```
<ejs-diagram id="diagram" :width='width' :height='height' ></ejs-diagram>
```

```
</div>
```

```
</template>
```

```
<script>
```

```
import Vue from 'vue';
```

```
import { DiagramPlugin ,NodeModel,ConnectorModel,Rect} from '@syncfusion/ej2-vue-diagrams';
```

```
Vue.use(DiagramPlugin);
```

```
export default {
```

```
  name: 'app'
```

```
  data() {
```

```
    return {
```

```
      width: "100%",
```

```
      height: "350px",
```

```
    }
```

```
  }
```

```
  mounted: function() {
```

```
    let diagramInstance: Diagram;
```

```
    let diagramObj: any = document.getElementById("diagram");
```

```
    diagramInstance = diagramObj.ej2_instances[0];
```

```
    //Brings the specified rectangular region of the Diagram content to the center of the viewport.
```

```
    let bound: Rect = new Rect(200, 400, 500, 400);
```

```
    diagramInstance.bringToCenter(bound);
```

```
  }
```

```
}
```

```
</script>
```



```
<style>
@import "../node_modules/@syncfusion/ej2-vue-diagrams/styles/material.css";
</style>
`
```

### FitToPage command

The [fitToPage](#) command helps to fit the diagram content into the view with respect to either width, height, or at the whole.

The [mode](#) parameter defines whether the diagram has to be horizontally/vertically fit into the viewport with respect to width, height, or entire bounds of the diagram.

The [region](#) parameter defines the region that has to be drawn as an image.

The [margin](#) parameter defines the region/bounds of the diagram content that is to be fit into the view.

The [canZoomIn](#) parameter enables/disables zooming to fit the smaller content into a larger viewport.

The [customBounds](#) parameter the custom region that has to be fit into the viewport.

The following code illustrates how to execute **FitToPage** command.

```
`
<template>
<div id="app">
<ejs-diagram id="diagram" :width='width' :height='height' ></ejs-diagram>
</div>
</template>
<script>
import Vue from 'vue';
import { DiagramPlugin ,NodeModel,ConnectorModel,Rect} from '@syncfusion/ej2-vue-diagrams';
Vue.use(DiagramPlugin);
export default {
name: 'app'
data() {
return {
width: "100%",
height: "350px",
}
}
mounted: function() {
let diagramInstance: Diagram;
```

```

let diagramObj: any = document.getElementById("diagram");
diagramInstance = diagramObj.ej2_instances[0];
//fit the diagram to the page with respect to mode and region
diagramInstance.fitToPage({
  mode: 'Page',
  region: 'Content',
  margin: {
    bottom: 50
  },
  canZoomIn: false
});
}
}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-vue-diagrams/styles/material.css";
</style>
`

```

### Command manager

Diagram provides support to map/bind command execution with desired combination of key gestures. Diagram provides some built-in commands.

[CommandManager](#) provides support to define custom commands. The custom commands are executed, when the specified key gesture is recognized.

### Custom command

To define a custom command, specify the following properties:

- [execute](#): A method to be executed.
- [canExecute](#): A method to define whether the command can be executed at the moment.
- [gesture](#): A combination of [keys](#) and [KeyModifiers](#).
- [parameter](#): Defines any additional parameters that are required at runtime.
- [name](#): Defines the name of the command.

To explore the properties of custom commands, refer to [Commands](#).

The following code example illustrates how to define a custom command.

```

`
<template>
<div id="app">

```

```
<ejs-diagram id="diagram" :width='width' :height='height' :commandManager='commandManager'
></ejs-diagram>
</div>
</template>
<script>
import Vue from 'vue';
import { DiagramPlugin ,Keys,KeyModifiers} from '@syncfusion/ej2-vue-diagrams';
Vue.use(DiagramPlugin);
export default {
  name: 'app'
  data() {
    return {
      width: "100%",
      height: "350px",
      commandManager: {
        commands: [{
          name: 'customCopy',
          parameter: 'node',
          //Method to define whether the command can be executed at the current moment
          canExecute: function() {
            //Defines that the clone command can be executed, if and only if the selection list is not empty.
            if (diagramInstance.selectedItems.nodes.length > 0 || diagramInstance.selectedItems.connectors.length
            > 0) {
              return true;
            }
            return false;
          },
          //Command handler
          execute: function() {
            //Logic to clone the selected element
            diagramInstance.copy();
            diagramInstance.paste();
            diagramInstance.dataBind();
          },
        }
      ]
    }
  }
}
```

//Defines that the clone command has to be executed on the recognition of key press.

```
gesture: {
  key: Keys.G,
  keyModifiers: KeyModifiers.Shift | KeyModifiers.Alt
}
}},
},
}
}

mounted: function() {
  let diagramInstance: Diagram;
  let obj: any = document.getElementById("diagram");
  diagramInstance = obj.ej2_instances[0];
}
}

</script>
<style>
@import "../node_modules/@syncfusion/ej2-vue-diagrams/styles/material.css";
</style>
`
```

### Modify the existing command

When any one of the default commands is not desired, they can be disabled. To change the functionality of a specific command, the command can be completely modified.

The following code example illustrates how to disable a command and how to modify the built-in commands.

```
`
<template>
<div id="app">
<ejs-diagram id="diagram" :width='width' :height='height' :commandManager='commandManager'
></ejs-diagram>
</div>
</template>
<script>
import Vue from 'vue';
```

```
import {
  DiagramPlugin,
  Keys,
  KeyModifiers
} from '@syncfusion/ej2-vue-diagrams';
Vue.use(DiagramPlugin);
export default {
  name: 'app'
  data() {
    return {
      width: "100%",
      height: "350px",
      commandManager: {
        commands: [
          {
            name: 'nudgeUp',
            canExecute: function () {
              return false;
            },
            gesture: {
              key: Keys.Up,
            }
          },
          {
            name: 'nudgeDown',
            canExecute: function () {
              return false;
            },
            gesture: {
              key: Keys.Down,
            }
          },
          {
```

```

name: 'nudgeRight',
canExecute: function () {
return false;
},
gesture: {
key: Keys.Right,
}
}
]
},
}
}
mounted: function() {
let diagramInstance: Diagram;
let obj: any = document.getElementById("diagram");
diagramInstance = obj.ej2_instances[0];
}
}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-vue-diagrams/styles/material.css";
</style>
`

```

See Also

- [How to create the custom context menu items](#)

### Undo redo in Vue Diagram component

Diagram tracks the history of actions that are performed after initializing the diagram and provides support to reverse and restore those changes.

#### Undo and redo

Diagram provides built-in support to track the changes that are made through interaction and through public APIs. The changes can be reverted or restored either through shortcut keys or through commands.

Note: If you want to use Undo-Redo in diagram, you need to inject UndoRedo in the diagram.

### Undo/redo through shortcut keys

Undo/redo commands can be executed through shortcut keys. Shortcut key for undo is Ctrl+z and shortcut key for redo is Ctrl+y.

### Undo/redo through public APIs

The client-side methods [undo](#) and [redo](#) help you to revert/restore the changes. The following code example illustrates how to undo/redo the changes through script.

```
`javascript
export default {
  name: 'app'
  data() {
    return {
      width: "100%",
      height: "350px",
    }
  }
  mounted: function() {
    let diagramInstance: Diagram;
    let diagramObj: any = document.getElementById("diagram");
    diagramInstance = diagramObj.ej2_instances[0];
    // Reverts the last action performed
    diagramInstance.undo();
    // Restores the last undone action
    diagramInstance.redo();
  }
}
```

When a change in the diagram is reverted or restored (undo/redo), the historyChange event gets triggered.

### Group multiple changes

History list allows to revert or restore multiple changes through a single undo/redo command. For example, revert/restore the fill color change of multiple elements at a time.

The client-side method [startGroupAction](#) is used to notify the diagram to start grouping the changes. The client-side method [endGroupAction](#) is used to notify to stop grouping the changes. The following code illustrates how to undo/redo fillColor change of multiple elements at a time.

### APP.VUE

```
<template>
```

```

<div id="app">
  <ejs-diagram id="diagram" :width='width' :height='height'
:nodes='nodes' :getNodeDefaults='getNodeDefaults' ></ejs-diagram>
</div>
</template>
<script>
  import Vue from 'vue';
  import { DiagramPlugin ,UndoRedo,Diagram} from '@syncfusion/ej2-vue-
diagrams';
  Diagram.Inject(UndoRedo);
  Vue.use(DiagramPlugin);
  let nodes = [{
    id: 'Start',
    width: 140,
    height: 50,
    offsetX: 300,
    offsetY: 50,
    annotations: [{
      id: 'labell',
      content: 'Start'
    }],
    shape: {
      type: 'Flow',
      shape: 'Terminator'
    }
  }]
  export default {
    name: 'app'
    data() {
      return {
        width: "100%",
        height: "350px",
        nodes: nodes,
        getNodeDefaults: (node) => {
          node.height = 100;
          node.width = 100;
          node.style.fill = '#6BA5D7';
          node.style.strokeColor = 'white';
          return node;
        },
      }
    }
    mounted: function() {
      let diagramInstance: Diagram;
      let diagramObj: any = document.getElementById("diagram");
      diagramInstance = diagramObj.ej2_instances[0];
      //Start to group the changes
      diagramInstance.startGroupAction();
      //Makes the changes
      let color: string[] = ['black', 'red', 'green', 'yellow']
      for (var i = 0; i < color.length; i++) {
        // Updates the fillColor for all the child elements.
        diagramInstance.nodes[0].style.fill = color[i];
        diagramInstance.dataBind();
      }
      //Ends grouping the changes
      diagramInstance.endGroupAction();
    }
  }

```



```

    }
  }
</script>
<style>
  @import "../node_modules/@syncfusion/ej2-vue-
  diagrams/styles/material.css";
</style>

```

```
{% previewsample "page.domainurl/code-snippet/diagram/undo-redo/undo-redo-cs1" %}
```

#### *Track custom changes*

Diagram provides options to track the changes that are made to custom properties. For example, in case of an employee relationship diagram, track the changes in the employee information. The historyList of the diagram enables you to track such changes.

The following example illustrates how to track such custom property changes.

Before changing the employee information, save the existing information to historyList by using the client-side method push of historyList.

The historyList canLog method can be used which takes a history entry as argument and returns whether the specific entry can be added or not.

The following code example illustrates how to save the existing property values.

```

`javascript
export default {
  name: 'app'
  data() {
    return {
      width: "100%",
      height: "350px",
    }
  }
  mounted: function() {
    let diagramInstance: Diagram;
    let diagramObj: any = document.getElementById("diagram");
    diagramInstance = diagramObj.ej2_instances[0];
    //Creates a custom entry
    let entry: HistoryEntry = {
      undoObject: diagramInstance.nodes[0];
    };
    // adds that to history list
    diagramInstance.historyList.push(entry);

```

```

diagramInstance.dataBind();
}
}
,

```

### canLog

canLog in the history list, which takes a history entry as argument and returns whether the specific entry can be added or not.

### APP.VUE

```

<template>
  <div id="app">
    <ejs-diagram id="diagram" :width='width' :height='height'
:nodes='nodes'
:getNodeDefaults='getNodeDefaults'></ejs-diagram>
  </div>
</template>
<script>
  import Vue from 'vue';
  import { DiagramPlugin,UndoRedo ,Diagram, HistoryEntry} from
'@syncfusion/ej2-vue-diagrams';
  Diagram.Inject(UndoRedo);
  Vue.use(DiagramPlugin);
  let nodes = [{
    id: 'Start',
    width: 140,
    height: 50,
    offsetX: 300,
    offsetY: 50,
    annotations: [{
      id: 'labell',
      content: 'Start'
    }],
    shape: {
      type: 'Flow',
      shape: 'Terminator'
    }
  }]
  export default {
    name: 'app',
    data() {
      return {
        width: "100%",
        height: "350px",
        nodes: nodes,
        getNodeDefaults: (node) => {
          node.height = 100;
          node.width = 100;
          node.style.fill = '#6BA5D7';
          node.style.strokeColor = 'white';
          return node;
        },
      }
    },
  },

```

```

    mounted: function() {
      let diagramInstance: Diagram;
      let diagramObj: any = document.getElementById("diagram");
      diagramInstance = diagramObj.ej2_instances[0];
      // canLog decide whether the entry add or not in history List
      diagramInstance.historyManager.canLog = function(entry:
HistoryEntry) {
        entry.cancel = true;
        return entry;
      }
    }
  }
</script>
<style>
  @import "../node_modules/@syncfusion/ej2-vue-
diagrams/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/diagram/undo-redo/undo-redo-cs2" %}

#### [Track undo/redo actions](#)

The historyList undoStack property is used to get the collection of undo actions which should be performed in the diagram.

The undoStack/redoStack is the read-only property.

```

`javascript
export default {
  name: 'app'
  data() {
    return {
      width: "100%",
      height: "350px",
    }
  }
  mounted: function() {
    let diagramInstance: Diagram;
    let diagramObj: any = document.getElementById("diagram");
    diagramInstance = diagramObj.ej2_instances[0];
    //get the collection of undoStack objects
    let undoStack = diagramInstance.historyList.undoStack;
    //get the collection of redoStack objects
    let redoStack = diagramInstance.historyList.redoStack;
  }
}

```

```

}
`

```

### History change event

The [historyChange](#) event triggers, whenever the interaction of the node and connector is take place.

```

`javascript
export default {
  name: 'app'
  data() {
    return {
      width: "100%",
      height: "350px",
    }
  }
  mounted: function() {
    let diagramInstance: Diagram;
    let diagramObj: any = document.getElementById("diagram");
    diagramInstance = diagramObj.ej2_instances[0];
    diagramInstance.historyChange = (arg: IHistoryChangeArgs) => {
      //causes of history change
      let cause: string = arg.cause;
    }
  }
}
`

```

### Stack limit

The [stackLimit](#) property of history manager is used to limits the number of actions to be stored on the history manager.

### **APP.VUE**

```

<template>
  <div id="app">
    <ejs-diagram id="diagram" :width='width' :height='height'
:nodes='nodes'
:getNodeDefaults='getNodeDefaults'></ejs-diagram>
  </div>
</template>
<script>
  import Vue from 'vue';

```

```

import { DiagramPlugin,UndoRedo ,Diagram} from '@syncfusion/ej2-vue-
diagrams';
Diagram.Inject(UndoRedo);
Vue.use(DiagramPlugin);
let nodes = [{
  id: 'Start',
  width: 140,
  height: 50,
  offsetX: 300,
  offsetY: 50,
  annotations: [{
    id: 'label1',
    content: 'Start'
  }],
  shape: {
    type: 'Flow',
    shape: 'Terminator'
  }
}]
export default {
  name: 'app'
  data() {
    return {
      width: "100%",
      height: "350px",
      nodes: nodes,
      getNodeDefaults: (node) => {
        node.height = 100;
        node.width = 100;
        node.style.fill = '#6BA5D7';
        node.style.strokeColor = 'white';
        return node;
      },
      historyManager:{
        stackLimit:2
      }
    }
  }
}
</script>
<style>
  @import "../node_modules/@syncfusion/ej2-vue-
diagrams/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/diagram/undo-redo/undo-redo-cs3" %}

### Retain selection

You can retain a selection at undo/redo operation by using the client-side API Method called **updateSelection**. Using this method, we can select a diagram objects.

`ts

```
let diagramInstance: DiagramComponent;
```

```
ReactDOM.render( < DiagramComponent id = "diagram" ref={diagram => diagramInstance = diagram}
```

```

width = {
  '100%'
}
height = {
  '600px'
}
nodes = {
  nodes
}
/>, document.getElementById("diagram") );
// history change event
diagramInstance.updateSelection: (object: NodeModel, diagram: Diagram) => {
  let selArr = [];
  selArr.push(object)
  diagram.select(selArr);
},
`

```

## Virtualization in Vue Diagram component

### Virtualization in Diagram

Virtualization is the process of loading the diagramming objects available in the visible area of the Diagram control, that is, only the diagramming objects that lie within the ViewPort of the Scroll Viewer are loaded (remaining objects are loaded only when they come into view).

This feature gives an optimized performance while loading and dragging items to the Diagram that consists of many Nodes and Connectors.

The following code illustrates how to enable Virtualization mode in the diagram.

```

`javascript
export default {
  name: 'app'
  data() {
    return {
      width: "100%",
      height: "350px",
      connectors: connectors,
    }
  }
}
//Enable virtualization in diagram
`

```

```
constraints: DiagramConstraints.Default | DiagramConstraints.Virtualization
```

```
}
```

```
}
```

```
}
```

```
,
```

## Serialization in Vue Diagram component

**Serialization** is the process of saving and loading for state persistence of the diagram.

### Save

The diagram is serialized as string while saving. The client-side method, [saveDiagram](#) helps to serialize the diagram as a string. The following code illustrates how to save the diagram.

```
`html
<template>
<div id="app">
<ejs-diagram ref="diagramObject" id="diagram" :width='width' :height='height'></ejs-diagram>
</div>
</template>
<script>
export default {
name: 'app'
data() {
return {
width: "100%",
height: "350px",
}
},
mounted: function() {
let diagramInstance: Diagram = this.$refs.diagramObject.ej2Instances;
let saveData: string;
//returns serialized string of the Diagram
saveData = diagramInstance.saveDiagram();
}
}
</script>
`
```

This string can be converted to JSON data and stored for future use. The following snippet illustrates how to save the serialized string into local storage.

```
`javascript
//Saves the string in to local storage
localStorage.setItem('fileName', saveData);
saveData = localStorage.getItem('fileName');
`
```

Diagram can also be saved as raster or vector image files. For more information about saving the diagram as images, refer to [Print and Export](#).

#### Load

Diagram is loaded from the serialized string data by client-side method, [loadDiagram](#).

The following code illustrates how to load the diagram from serialized string data.

```
`html
<template>
<div id="app">
<ejs-diagram ref="diagramObject" id="diagram" :width='width' :height='height'></ejs-diagram>
</div>
</template>
<script>
export default {
name: 'app'
data() {
return {
width: "100%",
height: "350px",
}
},
mounted: function() {
let diagram: Diagram = this.$refs.diagramObject.ej2Instances;
//Loads the Diagram from saved json data
diagram.loadDiagram(saveData);
}
}
</script>
```



`

Note: Before loading a new diagram, existing diagram is cleared.

#### Prevent default values

The [preventDefaults](#) property of `serializationSettings` is used to simplifying the saved JSON object without adding the default properties that are presented in the diagram.

The following code illustrates how to simplify the JSON object.

`ts

```
let diagram: Diagram = new Diagram({
  serializationSettings: { preventDefaults: true },
});
```

#### Export in Vue Diagram component

Diagram provides support to export its content as image/svg files. The client-side method [exportDiagram](#) helps to export the diagram. The following code illustrates how to export the diagram as image.

Note: To use Print and Export, you need to inject `PrintAndExport` in the diagram.

<!-- markdownlint-disable MD033 -->

```
`javascript
export default {
  name: 'app'
  data() {
    return {
      width: "100%",
      height: "350px",
      tool: DiagramTools.DrawOnce || DiagramTools.ZoomPan,
    }
  }
  mounted: function() {
    let diagramInstance: Diagram;
    let diagramObj: any = document.getElementById("diagram");
    diagramInstance = diagramObj.ej2_instances[0];
    let options: IExportOptions = {};
    options.mode = 'Download';
    diagramInstance.exportDiagram(options);
  }
}
```

```

}
`

```

### Exporting options

Diagram provides support to export the desired region of the diagram to desired formats.

#### File Name

[FileName](#) is the name of the file to be downloaded. By default, the file name is set to **Diagram**.

#### Format

[Format](#) is to specify the type/format of the exported file. By default, the diagram is exported as .jpg format. You can export diagram to the following formats:

- JPG
- PNG
- BMP
- SVG

```

`javascript
export default {
  name: 'app'
  data() {
    return {
      width: "100%",
      height: "350px",
      tool: DiagramTools.DrawOnce || DiagramTools.ZoomPan,
    }
  }
  mounted: function() {
    let diagramInstance: Diagram;
    let diagramObj: any = document.getElementById("diagram");
    diagramInstance = diagramObj.ej2_instances[0];
    let options: IExportOptions = {};
    options.mode = 'Download';
    options.format = 'SVG';
    diagramInstance.exportDiagram(options);
  }
}
`

```

### Margin

[Margin](#) specifies the amount of space that has to be left around the diagram.

<!-- markdownlint-disable MD033 -->

```
`javascript
export default {
  name: 'app'
  data() {
    return {
      width: "100%",
      height: "350px",
      tool: DiagramTools.DrawOnce || DiagramTools.ZoomPan,
    }
  }
  mounted: function() {
    let diagramInstance: Diagram;
    let diagramObj: any = document.getElementById("diagram");
    diagramInstance = diagramObj.ej2_instances[0];
    let options: IExportOptions = {};
    options.mode = 'Download';
    options.margin = {
      left: 10,
      right: 10,
      top: 10,
      bottom: 10
    };
    options.fileName = 'format';
    options.format = 'SVG';
    diagramInstance.exportDiagram(options);
  }
}
```

### Mode

[Mode](#) specifies whether the diagram will be exported as files or get base64 data (ImageURL/SVG). The export options are as follows:

- Download: Exports and downloads the diagram as image/SVG.
- Data: return a base64 string.

The following code example illustrates how to export the diagram as raw data.

```
`javascript
export default {
  name: 'app'
  data() {
    return {
      width: "100%",
      height: "350px",
      tool: DiagramTools.DrawOnce || DiagramTools.ZoomPan,
    }
  }
  mounted: function() {
    let diagramInstance: Diagram;
    let diagramObj: any = document.getElementById("diagram");
    diagramInstance = diagramObj.ej2_instances[0];
    let options: IExportOptions = {};
    options.mode = 'Data';
    options.margin = {
      left: 10,
      right: 10,
      top: 10,
      bottom: 10
    };
    options.fileName = 'format';
    options.format = 'SVG';
    let base64data = diagramInstance.exportDiagram(options);
  }
}
```

### Region

You can export any particular [region](#) of the diagram and it is categorized into three types as follows.

- PageSettings
- Content
- CustomBounds

### PageSettings

Diagram is exported based on the given PageSettings width and height. The Properties available in page settings are as follows.

- width
- height
- margin
- orientation
- boundaryConstraints
- background
- multiplePage
- showPageBreaks
- fitOptions

### boundaryConstraints

Defines the editable region of the diagram.

- Infinity - Allow the interactions to take place at infinite height and width.
- Diagram - Allow the interactions to take place around the diagram's height and width.
- Page - Allow the interactions to take place around the page's height and width.

### multiplePage

While setting multiple pages as false, the diagram is exported as a single image and while setting multiple pages as true, the diagram is exported as a separate image based on width and height.

The following code example illustrates how to export the region occupied by the diagram elements.

```
`javascript
export default {
  name: 'app'
  data() {
    return {
      width: "100%",
      height: "350px",
      tool: DiagramTools.DrawOnce || DiagramTools.ZoomPan,
    }
  }
  mounted: function() {
    let diagramInstance: Diagram;
```

```

let diagramObj: any = document.getElementById("diagram");
diagramInstance = diagramObj.ej2_instances[0];
let options: IExportOptions = {};
options.mode = 'Download';
options.margin = {
left: 10,
right: 10,
top: 10,
bottom: 10
};
options.fileName = 'format';
options.format = 'SVG';
options.region = 'Content';
diagram.exportDiagram(options);
}
}
`

```

### Content

The diagram content alone will be exported as an image.

The following code example illustrates how to export the region occupied by the diagram elements.

```

`javascript
var diagram = new ej.diagrams.Diagram({
width: 1500, height: 1500
}, '#element');
var options = {};
options.mode = 'Download';
options.margin = { left: 10, right: 10, top: 10, bottom: 10};
options.fileName = 'format';
options.format = 'SVG';
options.region = 'PageSettings';
diagram.exportDiagram(options);
`

```

### Custom bounds

Diagram provides support to export any specific region of the diagram by using [bounds](#).

The following code example illustrates how to export the region occupied by the diagram elements.

```
`javascript
export default {
  name: 'app'
  data() {
    return {
      width: "100%",
      height: "350px",
      tool: DiagramTools.DrawOnce || DiagramTools.ZoomPan,
    }
  }
  mounted: function() {
    let diagramInstance: Diagram;
    let diagramObj: any = document.getElementById("diagram");
    diagramInstance = diagramObj.ej2_instances[0];
    let options: IExportOptions = {};
    options.mode = 'Download';
    options.margin = {
      left: 10,
      right: 10,
      top: 10,
      bottom: 10
    };
    options.fileName = 'region';
    options.format = 'SVG';
    options.region = 'CustomBounds';
    options.bounds.x = 10;
    options.bounds.y = 10;
    options.bounds.height = 100;
    options.bounds.width = 100;
    diagramInstance.exportDiagram(options);
  }
}
```

,

### Export diagram with stretch option

Diagram provides support to export the diagram as image for [stretch](#) option. The exported images will be clearer but larger in file size.

The following code example illustrates how to export the region occupied by the diagram elements.

```
`javascript
export default {
  name: 'app'
  data() {
    return {
      width: "100%",
      height: "350px",
      tool: DiagramTools.DrawOnce || DiagramTools.ZoomPan,
    }
  }
  mounted: function() {
    let diagramInstance: Diagram;
    let diagramObj: any = document.getElementById("diagram");
    diagramInstance = diagramObj.ej2_instances[0];
    let options: IExportOptions = {};
    options.mode = 'Download';
    options.margin = {
      left: 10,
      right: 10,
      top: 10,
      bottom: 10
    };
    options.fileName = 'region';
    options.format = 'SVG';
    options.region = 'Content';
    options.stretch = 'Stretch';
    diagram.exportDiagram(options);
  }
}
```



## Print

The client-side method [print](#) helps to print the diagram as image.

Name	Type	Description
region	enum	Sets the region of the diagram to be printed.
bounds	object	Prints any custom region of diagram.
stretch	enum	Resizes the diagram content to fill its allocated space and printed.
multiplePage	boolean	Prints the diagram into multiple pages.
pageWidth	number	Sets the page width of the diagram while printing the diagram into multiple pages.
pageHeight	number	Sets the page height of the diagram while printing the diagram into multiple pages.
pageOrientation	enum	Sets the orientation of the page.

The following code example illustrates how to export the region occupied by the diagram elements.

```

`javascript
export default {
  name: 'app'
  data() {
    return {
      width: "100%",
      height: "350px",
      tool: DiagramTools.DrawOnce || DiagramTools.ZoomPan,
    }
  }
  mounted: function() {
    let diagramInstance: Diagram;
    let diagramObj: any = document.getElementById("diagram");
    diagramInstance = diagramObj.ej2_instances[0];
    let options: IExportOptions = {};
    options.mode = 'Download';
    options.region = 'PageSettings';
    options.multiplePage = true;
    options.pageHeight = 300;
  }
}

```

```
options.pageWidth = 300;
diagram.print(options);
}
}
,
```

### Limitations

We have a limitation in exporting the image with HTML and Native node. So, Syncfusion Essential PDF library is used, which supports HTML Content to Image conversion by using the advanced Qt WebKit rendering engine. You can refer to the following KB link for more details.

[<https://www.syncfusion.com/kb/13298/how-to-print-or-export-the-html-and-native-node-into-image-format>]

### Tooltip in Vue Diagram component

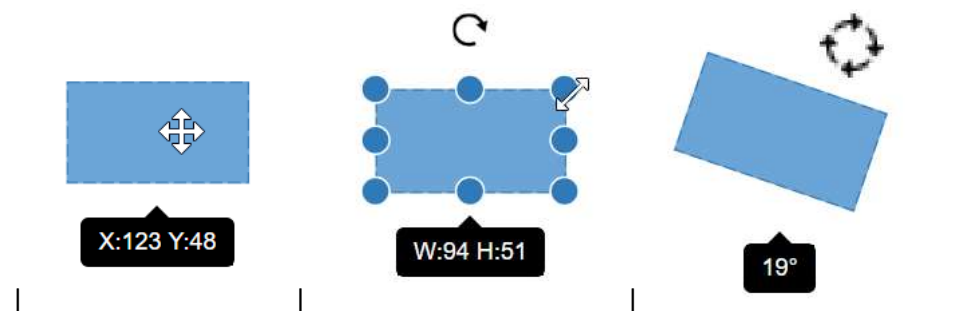
<!-- markdownlint-disable MD010 -->

In Graphical User Interface (GUI), the tooltip is a message that is displayed when mouse hovers over an element. The diagram provides tooltip support while dragging, resizing, rotating a node, and when the mouse hovers any diagram element.

#### Default tooltip

By default, diagram displays a tooltip to provide the size, position, and angle related information while dragging, resizing, and rotating. The following images illustrate how the diagram displays the node information during an interaction.

Drag	Resize	Rotate



#### Common tooltip for all nodes and connectors

The diagram provides support to show tooltip when the mouse hovers over any node/connector.

To show tooltip on mouse over, the [tooltip](#) property of diagram model needs to be set with the tooltip [content](#) and [position](#) as shown in the following example.

#### APP.VUE

```
<template>
  <div id="app">
    <ejs-diagram id="diagram" :width='width' :height='height'
      :nodes='nodes'
      :tooltip='tooltip' :constraints='constraints'></ejs-diagram>
```

```

    </div>
  </template>
  <script>
    import Vue from 'vue';
    import { DiagramPlugin, DiagramConstraints } from '@syncfusion/ej2-vue-
    diagrams';
    Vue.use(DiagramPlugin);
    let nodes = [{
      id: "node1",
      height: 60,
      offsetX: 300,
      offsetY: 80,
      annotations: [{
        content: "start"
      }]
    }]
    export default {
      name: 'app'
      data() {
        return {
          width: "100%",
          height: "350px",
          constraints: DiagramConstraints.Default |
DiagramConstraints.Tooltip,
          nodes: nodes,
          tooltip: {
            content: 'Nodes',
            position: 'TopLeft'
          }
        }
      }
    }
  </script>
  <style>
    @import "../../node_modules/@syncfusion/ej2-vue-
    diagrams/styles/material.css";
  </style>

```

{% previewsample "page.domainurl/code-snippet/diagram/tooltip/tooltip-cs1" %}

#### Disable tooltip at runtime

The tooltip on mouse over can be disabled by assigning the [tooltip](#) property as `null`. The following code example illustrates how to disable the mouse over tooltip at runtime.

```

`javascript
export default {
  name: 'app'
  data () {
    return {
      width: "100%",
      height: "350px",

```

tooltip: null

```
}
}
}
,
```

### Tooltip for a specific node/connector

The tooltip can be customized for each node and connector. Remove the **InheritTooltip** option from the [constraints](#) of that node/connector. The following code example illustrates how to customize the tooltip for individual elements.

#### APP.VUE

```
<template>
  <div id="app">
    <ejs-diagram id="diagram" :width='width' :height='height'
:nodes='nodes'
:tooltip='tooltip' :constraints='constraints'></ejs-diagram>
  </div>
</template>
<script>
  import Vue from 'vue';
  import { DiagramPlugin, DiagramConstraints } from '@syncfusion/ej2-vue-
diagrams';
  Vue.use(DiagramPlugin);
  let nodes = [{
    id: "node1",
    height: 60,
    offsetX: 300,
    offsetY: 80,
    annotations: [{
      content: "start"
    }]
  }]
  export default {
    name: 'app'
    data() {
      return {
        width: "100%",
        height: "350px",
        constraints: DiagramConstraints.Default |
DiagramConstraints.Tooltip,
        nodes: nodes,
        //Defines mouse over tooltip for a node
        tooltip: {
          //Sets the content of the Tooltip
          content: 'Node1',
          //Sets the position of the Tooltip
          position: 'BottomRight',
          //Sets the tooltip position relative to the node
          relativeMode: 'Object'
        },
      }
    }
  }
}
```

```

    }
  </script>
  <style>
    @import ".../node_modules/@syncfusion/ej2-vue-
    diagrams/styles/material.css";
  </style>

```

```
{% previewsample "page.domainurl/code-snippet/diagram/tooltip/tooltip-cs2" %}
```

### Tooltip for Ports

The tooltip feature has been implemented to support Ports, providing the ability to display information or descriptions when the mouse hovers over them.

To display tooltips on mouseover, set the desired tooltip [content](#) by utilizing the [tooltip](#) property.

Tooltips for Ports can be enabled or disabled using the [PortConstraints](#) Tooltip property.

```

`ts
let ports: [{
  offset: {x: 1,y: 0.5},
  tooltip: {content: 'Port Tooltip'},
  //enable Port Tooltip Constraints
  constraints: PortConstraints.Default | PortConstraints.ToolTip,
  //disable Port Tooltip Constraints
  constraints: PortConstraints.Default ~& PortConstraints.ToolTip
}]
`

```

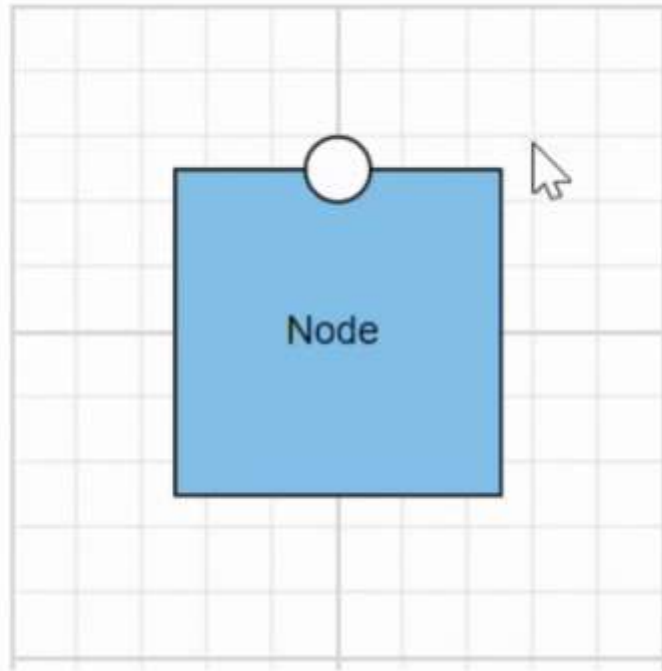
Dynamic modification of tooltip content is supported, allowing you to change the displayed tooltip content during runtime.

```

`ts
{
  //change tooltip content at run time
  diagram.nodes[0].ports[0].tooltip.content = 'New Tooltip Content';
  diagram.databind;
}
`

```

The following image illustrates how the diagram displays tooltips during an interaction with ports:



Here, the code provided below demonstrates the port tooltip Interaction.

#### **APP.VUE**

```
<template>
  <div id="app">
    <ejs-diagram id="diagram" :width='width' :height='height'
:nodes='nodes'
    ></ejs-diagram>
  </div>
</template>
<script>
  import Vue from 'vue';
  import { DiagramPlugin, PortVisibility, PointPortModel, PortConstraints }
from '@syncfusion/ej2-vue-diagrams';
  Vue.use(DiagramPlugin);
  let nodes = [{
    id: "node1",
    // Position of the node
    offsetX: 250,
    offsetY: 250,
    // Size of the node
    width: 100,
    height: 100,
    // Initialize port collection
    ports: [{
      // Sets the position for the port
      offset: {
        x: 0.5,
        y: 0.5
      },
      visibility: PortVisibility.Visible,
      shape: 'Circle',
    }],
  }];
```

```

        tooltip: {content: 'Port Tootip', relativeMode: 'Object',
position: 'TopRight'},
        constraints: PortConstraints.Default | PortConstraints.ToolTip
    ]]
    ]]
    export default {
        name: 'app',
        data() {
            return {
                width: "100%",
                height: "350px",
                nodes: nodes,
            }
        }
    }
</script>
<style>
    @import "../node_modules/@syncfusion/ej2-vue-
diagrams/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/diagram/tooltip/tooltip-cs7" %}

### Tooltip template content

Any text or image can be added to the tooltip, by default. To customize the tooltip layout or to create your own visualized element on the tooltip, template can be used.

The following code example illustrates how to add formatted HTML content to the tooltip.

### APP.VUE

```

<template>
    <div id="app">
        <ejs-diagram id="diagram" :width='width' :height='height'
:nodes='nodes'
        :tooltip='tooltip' :constraints='constraints'></ejs-diagram>
    </div>
</template>
<script>
    import Vue from 'vue';
    import { DiagramPlugin,DiagramConstraints } from '@syncfusion/ej2-vue-
diagrams';
    Vue.use(DiagramPlugin);
    let nodes = [{
        id: "node1",
        height: 60,
        offsetX: 300,
        offsetY: 80,
        annotations: [{
            content: "start"
        }]
    }]
    function getContent(): HTMLElement {
        let tooltipContent: HTMLElement = document.createElement('div');
        tooltipContent.innerHTML = '<div style="background-color: #f4f4f4;
color: black; border-width:1px;border-style: solid;border-color: #d3d3d3;

```

```

border-radius: 8px;white-space: nowrap;"> <span style="margin: 10px;">
Tooltip !!! </span> </div>';
    return tooltipContent;
  }
  export default {
    name: 'app'
    data() {
      return {
        width: "100%",
        height: "350px",
        constraints: DiagramConstraints.Default |
DiagramConstraints.Tooltip,
        nodes: nodes,
        //Defines mouse over tooltip for a node
        tooltip: {
          //Sets the content of the Tooltip
          content: getContent(),
          //Sets the position of the Tooltip
          position: 'TopLeft',
          //Sets the tooltip position relative to the node
          relativeMode: 'Object'
        },
      },
    },
  }
}
</script>
<style>
  @import "../node_modules/@syncfusion/ej2-vue-
diagrams/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/diagram/tooltip/tooltip-cs3" %}

### Tooltip alignments

#### *Tooltip relative to object*

The diagram provides support to show tooltip around the node/connector that is hovered by the mouse. The tooltip can be aligned by using the [position](#) property of the tooltip.

The [relativeMode](#) property of the tooltip defines whether the tooltip has to be displayed around the object or at the mouse position.

The following code example illustrates how to position the tooltip around object.

#### **APP.VUE**

```

<template>
  <div id="app">
    <ejs-diagram id="diagram" :width='width' :height='height'
:nodes='nodes'
:tooltip='tooltip' :constraints='constraints'></ejs-diagram>
  </div>
</template>
<script>
  import Vue from 'vue';
  import { DiagramPlugin,DiagramConstraints } from '@syncfusion/ej2-vue-
diagrams';

```



```

Vue.use(DiagramPlugin);
let nodes = [{
  id: "node1",
  height: 60,
  offsetX: 300,
  offsetY: 80,
  annotations: [{
    content: "start"
  }]
}]
export default {
  name: 'app'
  data() {
    return {
      width: "100%",
      height: "350px",
      constraints: DiagramConstraints.Default |
DiagramConstraints.Tooltip,
      nodes: nodes,
      //Defines mouse over tooltip for a node
      tooltip: {
        content: 'Node1',
        //Sets the alignment properties
        position: 'BottomRight',
        //Sets to show tooltip around the element
        relativeMode: 'Object',
      },
    }
  }
}
</script>
<style>
  @import "../node_modules/@syncfusion/ej2-vue-
diagrams/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/diagram/tooltip/tooltip-cs4" %}

#### Tooltip relative to mouse position

To display the tooltip at mouse position, need to set **mouse** option to the [relativeMode](#) property of the tooltip.

The following code example illustrates how to show tooltip at mouse position.

#### APP.VUE

```

<template>
  <div id="app">
    <ejs-diagram id="diagram" :width='width' :height='height'
:nodes='nodes'
:tooltip='tooltip' :constraints='constraints'></ejs-diagram>
  </div>
</template>
<script>
  import Vue from 'vue';
  import { DiagramPlugin, DiagramConstraints } from '@syncfusion/ej2-vue-
diagrams';

```

```

Vue.use(DiagramPlugin);
let nodes = [{
  id: "node1",
  height: 60,
  offsetX: 300,
  offsetY: 80,
  annotations: [{
    content: "start"
  }]
}]
export default {
  name: 'app'
  data() {
    return {
      width: "100%",
      height: "350px",
      constraints: DiagramConstraints.Default |
DiagramConstraints.Tooltip,
      nodes: nodes,
      //Defines mouse over tooltip for a node
      tooltip: {
        content: 'Node1',
        //Sets to show tooltip at mouse position
        relativeMode: 'Mouse',
      },
    }
  }
}
</script>
<style>
  @import "../node_modules/@syncfusion/ej2-vue-
diagrams/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/diagram/tooltip/tooltip-cs5" %}

### Tooltip animation

To animate the tooltip, a set of specific animation effects are available, and it can be controlled by using the [animation](#) property. The animation property also allows you to set delay, duration, and various other effects of your choice.

### APP.VUE

```

<template>
  <div id="app">
    <ejs-diagram id="diagram" :width='width' :height='height'
:nodes='nodes'
:tooltip='tooltip' :constraints='constraints'></ejs-diagram>
  </div>
</template>
<script>
  import Vue from 'vue';
  import { DiagramPlugin, DiagramConstraints } from '@syncfusion/ej2-vue-
diagrams';
  Vue.use(DiagramPlugin);
  let nodes = [{

```

```

        id: "node1",
        height: 60,
        offsetX: 300,
        offsetY: 80,
        annotations: [{
            content: "start"
        }]
    }]
    export default {
        name: 'app'
        data() {
            return {
                width: "100%",
                height: "350px",
                constraints: DiagramConstraints.Default |
DiagramConstraints.Tooltip,
                nodes: nodes,
                //Defines mouse over tooltip for a node
                tooltip: {
                    content: 'Node1',
                    position: 'BottomCenter',
                    relativeMode: 'Object',
                    animation: {
                        //Animation settings to be applied on the Tooltip,
                        while it is being shown over the target.
                        open: {
                            //Animation effect on the Tooltip is applied
                        during open and close actions.
                            effect: 'ZoomIn',
                            //Duration of the animation that is completed
                        per animation cycle.
                            duration: 1000,
                            //Indicating the waiting time before animation
                        begins.
                            delay: 0
                        },
                        //Animation settings to be applied on the Tooltip,
                        when it is closed.
                        close: {
                            effect: 'ZoomOut',
                            duration: 500,
                            delay: 0
                        }
                    }
                },
            },
        },
    }
}
</script>
<style>
    @import "../node_modules/@syncfusion/ej2-vue-
diagrams/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/diagram/tooltip/tooltip-cs6" %}

## User handle in Vue Diagram component

- User handles are used to add some frequently used commands around the selector. To create user handles, define and add them to the [userHandles](#) collection of the [selectedItems](#) property.
- The name property of user handle is used to define the name of the user handle and its further used to find the user handle at runtime and do any customization.

### Alignment

User handles can be aligned relative to the node boundaries. It has [margin](#), [offset](#), [side](#), [horizontalAlignment](#), and [verticalAlignment](#) settings. It is quite tricky when all four alignments are used together but gives more control over alignment.

#### Offset for user handle

The [offset](#) property of [userHandles](#) is used to align the user handle based on fractions. 0 represents top/left corner, 1 represents bottom/right corner, and 0.5 represents half of width/height.

#### Side

The [side](#) property of [userHandles](#) is used to align the user handle by using the [Top](#), [Bottom](#), [Left](#), and [Right](#) options.

#### Horizontal and vertical alignments

The [horizontalAlignment](#) property of [userHandles](#) is used to set how the user handle is horizontally aligned at the position based on the [offset](#). The [verticalAlignment](#) property is used to set how user handle is vertically aligned at the position.

#### Margin for user handle

Margin is an absolute value used to add some blank space in any one of its four sides. The [userHandles](#) can be displaced with the [margin](#) property.

### Appearance

The appearance of the user handle can be customized by using the [size](#), [borderColor](#), [backgroundColor](#), [visible](#), [pathData](#), and [pathColor](#) properties of the [userHandles](#).

## APP.VUE

```
<template>
  <div id="app">
    <ejs-diagram id="diagram" :width='width' :height='height'
:nodes='nodes' :getCustomTool='getCustomTool'
:selectedItems='selectedItems'></ejs-diagram>
  </div>
</template>
<script>
  import Vue from 'vue';
  import { DiagramPlugin, NodeModel
, MoveTool, BasicShapeModel, UserHandleModel, randomId,
SelectorConstraints, cloneObject } from '@syncfusion/ej2-vue-diagrams';
  Vue.use(DiagramPlugin);
  let diagramInstance: Diagram;
  let shape: BasicShapeModel = {
    type: 'Basic',
    shape: 'Rectangle'
  };
  function getTool(action: string): ToolBase {
```

```

    let tool: ToolBase | any;
    if (action === "clone") {
        tool = new CloneTool(diagramInstance.commandHandler);
    }
    return tool;
}

class CloneTool extends MoveTool {
    public mouseDown(args: MouseEventArgs): void {
        let newObject: any;
        if (diagramInstance.selectedItems.nodes.length > 0) {
            newObject =
cloneObject(diagramInstance.selectedItems.nodes[0]) as NodeModel;
        } else {
            newObject =
cloneObject(diagramInstance.selectedItems.connectors[0]) as ConnectorModel;
        }
        newObject.id += randomId();
        diagramInstance.paste([newObject]);
        args.source = diagramInstance.nodes[diagramInstance.nodes.length
- 1] as IElement;
        args.sourceWrapper = args.source.wrapper;
        super.mouseDown(args);
        this.inAction = true;
    }
}

let nodes = [{
    id: 'node',
    offsetX: 100,
    offsetY: 100,
    shape: shape
}]

let handles: UserHandleModel[] = [{
    name: 'clone',
    pathData: 'M60.3,18H27.5c-3,0-5.5,2.4-5.5,5.5v38.2h5.5V23.5h32.7V18z
M68.5,28.9h-30c-3, 0-5.5,2.4-5.5,5.5v38.2c0,3,2.4,5.5,5.5,5.5h30c3,0,5.5-
2.4,5.5-5.5V34.4C73.9,31.4,71.5,28.9,68.5,28.9z M68.5,72.5h-
30V34.4h30V72.5z',
    visible: true,
    offset: 0,
    side: 'Bottom',
    margin: {
        top: 0,
        bottom: 0,
        left: 0,
        right: 0
    }
}];

export default {
    name: 'app'
    data() {
        return {
            width: "100%",
            height: "350px",
            nodes: nodes,
            selectedItems: {
                constraints: SelectorConstraints.UserHandle,
                userHandles: handles
            }
        }
    }
}

```

```

        },
        getCustomTool: getTool
    }
}
mounted: function() {
    let obj: any = document.getElementById("diagram");
    diagramInstance = obj.ej2_instances[0];
}
}
</script>
<style>
    @import "../node_modules/@syncfusion/ej2-vue-
    diagrams/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/diagram/interaction/UserHandle-cs2" %}

### Fixed user handles

The fixed user handles are used to add some frequently used commands around the node and connector even without selecting it.

### Initialization an fixed user handles

To create the fixed user handles, define and add them to the collection of nodes and connectors property. The following code example used to create an fixed user handles for the nodes and connectors.

### APP.VUE

```

<template>
    <div id="app">
        <ejs-diagram id="diagram" :width='width' :height='height'
        :nodes='nodes' ></ejs-diagram>
    </div>
</template>
<script>
    import Vue from 'vue';
    import { DiagramPlugin } from '@syncfusion/ej2-vue-diagrams';
    Vue.use(DiagramPlugin);
    let nodes = [{
        offsetX: 250,
        offsetY: 250,
        width: 100,
        height: 100,
        style: {
            fill: '#6BA5D7',
            strokeColor: 'white'
        },
    },
    // A fixed user handle is created and stored in fixed user handle
    collection of Node.
    fixedUserHandles: [{margin: { right: 20 },pathData: 'M60.3,18H27.5c-
    3,0-5.5,2.4-5.5,5.5v38.2h5.5V23.5h32.7V18z M68.5,28.9h-30c-3,0-5.5,2.4-
    5.5,5.5v38.2c0,3,2.4,5.5,5.5,5.5h30c3,0,5.5-2.4,5.5-
    5.5V34.4C73.9,31.4,71.5,28.9,68.5,28.9z M68.5,72.5h-30V34.4h30V72.5z' }]
    }]
    export default {
        name: 'app'
    }

```

```
data() {  
  return {  
    width: "100%",  
    height: "350px",  
    nodes: nodes,  
  }  
}  
}  
</script>  
<style>  
  @import "../node_modules/@syncfusion/ej2-vue-  
    diagrams/styles/material.css";  
</style>
```

{% previewsample "page.domainurl/code-snippet/diagram/interaction/FixedUserHandle-cs1" %}

### Customization

- The id property of fixed user handle is used to define the unique identification of the fixed user handle and it is further used to add custom events to the fixed user handle.
- The fixed user handle can be positioned relative to the node and connector boundaries. It has offset, padding and cornerRadius settings. It is used to position and customize the fixed user handle.
- The `Padding` is used to leave the space that is inside the fixed user handle between the icon and the border.
- The corner radius allows to create fixed user handles with rounded corners. The radius of the rounded corner is set with the `cornerRadius` property.

Note: The PathData needs to be provided to render fixed user handle.

### Size

Diagram allows you set size for the fixed user handles by using the `width` and `height` property. The default value of the width and height property is 10.

### Style

- You can change the style of the fixed user handles with the specific properties of `borderColor`, `borderWidth`, and background color using the `handleStrokeColor`, `handleStrokeWidth`, and `fill` properties, and the icon `borderColor`, and `borderWidth` using the `iconStrokeColor` and `iconStrokeWidth`.
- The fixed user handle's `iconStrokeColor` and `iconStrokeWidth` property used to change the stroke color and stroke width of the given `pathData`.
- The fixed user handle `handleStrokeColor` and, `fill` properties are used to define the background color and border color of the userhandle and the `handleStrokeWidth` property is used to define the border width of the fixed user handle.
- The `visible` property of the fixed user handle enables or disables the visibility of fixed user handle.

The following code explains how to customize the appearance of the fixed user handles.

**APP.VUE**

```

<template>
  <div id="app">
    <ejs-diagram id="diagram" :width='width' :height='height'
      :nodes='nodes' :connectors='connectors' ></ejs-diagram>
    </div>
  </template>
<script>
  import Vue from 'vue';
  import { DiagramPlugin } from '@syncfusion/ej2-vue-diagrams';
  Vue.use(DiagramPlugin);
  let nodes = [{
    offsetX: 150,
    offsetY: 150,
    width: 100,
    height: 100,
    style: {
      fill: '#6BA5D7',
      strokeColor: 'white'
    },
  },
  // A fixed user handle is created and stored in fixed user handle
  // collection of Node.
  {
    fixedUserHandles: [{offset: { x: 0, y: 0 },margin: { right: 20 },
    width: 20, height: 20,

padding:{left:3,right:3,top:3,bottom:3},iconStrokeColor:'white',fill:'black'
, id: 'user1', pathData: 'M60.3,18H27.5c-3,0-5.5,2.4-
5.5,5.5v38.2h5.5V23.5h32.7V18z M68.5,28.9h-30c-3,0-5.5,2.4-
5.5,5.5v38.2c0,3,2.4,5.5,5.5,5.5h30c3,0,5.5-2.4,5.5-
5.5V34.4C73.9,31.4,71.5,28.9,68.5,28.9z M68.5,72.5h-30V34.4h30V72.5z' }}
  ]};
  let connectors = [{
    sourcePoint: {
      x: 300,
      y: 100
    },
    targetPoint: {
      x: 400,
      y: 200
    },
    type: 'Orthogonal',
    style: {
      strokeColor: '#6BA5D7'
    },
  },
  // A fixed user handle is created and stored in fixed user handle
  // collection of Connector.
  {
    fixedUserHandles: [{ offset: 0.5, width: 20, alignment: 'Before',
height: 20,

padding:{left:3,right:3,top:3,bottom:3},iconStrokeColor:'white',fill:'black'
, id: 'usercon1', displacement:{x:10,y:10}, pathData: 'M60.3,18H27.5c-3,0-
5.5,2.4-5.5,5.5v38.2h5.5V23.5h32.7V18z M68.5,28.9h-30c-3,0-5.5,2.4-
5.5,5.5v38.2c0,3,2.4,5.5,5.5,5.5h30c3,0,5.5-2.4,5.5-
5.5V34.4C73.9,31.4,71.5,28.9,68.5,28.9z M68.5,72.5h-30V34.4h30V72.5z' }}
  ]}
  export default {

```



```

    name: 'app'
    data() {
      return {
        width: "100%",
        height: "350px",
        nodes: nodes,
        connectors: connectors
      }
    }
  }
</script>
<style>
  @import "../node_modules/@syncfusion/ej2-vue-
  diagrams/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/diagram/interaction/CustomizingFixedUserHandle-cs1" %}

Note: The Fixed user handle id should need to be unique.

### Customizing the node fixed user handle

- The node fixed user handle can be aligned relative to the node boundaries. It has **margin** and **offset** settings. It is quite useful to position the node fixed userhandle and used together and gives you more control over the node fixed user handle positioning.

#### Margin for the node fixed user handle

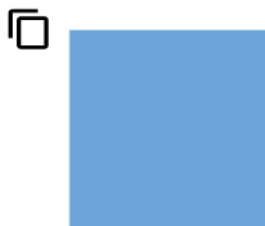
Margin is an absolute value used to add some blank space in any one of its four sides. The fixed user handle can be displaced with the **margin** property.

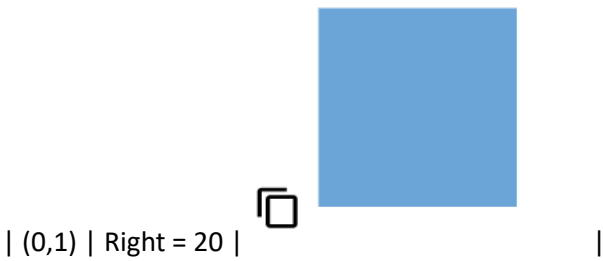
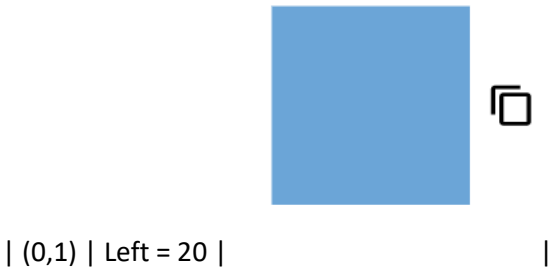
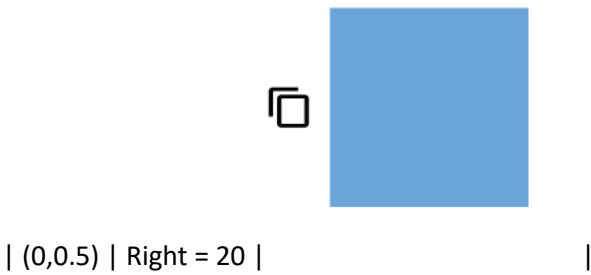
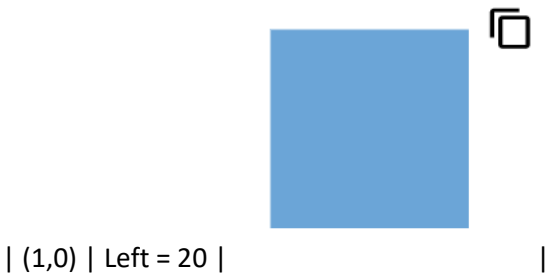
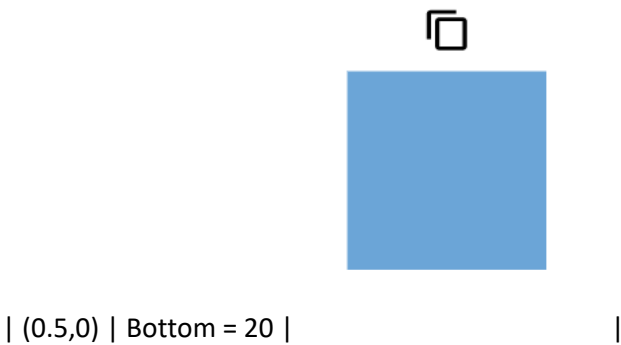
#### Offset for the node fixed user handle

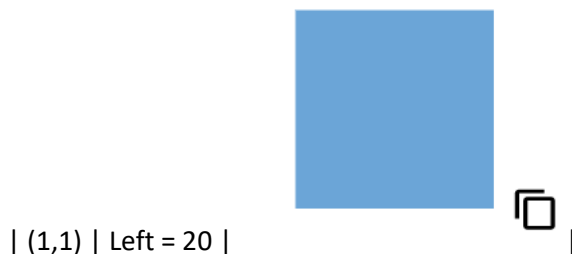
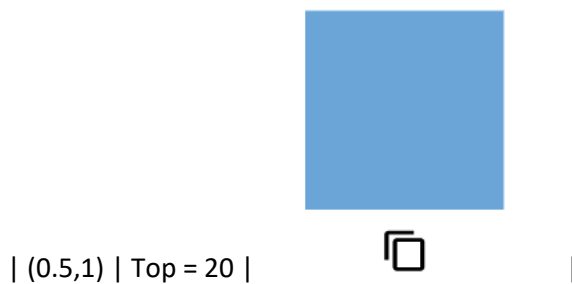
The **offset** property of fixed user handle is used to align the user handle based on the **x** and **y** points. (0,0) represents the top or left corner and (1,1) represents the bottom or right corner.

The following table shows all the possible alignments visually shows the fixed user handle positions.

Offset	Margin	Output
(0,0)	Right = 20	







The following code explains how to customize the node fixed user handle.

#### APP.VUE

```
<template>
  <div id="app">
    <ejs-diagram id="diagram" :width='width' :height='height'
:nodes='nodes' ></ejs-diagram>
  </div>
</template>
<script>
  import Vue from 'vue';
  import { DiagramPlugin } from '@syncfusion/ej2-vue-diagrams';
  Vue.use(DiagramPlugin);
  let nodes = [{
    offsetX: 250,
    offsetY: 250,
    width: 100,
    height: 100,
    style: {
      fill: '#6BA5D7',
      strokeColor: 'white'
    },
  },
  // A fixed user handle is created and stored in fixed user handle
collection of Node.
    fixedUserHandles: [{offset: { x: 0, y: 0 },margin: { right: 20 },
width: 20, height: 20,
      id: 'user1', pathData: 'M60.3,18H27.5c-3,0-5.5,2.4-
5.5,5.5v38.2h5.5V23.5h32.7V18z M68.5,28.9h-30c-3,0-5.5,2.4-
5.5,5.5v38.2c0,3,2.4,5.5,5.5,5.5h30c3,0,5.5-2.4,5.5-
5.5V34.4C73.9,31.4,71.5,28.9,68.5,28.9z M68.5,72.5h-30V34.4h30V72.5z' }]
  ]
  export default {
    name: 'app'
    data() {
```

```
        return {
            width: "100%",
            height: "350px",
            nodes: nodes,
        }
    }
}
</script>
<style>
    @import "../node_modules/@syncfusion/ej2-vue-
    diagrams/styles/material.css";
</style>
```

{% previewsample "page.domainurl/code-snippet/diagram/interaction/NodeFixedUserHandle-cs1" %}

Customizing the connector fixed user handle

- The connector fixed user handle can be aligned relative to the connector boundaries. It has alignment, displacement and offset settings. It is useful to position the connector fixed userhandle and used together and gives you more control over the connector fixed user handle positioning.
- The `offset` and `alignment` properties of fixed user handle allows you to align the connector fixed user handles to the segments.

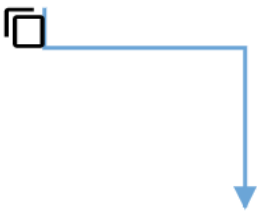
Offset for the connector fixed user handle

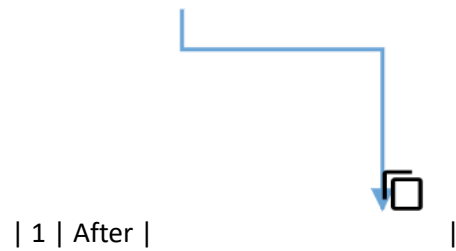
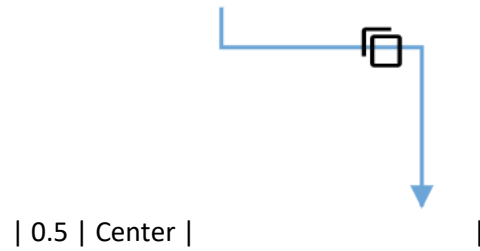
The `offset` property of connector fixed user handle is used to align the user handle based on fractions. 0 represents the connector source point, 1 represents the connector target point, and 0.5 represents the center point of the connector segment.

Alignment

The connector’s fixed user handle can be aligned over its segment path using the `alignment` property of fixed user handle.

The following table shows all the possible alignments visually shows the fixed user handle positions.

Offset	Alignment	Output
-----	-----	-----
		
0   Before		

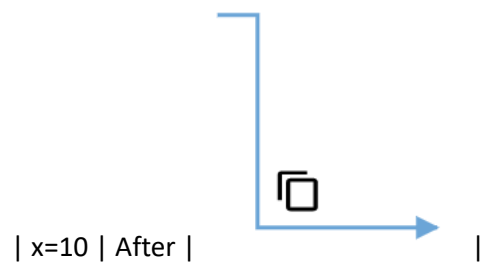
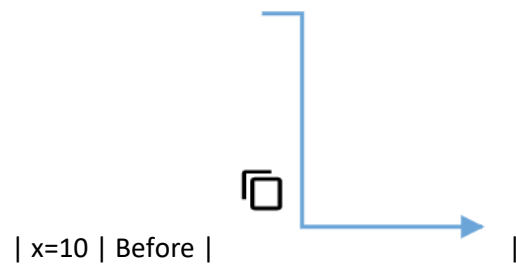


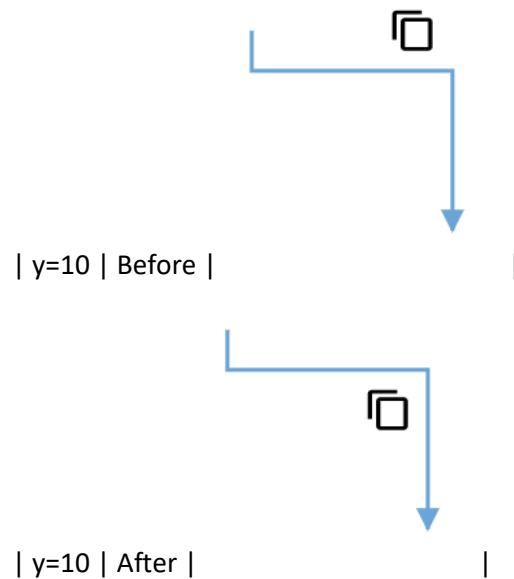
### Displacement

- The **displacement** property allows you to specify the space to be left from the connector segment based on the x and y value provided.

The following table shows all the possible alignments visually shows the fixed user handle positions.

Displacement	Alignment	Output
-----	-----	-----





Note: Displacement will not be done if the alignment is set to be center.

The following code explains how to customize the connector fixed user handle.

#### **APP.VUE**

```
<template>
  <div id="app">
    <ejs-diagram id="diagram" :width='width' :height='height'
:connectors='connectors' ></ejs-diagram>
  </div>
</template>
<script>
  import Vue from 'vue';
  import { DiagramPlugin } from '@syncfusion/ej2-vue-diagrams';
  Vue.use(DiagramPlugin);
  let connectors = [{
    id: "connector1",
    style: {
      strokeColor: '#6BA5D7',
      fill: '#6BA5D7',
      strokeWidth: 2
    },
    targetDecorator: {
      style: {
        fill: '#6BA5D7',
        strokeColor: '#6BA5D7'
      }
    },
    sourcePoint: {
      x: 100,
      y: 100
    },
    targetPoint: {
      x: 200,
      y: 200
    }
  }];
```

```

    }, type: 'Orthogonal', // A fixed user handle is created and stored in
    fixedUserHandles: [{ offset: 0.5, width: 20, alignment: 'Before',
    height: 20, id: 'usercon1', displacement: {x:10,y:10}, pathData:
    'M60.3,18H27.5c-3,0-5.5,2.4-5.5,5.5v38.2h5.5V23.5h32.7V18z M68.5,28.9h-30c-
    3,0-5.5,2.4-5.5,5.5v38.2c0,3,2.4,5.5,5.5,5.5h30c3,0,5.5-2.4,5.5-
    5.5V34.4C73.9,31.4,71.5,28.9,68.5,28.9z M68.5,72.5h-30V34.4h30V72.5z' }]
    ]]
    export default {
      name: 'app'
      data() {
        return {
          width: "100%",
          height: "350px",
          connectors: connectors
        }
      }
    }
  </script>
  <style>
    @import "../node_modules/@syncfusion/ej2-vue-
    diagrams/styles/material.css";
  </style>

```

```

{% previewsample "page.domainurl/code-snippet/diagram/interaction/ConnectorFixedUserHandle-cs1"
%}

```

#### Tooltip support for User Handle

The diagram provides support to show tooltip when the mouse hovers over any user handle.

To show tooltip on mouse over, the [tooltip](#) property of diagram model needs to be set with the tooltip [content](#) and [position](#) as shown in the following example.

#### APP.VUE

```

<template>
  <div id="app">
    <ejs-diagram id="diagram" :width='width' :height='height'
    :nodes='nodes' :constraints='constraints'
    :selectedItems='selectedItems'></ejs-diagram>
  </div>
</template>
<script>
  import Vue from 'vue';
  import { DiagramPlugin, NodeModel
  , MoveTool, BasicShapeModel, UserHandleModel, randomId, NodeConstraints,
  DiagramConstraints, SelectorConstraints, cloneObject } from '@syncfusion/ej2-
  vue-diagrams';
  Vue.use(DiagramPlugin);
  let diagramInstance;
  let nodes = [{
    id: 'node',
    // Position of the node
    offsetX: 250,
    offsetY: 250,
    // Size of the node

```

```

        width: 100,
        height: 100,
        style: { fill: '#6BA5D7', strokeColor: 'white' },
        tooltip: { content: 'node1', position: 'BottomRight', relativeMode:
'Object' },
        constraints: NodeConstraints.Default | NodeConstraints.Tooltip,
    }}
    let handles = [{
        name: 'clone', pathData: 'M60.3,18H27.5c-3,0-5.5,2.4-
5.5,5.5v38.2h5.5V23.5h32.7V18z M68.5,28.9h-30c-3,' +
        '0-5.5,2.4-5.5,5.5v38.2c0,3,2.4,5.5,5.5,5.5h30c3,0,5.5-2.4,5.5-
5.5V34.4C73.9,31.4,71.5,28.9,68.5,28.9z ' +
        'M68.5,72.5h-30V34.4h30V72.5z',
        visible: true, offset: 0, side: 'Bottom', margin: { top: 0, bottom:
0, left: 0, right: 0 },
        tooltip: { content: 'handle1', position: 'BottomRight',
relativeMode: 'Object' }
    }];
    export default {
        name: 'app',
        data() {
            return {
                width: "100%",
                height: "350px",
                nodes: nodes,
                constraints: DiagramConstraints.Default |
DiagramConstraints.Tooltip,
                selectedItems: {
                    constraints: SelectorConstraints.All,userHandles:
handles
                }
            }
        },
    }
</script>
<style>
    @import "../node_modules/@syncfusion/ej2-vue-
diagrams/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/diagram/interaction/UserHandle-cs3" %}

## Style in Vue Diagram component

### Customizing the connector end point handle

Use the following CSS to customize the connector end point handle.

```

`scss
.e-diagram-endpoint-handle {
    fill: red;
    stroke: green;
}
`

```



### Customizing the connector end point handle when connected

Use the following CSS to customize the connector end point handle when connected.

```
`scss
.e-diagram-endpoint-handle.e-connected {
  fill: red;
  stroke: green;
}
`
```

### Customizing the connector end point handle when disabled

Use the following CSS to customize the connector end point handle when disabled.

```
`scss
.e-diagram-endpoint-handle.e-disabled {
  fill: red;
  opacity: 1;
  stroke: green;
}
`
```

### Customizing the bezier connector handle

Use the following CSS to customize the bezier handle properties.

```
`scss
.e-diagram-bezier-handle {
  fill: red;
  stroke: green;
}
`
```

### Customizing the bezier connector line

Use the following CSS to customize the bezier line properties.

```
`scss
.e-diagram-bezier-line {
  stroke: black;
}
`
```

### Customizing the resize handle

Use the following CSS to customize the resize handle.

```
`scss
.e-diagram-resize-handle {
  fill: white;
  opacity: 1;
  stroke: white;
}
`
```

#### Customizing the selector pivot line

Use the following CSS to customize the line between the selector and rotate handle.

```
`scss
.e-diagram-pivot-line {
  stroke: red;
}
`
```

#### Customizing the selector border

Use the following CSS to customize the selector border.

```
`scss
.e-diagram-border {
  stroke: red;
}
`
```

#### Customizing the rotate handle

Use the following CSS to customize the rotate handle properties.

```
`scss
.e-diagram-rotate-handle {
  fill: red;
  stroke: green;
}
`
```

#### Customizing the symbolpalette while hovering

Use the following CSS to customize the symbolpalette while hovering.

```
`scss
.e-symbolpalette .e-symbol-hover:hover {
  background: red;
}
```

```
}  
,
```

#### [Customizing the symbolpalette when selected](#)

Use the following CSS to customize the symbolpalette when selected.

```
`scss  
  
.e-symbolpalette .e-symbol-selected {  
  background: white;  
}  
,
```

#### [Customizing the ruler](#)

Use the following CSS to customize the ruler properties.

```
`scss  
  
.e-diagram .e-ruler {  
  background-color: red;  
  font-size: 13px;  
}  
,
```

#### [Customizing the ruler overlap](#)

Use the following CSS to ruler overlap properties.

```
`scss  
  
.e-diagram .e-ruler-overlap {  
  background-color: red;  
}  
,
```

#### [Customizing the text edit](#)

Use the following CSS to customize the text edit properties.

```
`scss  
  
.e-diagram .e-diagram-text-edit {  
  background: white;  
  border-color: red;  
  border-style: dashed;  
  border-width: 1px;  
  box-sizing: content-box;  
  color: black;
```

```
min-width: 50px;
}
```

### Customizing the text edit on selection

Use the following CSS to customize the text edit on selection properties.

```
`scss
.e-diagram-text-edit::selection {
background: red;
color: green;
}
```

### Ruler in Vue Diagram component

The Ruler provides a horizontal and vertical guide for measuring in the Diagram control. The Ruler can be used to measure the diagram objects, indicate positions, and align diagram elements. This is especially useful in creating scale models.

#### Adding Rulers to the Diagram

- The [rulerSettings](#) property is used to control the visibility and appearance of the ruler in the diagram.
- The RulerSettings [showRulers](#) property is used to show or hide the rulers in the diagram.
- The RulerSettings [horizontalRuler](#) and [verticalRuler](#) properties are used to customize the rulers appearance in the diagram.

The following code shows how to add a ruler to the diagram.

#### APP.VUE

```
<template>
  <div id="app">
    <ejs-diagram id="diagram" :width='width' :height='height'
:rulerSettings='rulerSettings'></ejs-diagram>
  </div>
</template>
<script>
import Vue from 'vue';
import { DiagramPlugin } from '@syncfusion/ej2-vue-diagrams';
Vue.use(DiagramPlugin);
export default {
  name: 'app'
  data() {
    return {
      width: "100%",
      height: "600px",
      rulerSettings: {
        showRulers: true
      }
    }
  }
}
```

```

    }
  }
</script>
<style>
  @import "../node_modules/@syncfusion/ej2-vue-
  diagrams/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/diagram/ruler/ruler-cs1" %}

### Customizing the Ruler

By default, the ruler segments are arranged based on pixel values.

- The HorizontalRuler's [interval](#) property allows you to define the interval between ruler segments and the [segmentWidth](#) property allows you to define the segment width of the ruler. Similarly, you can use the VerticalRuler's [interval](#) and [segmentWidth](#) properties are used to define the interval and segment width of the vertical ruler
- The HorizontalRuler's [tickAlignment](#) property is used to align the ruler tick either left or right side of the ruler. The VerticalRuler's [tickAlignment](#) property is used to align the ruler tick either top or bottom side of the ruler.
- The HorizontalRuler's [arrangeTick](#) and VerticalRuler's [arrangeTick](#) function is provided for the purpose of customizing the appearance of ruler ticks. It will be called for each tick rendering.
- The HorizontalRuler's [markerColor](#) and VerticalRuler's [markerColor](#) properties are used to define the ruler marker color and marker will be shown when performing the interaction in the diagram.

The following code shows how the diagram ruler can be customized.

### APP.VUE

```

<template>
  <div id="app">
    <ejs-diagram id="diagram" :width='width' :height='height'
    :rulerSettings='rulerSettings' ></ejs-diagram>
  </div>
</template>
<script>
  import Vue from 'vue';
  import { DiagramPlugin } from '@syncfusion/ej2-vue-diagrams';
  Vue.use(DiagramPlugin);
  export default {
    name: 'app'
    data() {
      return {
        width: "100%",
        height: "600px",
        rulerSettings: {
          showRulers: true, horizontalRuler:{interval:8,
segmentWidth:100, thickness:25,
tickAlignment:"LeftOrTop"},verticalRuler:{interval:10, segmentWidth:150,
thickness:35, tickAlignment:"RightOrBottom"}
        }
      }
    }
  }

```

```
    }  
  }  
</script>  
<style>  
  @import "../node_modules/@syncfusion/ej2-vue-  
diagrams/styles/material.css";  
</style>
```

{% previewsample "page.domainurl/code-snippet/diagram/ruler/ruler-cs2" %}

Note : The MarkerColor property can be customized using the [marker](#) CSS style.

### Layers in Vue Diagram component

**Layer** is used to organize related shapes on a diagram control. A layer is a named category of shapes. By assigning shapes to different layers, you can selectively view, remove, and lock different categories of shapes.

In diagram, [Layers](#) provide a way to change the properties of all shapes that have been assigned to that layer. The following properties can be set.

- Visible
- Lock
- Objects
- AddInfo

#### Visible

The layer's [visible](#) property is used to control the visibility of the elements assigned to the layer.

```
`javascript  
let nodes = [{  
  id: 'node1',  
  width: 100,  
  height: 100,  
  offsetX: 100,  
  offsetY: 100,  
  annotations: [{  
    content: 'Default Shape'  
  }]  
},  
{  
  id: 'node2',  
  width: 100,  
  height: 100,
```

```
offsetX: 300,
offsetY: 100,
shape: {
  type: 'Path',
  data:
'M540.3643,137.9336L546.7973,159.7016L570.3633,159.7296L550.7723,171.9366L558.9053,194.9966L
540.3643,' +
'179.4996L521.8223,194.9966L529.9553,171.9366L510.3633,159.7296L533.9313,159.7016L540.3643,1
37.9336z'
},
annotations: [{
  content: 'Path Element'
}]
}
];

let connectors: ConnectorModel[] = [{
  id: 'connector1',
  type: 'Straight',
  sourcePoint: {
    x: 100,
    y: 300
  },
  targetPoint: {
    x: 200,
    y: 400
  },
}];

export default {
  name: 'app'
  data() {
    return {
      width: "100%",
      height: "350px",
      nodes: nodes,
```

```

layers: [{
  id: 'layer1',
  visible: true,
  objects: ['node1', 'node2', 'connector1']
}]
}
}}
`

```

### Lock

The layer's [lock](#) property is used to prevent or allow changes to the elements dimension and position.

```
`javascript
```

```

let nodes = [{
  id: 'node1',
  width: 100,
  height: 100,
  offsetX: 100,
  offsetY: 100,
  annotations: [{
    content: 'Default Shape'
  }]
},
{
  id: 'node2',
  width: 100,
  height: 100,
  offsetX: 300,
  offsetY: 100,
  shape: {
    type: 'Path',
    data:
'M540.3643,137.9336L546.7973,159.7016L570.3633,159.7296L550.7723,171.9366L558.9053,194.9966L
540.3643,' +
'179.4996L521.8223,194.9966L529.9553,171.9366L510.3633,159.7296L533.9313,159.7016L540.3643,1
37.9336z'

```



```
},
annotations: [{
  content: 'Path Element'
}]
}
];

let connectors: ConnectorModel[] = [{
  id: 'connector1',
  type: 'Straight',
  sourcePoint: {
    x: 100,
    y: 300
  },
  targetPoint: {
    x: 200,
    y: 400
  },
}];

export default {
  name: 'app'
  data() {
    return {
      width: "100%",
      height: "350px",
      nodes: nodes,
      layers: [{
        id: 'layer1',
        visible: true,
        objects: ['node1'],
        lock: true
      },
      {
        id: 'layer2',
```

```

visible: true,
objects: ['node2', 'connector1'],
lock: false
}}
}
}}
,

```

### Objects

The layer's [objects](#) property defines the diagram elements to the layer.

```

`javascript
let nodes = [{
id: 'node1',
width: 100,
height: 100,
offsetX: 100,
offsetY: 100,
annotations: [{
content: 'Default Shape'
}]
},
{
id: 'node2',
width: 100,
height: 100,
offsetX: 300,
offsetY: 100,
shape: {
type: 'Path',
data:
'M540.3643,137.9336L546.7973,159.7016L570.3633,159.7296L550.7723,171.9366L558.9053,194.9966L
540.3643,' +
'179.4996L521.8223,194.9966L529.9553,171.9366L510.3633,159.7296L533.9313,159.7016L540.3643,1
37.9336z'
},

```

```
annotations: [{
  content: 'Path Element'
}]
}
];
let connectors: ConnectorModel[] = [{
  id: 'connector1',
  type: 'Straight',
  sourcePoint: {
    x: 100,
    y: 300
  },
  targetPoint: {
    x: 200,
    y: 400
  },
}];
export default {
  name: 'app'
  data() {
    return {
      width: "100%",
      height: "350px",
      nodes: nodes,
      layers: [{
        id: 'layer1',
        visible: true,
        objects: ['node1', 'node2', 'connector1'],
        lock: true
      }]
    }
  }
}
```

### AddInfo

The [addInfo](#) property of layers allow you to maintain additional information to the layers.

The following code illustrates how to add additional information to the layers.

```
`javascript
let nodes = [{
  id: 'node1',
  width: 100,
  height: 100,
  offsetX: 100,
  offsetY: 100,
  annotations: [{
    content: 'Default Shape'
  }]
},
{
  id: 'node2',
  width: 100,
  height: 100,
  offsetX: 300,
  offsetY: 100,
  shape: {
    type: 'Path',
    data:
'M540.3643,137.9336L546.7973,159.7016L570.3633,159.7296L550.7723,171.9366L558.9053,194.9966L
540.3643,' +
'179.4996L521.8223,194.9966L529.9553,171.9366L510.3633,159.7296L533.9313,159.7016L540.3643,1
37.9336z'
  },
  annotations: [{
    content: 'Path Element'
  }]
}
];

let connectors: ConnectorModel[] = [{
```

```

id: 'connector1',
type: 'Straight',
sourcePoint: {
  x: 100,
  y: 300
},
targetPoint: {
  x: 200,
  y: 400
},
});
let addInfo: Object = { Description: 'Layer1' };
export default {
  name: 'app'
  data() {
    return {
      width: "100%",
      height: "350px",
      nodes: nodes,
      layers: [{
        id: 'layer1',
        visible: true,
        objects: ['node1', 'node2', 'connector1'],
        addInfo: addInfo
      }]
    }
  }
}

```

#### *Add layer at runtime*

Layers can be added at runtime by using the [addLayer](#) public method.

The layer's [ID](#) property defines the ID of the layer, and its further used to find the layer at runtime and do any customization.

The following code illustrates how to add a layer.

```
`javascript
```

```
export default {
  name: 'app'
  data() {
    return {
      width: "100%",
      height: "350px",
    }
  }
  mounted: function() {
    let diagramInstance: Diagram;
    let diagramObj: any = document.getElementById("diagram");
    diagramInstance = diagramObj.ej2_instances[0];
    // add the layers to the existing diagram layer collection
    diagram.addLayer({
      id: 'newlayer',
      objects: [],
      visible: true,
      lock: false,
      zIndex: -1
    }, [{
      type: 'Straight',
      sourcePoint: {
        x: 100,
        y: 300
      },
      targetPoint: {
        x: 200,
        y: 400
      }
    }]);
  }
}
```

### *Remove layer at runtime*

Layers can be removed at runtime by using the [removeLayer](#) public method.

The following code illustrates how to remove a layer.

```
`javascript
export default {
  name: 'app'
  data() {
    return {
      width: "100%",
      height: "350px",
    }
  }
  mounted: function() {
    let diagramInstance: Diagram;
    let diagramObj: any = document.getElementById("diagram");
    diagramInstance = diagramObj.ej2_instances[0];
    // remove the diagram layers
    diagram.removeLayer([diagram.model.layers[i]]);
  }
}
```

### *moveObjects*

Objects of the layers can be moved by using the [moveObjects](#) public method.

The following code illustrates how to move objects from one layer to another layer from the diagram.

```
`javascript
export default {
  name: 'app'
  data() {
    return {
      width: "100%",
      height: "350px",
    }
  }
  mounted: function() {
```

```

let diagramInstance: Diagram;
let diagramObj: any = document.getElementById("diagram");
diagramInstance = diagramObj.ej2_instances[0];
// move the objects of diagram layers
diagram.moveObjects(['connector1', 'layer2'];
}
}
`

```

#### *bringLayerForward*

Layers can be moved forward at runtime by using the [bringLayerForward](#) public method.

The following code illustrates how to bring forward to layer.

```

`javascript
export default {
  name: 'app'
  data() {
    return {
      width: "100%",
      height: "350px",
    }
  }
  mounted: function() {
    let diagramInstance: Diagram;
    let diagramObj: any = document.getElementById("diagram");
    diagramInstance = diagramObj.ej2_instances[0];
    // move the layer forward
    diagram.bringLayerForward('layer1');
  }
}
`

```

#### *sendLayerBackward*

Layers can be moved backward at runtime by using the [sendLayerBackward](#) public method.

The following code illustrates how to send backward to layer.

```

`javascript
export default {

```



```
name: 'app'
data() {
  return {
    width: "100%",
    height: "350px",
  }
}
mounted: function() {
  let diagramInstance: Diagram;
  let diagramObj: any = document.getElementById("diagram");
  diagramInstance = diagramObj.ej2_instances[0];
  // move the layer backward
  diagram.sendLayerBackward('layer1');
}
},
```

#### *cloneLayer*

Layers can be cloned with its object by using the [cloneLayer](#) public method.

The following code illustrates how to bring forward to layer.

```
`javascript
export default {
  name: 'app'
  data() {
    return {
      width: "100%",
      height: "350px",
    }
  }
  mounted: function() {
    let diagramInstance: Diagram;
    let diagramObj: any = document.getElementById("diagram");
    diagramInstance = diagramObj.ej2_instances[0];
    // clone a layer with its object
```

```
diagram.cloneLayer('layer2');  
}  
}  
`
```

#### [getActiveLayer](#)

To get the active layers back in diagram, use the [getActiveLayer](#) public method.

The following code illustrates how to bring forward to layer.

```
`javascript  
export default {  
  name: 'app'  
  data() {  
    return {  
      width: "100%",  
      height: "350px",  
    }  
  }  
  mounted: function() {  
    let diagramInstance: Diagram;  
    let diagramObj: any = document.getElementById("diagram");  
    diagramInstance = diagramObj.ej2_instances[0];  
    // gets the active layer back  
    diagram.getActiveLayer();  
  }  
}  
`
```

#### [setActiveLayer](#)

Set the active layer by using the [setActiveLayer](#) public method.

The following code illustrates how to bring forward to layer.

```
`javascript  
export default {  
  name: 'app'  
  data() {  
    return {  
      width: "100%",  
    }  
  }  
}
```

```

height: "350px",
}
}
mounted: function() {
let diagramInstance: Diagram;
let diagramObj: any = document.getElementById("diagram");
diagramInstance = diagramObj.ej2_instances[0];
// set the active layer
//@param layerName defines the name of the layer which is to be active layer
diagram.setActiveLayer('layer2');
}
}
`

```

### Context menu in Vue Diagram component

<!-- markdownlint-disable MD010 -->

In graphical user interface (GUI), a context menu is a type of menu that appears when you perform right-click operation. Nested level of context menu items can be created.

Diagram provides some in-built context menu items and allows to define custom menu items through the [contextMenuSettings](#) property.

#### Customize context menu

The [show](#) property helps you to enable/disable the context menu. Diagram provides some default context menu items to ease the execution of some frequently used commands.

The following code illustrates how to enable the default context menu items.

#### **APP.VUE**

```

<template>
  <div id="app">
    <ejs-diagram id="diagram" :width='width' :height='height'
      :nodes='nodes' :connectors='connectors'
      :contextMenuSettings='contextMenuSettings'></ejs-diagram>
    </div>
  </template>
<script>
  import Vue from 'vue';
  import { DiagramPlugin, DiagramContextMenu, Diagram } from
  '@syncfusion/ej2-vue-diagrams';
  Diagram.Inject(DiagramContextMenu);
  Vue.use(DiagramPlugin);
  let nodes = [{
    id: 'node1',
    width: 100,
    height: 100,
    offsetX: 100,

```

```

      offsetY: 100,
      style: {
        fill: '#6BA5D7',
        strokeColor: 'white',
        strokeWidth: 1
      },
      annotations: [{
        id: 'label1',
        content: 'Rectangle1',
        offset: {
          x: 0.5,
          y: 0.5
        },
        style: {
          color: 'white'
        }
      }]
    },
    {
      id: 'node2',
      width: 100,
      height: 100,
      offsetX: 300,
      offsetY: 100,
      style: {
        fill: '#6BA5D7',
        strokeColor: 'white',
        strokeWidth: 1
      },
      annotations: [{
        id: 'label2',
        content: 'Rectangle2',
        offset: {
          x: 0.5,
          y: 0.5
        },
        style: {
          color: 'white'
        }
      }]
    }
  ];
  let connectors = [{
    id: 'connector1',
    sourceID: 'node1',
    targetID: 'node2',
    type: 'Straight',
    style: {
      strokeColor: '#6BA5D7',
      fill: '#6BA5D7',
      strokeWidth: 2,
      targetDecorator: {
        style: {
          fill: '#6BA5D7',
          strokeColor: '#6BA5D7'
        }
      }
    }
  }]
}

```

```

    ]]
    export default {
      name: 'app'
      data() {
        return {
          width: "100%",
          height: "350px",
          nodes: nodes,
          connectors: connectors,
          contextMenuSettings: {
            show: true,
          }
        }
      }
    }
  </script>
  <style>
    @import "../node_modules/@syncfusion/ej2-vue-
    diagrams/styles/material.css";
  </style>

```

{% previewsample "page.domainurl/code-snippet/diagram/contextmenu/events-cs1" %}

Context menu can be defined for individual node with the desired context menu items.

- Apart from the default context menu items, define some additional context menu items. Those additional items have to be defined and added to the [items](#) property of the context menu.
- Set text and ID for context menu item using the context menu [text](#) and [ID](#) properties respectively.
- Set an image for the context menu item using the context menu [url](#) property.
- The [iconCss](#) property defines the class/multiple classes separated by a space for the menu item that is used to include an icon. Menu item can include font icon and sprite image.
- The [target](#) property used to set the target to show the menu item.
- The [separator](#) property defines the horizontal lines that are used to separate the menu items. You cannot select the separators. You can enable separators to group the menu items using the [separator](#) property.

The following code example illustrates how to add custom context menu items.

#### **APP.VUE**

```

<template>
  <div id="app">
    <ejs-diagram id="diagram" :width='width' :height='height'
    :nodes='nodes' :connectors='connectors'
    :contextMenuSettings='contextMenuSettings'></ejs-diagram>
  </div>
</template>
<script>
  import Vue from 'vue';
  import {
    DiagramPlugin, DiagramContextMenu, Diagram, DiagramBeforeMenuOpenEventArgs }
  from '@syncfusion/ej2-vue-diagrams';
  Diagram.Inject(DiagramContextMenu);

```

```
Vue.use(DiagramPlugin);
let nodes = [{
  id: 'node1',
  width: 100,
  height: 100,
  offsetX: 100,
  offsetY: 100,
  style: {
    fill: '#6BA5D7',
    strokeColor: 'white',
    strokeWidth: 1
  },
  annotations: [{
    id: 'label1',
    content: 'Rectangle1',
    offset: {
      x: 0.5,
      y: 0.5
    },
    style: {
      color: 'white'
    }
  }]
},
{
  id: 'node2',
  width: 100,
  height: 100,
  offsetX: 300,
  offsetY: 100,
  style: {
    fill: '#6BA5D7',
    strokeColor: 'white',
    strokeWidth: 1
  },
  annotations: [{
    id: 'label2',
    content: 'Rectangle2',
    offset: {
      x: 0.5,
      y: 0.5
    },
    style: {
      color: 'white'
    }
  }]
}
];
let connectors = [{
  id: 'connector1',
  sourceID: 'node1',
  targetID: 'node2',
  type: 'Straight',
  style: {
    strokeColor: '#6BA5D7',
    fill: '#6BA5D7',
    strokeWidth: 2,
    targetDecorator: {
```

```

        style: {
            fill : '#6BA5D7',
            strokeColor : '#6BA5D7'
        }
    }
}
} ]
export default {
    name: 'app'
    data() {
        return {
            width: "100%",
            height: "350px",
            nodes: nodes,
            connectors: connectors,
            contextMenuSettings: {
                //Enables the context menu
                show: true,
                // Defines the custom context menu items
                items: [{
                    // Text to be displayed
                    text: 'Save',
                    //Sets the id for the item
                    id: 'save',
                    //ContextMenu can be visible based on the target in
                    which you open the ContextMenu.
                    target: '.e-elementcontent',
                    // Sets the css icons for the item
                    iconCss: 'e-save'
                },
                {
                    text: 'Load',
                    id: 'load',
                    target: '.e-elementcontent',
                    iconCss: 'e-load'
                },
                {
                    text: 'Clear',
                    id: 'clear',
                    target: '.e-elementcontent',
                    iconCss: 'e-clear'
                }
            ],
            // Hides the default context menu items
            showCustomMenuOnly: false,
        }
    }
}
</script>
<style>
    @import
    "https://ej2.syncfusion.com/vue/documentation/../../node_modules/@syncfusion/ej2-vue-diagrams/styles/material.css";
</style>

```

```
{% previewsample "page.domainurl/code-snippet/diagram/contextmenu/default-cs1" %}
```

To display the custom context menu items alone, set the [showCustomMenuOnly](#) property to true.

### Template Support for Context menu

- Diagram provides template support for context menu. The context menu items can be customized by using the `contextMenuBeforeItemRender` event. The `contextMenuBeforeItemRender` event triggers while rendering each menu item.
- In the following sample, the menu item is rendered with key code for specified action in Context Menu using the template. Here, the key code is specified for the cut and copy at right corner of the menu items by adding a span element in the `contextMenuBeforeItemRender` event.

### APP.VUE

```
<template>
  <div id="app">
    <ejs-diagram id="diagram" :width='width' :height='height'
:nodes='nodes' :connectors='connectors'
:contextMenuSettings='contextMenuSettings'></ejs-diagram>
  </div>
</template>
<script>
  import Vue from 'vue';
  import { DiagramPlugin, DiagramContextMenu, Diagram ,
MenuEventArgs, } from '@syncfusion/ej2-vue-diagrams';
  Diagram.Inject(DiagramContextMenu);
  Vue.use(DiagramPlugin);
  let nodes = [{
    id: 'node1',
    width: 100,
    height: 100,
    offsetX: 100,
    offsetY: 100,
    style: {
      fill: '#6BA5D7',
      strokeColor: 'white',
      strokeWidth: 1
    },
    annotations: [{
      id: 'labell1',
      content: 'Rectangle1',
      offset: {
        x: 0.5,
        y: 0.5
      },
      style: {
        color: 'white'
      }
    }
  ]
},
{
  id: 'node2',
  width: 100,
  height: 100,
```



```

      offsetX: 300,
      offsetY: 100,
      style: {
        fill: '#6BA5D7',
        strokeColor: 'white',
        strokeWidth: 1
      },
      annotations: [{
        id: 'label2',
        content: 'Rectangle2',
        offset: {
          x: 0.5,
          y: 0.5
        },
        style: {
          color: 'white'
        }
      }]
    }
  ];

  let connectors = [{
    id: 'connector1',
    sourceID: 'node1',
    targetID: 'node2',
    type: 'Straight',
    style: {
      strokeColor: '#6BA5D7',
      fill: '#6BA5D7',
      strokeWidth: 2,
      targetDecorator: {
        style: {
          fill: '#6BA5D7',
          strokeColor: '#6BA5D7'
        }
      }
    }
  }]

  export default {
    name: 'app'
    data() {
      return {
        width: "100%",
        height: "350px",
        nodes: nodes,
        connectors: connectors,
        contextMenuSettings: {
          //Enables the context menu
          show: true,
          items: [{
            text: 'Cut', id: 'Cut', target: '.e-diagramcontent',
            iconCss: 'e-Cut'
          },
          {
            text: 'Copy', id: 'Copy', target: '.e-
diagramcontent',
            iconCss: 'e-Copy'
          }
        ]
      }
    }
  }

```

```

// Hides the default context menu items
showCustomMenuOnly: true,
contextMenuBeforeItemRender: function (args:
MenuEventArgs) {
    // To render template in li.
    let shortCutSpan: HTMLElement =
createElement('span');
    let text: string = args.item.text;
    let shortCutText: string = text === 'Cut' ? 'Ctrl +
S' : (text === 'Copy' ?
        'Ctrl + U' : 'Ctrl + Shift + I');
    shortCutSpan.textContent = shortCutText;
    args.element.appendChild(shortCutSpan);
    shortCutSpan.setAttribute('class', 'shortcut');
    },
},
}
}
</script>
<style>
    @import "../node_modules/@syncfusion/ej2-vue-
diagrams/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/diagram/contextmenu/menutemplate-cs1" %}

### Context menu events

You would be notified with events, when you try to open the context menu items [contextMenuOpen](#) and when you click the menu items [contextMenuClick](#)

The following code example illustrates how to define those events.

### APP.VUE

```

<template>
    <div id="app">
        <ejs-diagram id="diagram" ref="diagram" :width='width'
:height='height' :nodes='nodes' :connectors='connectors'
:contextMenuSettings='contextMenuSettings':contextMenuOpen="contextMenuOpen"
:contextMenuClick="contextMenuClick" ></ejs-diagram>
    </div>
</template>
<script>
    import Vue from 'vue';
    import { DiagramPlugin, DiagramContextMenu, Diagram
, DiagramBeforeMenuOpenEventArgs,
MenuEventArgs, } from '@syncfusion/ej2-vue-diagrams';
    Diagram.Inject(DiagramContextMenu);
    Vue.use(DiagramPlugin);
    let nodes = [{
        id: 'node1',
        width: 100,
        height: 100,
        offsetX: 100,
        offsetY: 100,
    }

```

```

    style: {
      fill: '#6BA5D7',
      strokeColor: 'white',
      strokeWidth: 1
    },
    annotations: [{
      id: 'label1',
      content: 'Rectangle1',
      offset: {
        x: 0.5,
        y: 0.5
      },
      style: {
        color: 'white'
      }
    }]
  },
  {
    id: 'node2',
    width: 100,
    height: 100,
    offsetX: 300,
    offsetY: 100,
    style: {
      fill: '#6BA5D7',
      strokeColor: 'white',
      strokeWidth: 1
    },
    annotations: [{
      id: 'label2',
      content: 'Rectangle2',
      offset: {
        x: 0.5,
        y: 0.5
      },
      style: {
        color: 'white'
      }
    }]
  }
];
let connectors = [{
  id: 'connector1',
  sourceID: 'node1',
  targetID: 'node2',
  type: 'Straight',
  style: {
    strokeColor: '#6BA5D7',
    fill: '#6BA5D7',
    strokeWidth: 2,
    targetDecorator: {
      style: {
        fill: '#6BA5D7',
        strokeColor: '#6BA5D7'
      }
    }
  }
}]

```

```

    } ]
    export default {
      name: 'app'
      data() {
        return {
          width: "100%",
          height: "350px",
          nodes: nodes,
          connectors: connectors,
          contextMenuSettings: {
            //Enables the context menu
            show: true,
            items: [{
              text: 'delete',
              id: 'delete',
            }],
            // Hides the default context menu items
            showCustomMenuOnly: false,
          },
          contextMenuOpen: (args) => {
            var diagram = this.$refs.diagram.ej2Instances;
            //do your custom action here.
            for (let item of args.items) {
              if (item.text === 'delete') {
                if (!diagram.selectedItems.nodes.length &&
!diagram.selectedItems.connectors.length) {
                  args.hiddenItems.push(item.id);
                }
              }
            }
          },
          contextMenuClick: (args) => {
            var diagram = this.$refs.diagram.ej2Instances;
            //do your custom action here.
            if (args.item.id === 'delete') {
              if ((diagram.selectedItems.nodes.length +
diagram.selectedItems.connectors.length) > 0) {
                diagram.cut();
              }
            }
          }
        }
      }
    }
  }
</script>
<style>
  @import "../node_modules/@syncfusion/ej2-vue-
diagrams/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/diagram/contextmenu/custom-cs1" %}

### Symbol palette in Vue Diagram component

The **SymbolPalette** displays a collection of palettes. The Palette shows a set of nodes and connectors. It allows to drag and drop the nodes and connectors into the Diagram.

### Create symbol palette

- The [width](#) and [height](#) property of the symbolpalette allows to define the size of the symbolpalette.

```
`javascript
export default {
  name: 'app'
  data() {
    return {
      //Defines how many palettes can be at expanded mode at a time
      expandMode: "Multiple",
      //Defines the palette collection
      palettes: [{
        //Sets the id of the palette
        id: 'flow',
        //Sets whether the palette expands/collapse its children
        expanded: true,
        //Adds the palette items to palette
        symbols: flowshapes,
        //Sets the header text of the palette
        title: 'Flow Shapes',
        iconCss: 'e-ddb-icons e-flow'
      },
      {
        id: 'basic',
        expanded: true,
        symbols: basicShapes,
        title: 'Basic Shapes',
        iconCss: 'e-ddb-icons e-basic'
      },
      {
        id: 'connectors',
        expanded: true,
        symbols: connectorSymbols,
```

```

title: 'Connectors',
iconCss: 'e-ddb-icons e-connector'
}
],
palettewidth: "100%",
paletteheight: "700px",
symbolHeight: 60,
symbolWidth: 60,
}
}
}
,

```

<!-- markdownlint-disable MD010 -->

### Add palettes to SymbolPalette

A palette allows to display a group of related symbols and it textually annotates the group with its header.

Palettes [palettes](#) can be added as a collection of symbol groups.

The collection of predefined symbols can be added in palettes using [symbols](#) property.

To initialize a palette, define a JSON object with the property [id](#) that is unique id is set to the palettes.

The following code example illustrates how to define a palette and how its added to symbol palette.

### APP.VUE

```

<template>
  <div id="app">
    <ejs-symbolpalette id="symbolpalette" :expandMode='expandMode'
    :palettes='palettes' :width='palettewidth' :height='paletteheight'
    :symbolHeight='symbolHeight'
                                :symbolWidth='symbolWidth'></ejs-
symbolpalette>
  </div>
</template>
<script>
  import Vue from 'vue';
  import { DiagramPlugin,
    Diagram,
    NodeModel,
    UndoRedo,
    ConnectorModel,
    Node,
    Connector,
    SymbolPalette, SymbolPalettePlugin,
    SymbolInfo,
    ShapeModel } from '@syncfusion/ej2-vue-diagrams';
  import { ExpandMode } from "@syncfusion/ej2-vue-navigations";

```

```

Vue.use(DiagramPlugin);
Vue.use(SymbolPalettePlugin);
let flowshapes: NodeModel[] = [{
  id: 'process',
  shape: {
    type: 'Flow',
    shape: 'Process'
  }
},
{
  id: 'document',
  shape: {
    type: 'Flow',
    shape: 'Document'
  }
},
{
  id: 'predefinedprocess',
  shape: {
    type: 'Flow',
    shape: 'PreDefinedProcess'
  }
}
];
let svgshapes: NodeModel[] = [{
  id: 'node2', style: { fill: 'none' },
  annotations: [{ content: 'Start \n Text Editing' }],
  shape: {
    type: 'Native', content: '<g
xmlns="http://www.w3.org/2000/svg"> <g transform="translate(1 1)">
  <g> <path style="fill:#61443C;"
d="M61.979,435.057c2.645-0.512,5.291-0.853,7.936-1.109c-2.01,1.33-
4.472,1.791-6.827,1.28 C62.726,435.13,62.354,435.072,61.979,435.057z"/>
    <path style="fill:#61443C;" d="M502.469,502.471h-
25.6c0.163-30.757-20.173-57.861-49.749-66.304 c-5.784-1.581-11.753-
2.385-17.749-2.389c-2.425-0.028-4.849,0.114-7.253,0.427c1.831-7.63,2.747-
15.45,2.731-23.296 c0.377-47.729-34.52-88.418-81.749-95.317c4.274-
0.545,8.577-0.83,12.885-0.853c25.285,0.211,49.448,10.466,67.167,28.504
c17.719,18.039,27.539,42.382,27.297,67.666c0.017,7.846-0.9,15.666-
2.731,23.296c2.405-0.312,4.829-0.455,7.253-0.427
C472.572,434.123,502.783,464.869,502.469,502.471z"/> </g>
    <path style="fill:#8B685A;" d="M476.869,502.471H7.536c-0.191-
32.558,22.574-60.747,54.443-67.413
c0.375,0.015,0.747,0.072,1.109,0.171c2.355,0.511,4.817,0.05,6.827-
1.28c1.707-0.085,3.413-0.171,5.12-0.171
c4.59,0,9.166,0.486,13.653,1.451c2.324,0.559,4.775,0.147,6.787-1.141c2.013-
1.288,3.414-3.341,3.879-5.685 c7.68-39.706,39.605-70.228,79.616-
76.117c4.325-0.616,8.687-0.929,13.056-0.939c13.281-
0.016,26.409,2.837,38.485,8.363
c3.917,1.823,7.708,3.904,11.349,6.229c2.039,1.304,4.527,1.705,6.872,1.106c2.
345-0.598,4.337-2.142,5.502-4.264 c14.373-25.502,39.733-42.923,68.693-
47.189h0.171c47.229,6.899,82.127,47.588,81.749,95.317c0.017,7.846-
0.9,15.666-2.731,23.296 c2.405-0.312,4.829-0.455,7.253-
0.427c5.996,0.005,11.965,0.808,17.749,2.389C456.696,444.61,477.033,471.713,4
76.869,502.471 L476.869,502.471z"/> <path style="fill:#66993E;"
d="M502.469,7.537c0,0-6.997,264.96-192.512,252.245c-20.217-1.549-40.166-
5.59-59.392-12.032 c-1.365-0.341-2.731-0.853-4.096-1.28c0,0-0.597-2.219-

```

```

1.451-6.144c-6.656-34.048-25.088-198.997,231.765-230.144
C485.061,9.159,493.595,8.22,502.469,7.537z"/>          <path
style="fill:#9ACA5C;" d="M476.784,10.183c-1.28,26.197-16.213,238.165-
166.827,249.6      c-20.217-1.549-40.166-5.59-59.392-12.032c-1.365-0.341-
2.731-0.853-4.096-1.28c0,0-0.597-2.219-1.451-6.144
C238.363,206.279,219.931,41.329,476.784,10.183z"/>          <path
style="fill:#66993E;" d="M206.192,246.727c-0.768,3.925-1.365,6.144-
1.365,6.144c-1.365,0.427-2.731,0.939-4.096,1.28      c-21.505,7.427-
44.293,10.417-66.987,8.789C21.104,252.103,8.816,94.236,7.621,71.452c-0.085-
1.792-0.085-2.731-0.085-2.731
C222.747,86.129,211.653,216.689,206.192,246.727z"/>          <path
style="fill:#9ACA5C;" d="M180.336,246.727c-0.768,3.925-1.365,6.144-
1.365,6.144c-1.365,0.427-2.731,0.939-4.096,1.28      c-13.351,4.412-
27.142,7.359-41.131,8.789C21.104,252.103,8.816,94.236,7.621,71.452
C195.952,96.881,185.541,217.969,180.336,246.727z"/>    </g>    <g>
    <path d="M162.136,426.671c3.451-0.001,6.562-2.08,7.882-5.268s0.591-
6.858-1.849-9.2981-8.533-8.533      c-3.341-3.281-8.701-3.256-12.012,0.054c-
3.311,3.311-3.335,8.671-0.054,12.01218.533,8.533
C157.701,425.773,159.872,426.673,162.136,426.671L162.136,426.671z"/>
    <path d="M292.636,398.57c3.341,3.281,8.701,3.256,12.012-0.054c3.311-
3.311,3.335-8.671,0.054-12.0121-8.533-8.533      c-3.341-3.281-8.701-3.256-
12.012,0.054s-3.335,8.671-0.054,12.012L292.636,398.57z"/>          <path
d="M296.169,454.771c-3.341-3.281-8.701-3.256-12.012,0.054c-3.311,3.311-
3.335,8.671-0.054,12.01218.533,8.533      c3.341,3.281,8.701,3.256,12.012-
0.054c3.311-3.311,3.335-8.671,0.054-12.012L296.169,454.771z"/>
    <path d="M386.503,475.37c3.341,3.281,8.701,3.256,12.012-0.054c3.311-
3.311,3.335-8.671,0.054-12.0121-8.533-8.533      c-3.341-3.281-8.701-3.256-
12.012,0.054c-3.311,3.311-3.335,8.671-0.054,12.012L386.503,475.37z"/>
    <path d="M204.803,409.604c2.264,0.003,4.435-0.897,6.033-
2.518.533-8.533c3.281-3.341,3.256-8.701-0.054-12.012      c-3.311-3.311-8.671-
3.335-12.012-0.0541-8.533,8.533c-2.44,2.44-3.169,6.11-1.849,9.298
C198.241,407.524,201.352,409.603,204.803,409.604z"/>          <path
d="M332.803,443.737c2.264,0.003,4.435-0.897,6.033-2.518.533-8.533c3.281-
3.341,3.256-8.701-0.054-12.012      c-3.311-3.311-8.671-3.335-12.012-0.0541-
8.533,8.533c-2.44,2.44-3.169,6.11-1.849,9.298
C326.241,441.658,329.352,443.737,332.803,443.737z"/>          <path
d="M341.336,366.937c2.264,0.003,4.435-0.897,6.033-2.518.533-8.533c3.281-
3.341,3.256-8.701-0.054-12.012      c-3.311-3.311-8.671-3.335-12.012-0.0541-
8.533,8.533c-2.44,2.44-3.169,6.11-1.849,9.298
C334.774,364.858,337.885,366.937,341.336,366.937z"/>          <path
d="M164.636,454.7711-8.533,8.533c-2.188,2.149-3.055,5.307-
2.27,8.271c0.785,2.965,3.1,5.28,6.065,6.065      c2.965,0.785,6.122-
0.082,8.271-2.2718.533-8.533c3.281-3.341,3.256-8.701-0.054-12.012
C173.337,451.515,167.977,451.49,164.636,454.771L164.636,454.771z"/>
    <path d="M232.903,429.1711-8.533,8.533c-2.188,2.149-3.055,5.307-
2.27,8.271c0.785,2.965,3.1,5.28,6.065,6.065      c2.965,0.785,6.122-
0.082,8.271-2.2718.533-8.533c3.281-3.341,3.256-8.701-0.054-12.012
C241.604,425.915,236.243,425.89,232.903,429.171L232.903,429.171z"/>
    <path d="M384.003,409.604c2.264,0.003,4.435-0.897,6.033-2.518.533-
8.533c3.281-3.341,3.256-8.701-0.054-12.012      c-3.311-3.311-8.671-3.335-
12.012-0.0541-8.533,8.533c-2.44,2.44-3.169,6.11-1.849,9.298
C377.441,407.524,380.552,409.603,384.003,409.604z"/>          <path
d="M70.77,463.3041-8.533,8.533c-2.188,2.149-3.055,5.307-
2.27,8.271s3.1,5.28,6.065,6.065      c2.965,0.785,6.122-0.082,8.271-
2.2718.533-8.533c3.281-3.341,3.256-8.701-0.054-12.012
C79.47,460.048,74.11,460.024,70.77,463.304L70.77,463.304z"/>          <path
d="M121.97,446.2381-8.533,8.533c-2.188,2.149-3.055,5.307-

```



```
2.27,8.271c0.785,2.965,3.1,5.28,6.065,6.065      c2.965,0.785,6.122-
0.082,8.271-2.2718.533-8.533c3.281-3.341,3.256-8.701-0.054-12.012
C130.67,442.981,125.31,442.957,121.97,446.238L121.97,446.238z"/>
    <path d="M202.302,420.638c-1.6-1.601-3.77-2.5-6.033-2.5c-2.263,0-
4.433,0.899-6.033,2.51-8.533,8.533      c-2.178,2.151-3.037,5.304-
2.251,8.262c0.786,2.958,3.097,5.269,6.055,6.055c2.958,0.786,6.111-
0.073,8.262-2.25118.533-8.533      c1.601-1.6,2.5-3.77,2.5-
6.033C204.802,424.408,203.903,422.237,202.302,420.638L202.302,420.638z"/>
    <path d="M210.836,463.304c-3.341-3.281-8.701-3.256-
12.012,0.054c-3.311,3.311-3.335,8.671-0.054,12.01218.533,8.533
c2.149,2.188,5.307,3.055,8.271,2.27c2.965-0.785,5.28-3.1,6.065-6.065c0.785-
2.965-0.082-6.122-2.27-8.271L210.836,463.304z"/>    <path
d="M343.836,454.7711-8.533,8.533c-2.188,2.149-3.055,5.307-
2.27,8.271c0.785,2.965,3.1,5.28,6.065,6.065      c2.965,0.785,6.122-
0.082,8.271-2.2718.533-8.533c3.281-3.341,3.256-8.701-0.054-12.012
C352.537,451.515,347.177,451.49,343.836,454.771L343.836,454.771z"/>
    <path d="M429.17,483.904c3.341,3.281,8.701,3.256,12.012-0.054s3.335-
8.671,0.054-12.0121-8.533-8.533      c-3.341-3.281-8.701-3.256-12.012,0.054c-
3.311,3.311-3.335,8.671-0.054,12.012L429.17,483.904z"/>    <path
d="M341.336,401.071c2.264,0.003,4.435-0.897,6.033-2.518.533-8.533c3.281-
3.341,3.256-8.701-0.054-12.012      s-8.671-3.335-12.012-0.0541-8.533,8.533c-
2.44,2.441-3.169,6.11-
1.849,9.298C334.774,398.991,337.885,401.07,341.336,401.071z"/>
    <path d="M273.069,435.204c2.264,0.003,4.435-0.897,6.033-2.518.533-
8.533c3.281-3.341,3.256-8.701-0.054-12.012      s-8.671-3.335-12.012-0.0541-
8.533,8.533c-2.44,2.44-3.169,6.11-
1.849,9.298C266.508,433.124,269.618,435.203,273.069,435.204z"/>
    <path
d="M253.318,258.138c22.738,7.382,46.448,11.338,70.351,11.737c31.602,0.543,62
.581-8.828,88.583-26.796      c94.225-65.725,99.567-227.462,99.75-
234.317c0.059-2.421-0.91-4.754-2.667-6.421c-1.751-1.679-4.141-2.52-6.558-
2.308      C387.311,9.396,307.586,44.542,265.819,104.5c-28.443,42.151-
38.198,94.184-26.956,143.776c-3.411,8.366-6.04,17.03-7.852,25.881      c-
4.581-7.691-9.996-14.854-16.147-21.358c8.023-38.158,0.241-77.939-21.57-
110.261C160.753,95.829,98.828,68.458,9.228,61.196      c-2.417-0.214-
4.808,0.628-6.558,2.308c-1.757,1.667-2.726,4-
2.667,6.421c0.142,5.321,4.292,130.929,77.717,182.142
c20.358,14.081,44.617,21.428,69.367,21.008c18.624-0.309,37.097-3.388,54.814-
9.138c11.69,12.508,20.523,27.407,25.889,43.665
c0.149,15.133,2.158,30.19,5.982,44.832c-12.842-5.666-26.723-8.595-40.759-
8.6c-49.449,0.497-91.788,35.567-101.483,84.058      c-5.094-1.093-10.29-1.641-
15.5-1.638c-42.295,0.38-76.303,34.921-76.025,77.217c-
0.001,2.263,0.898,4.434,2.499,6.035
c1.6,1.6,3.771,2.499,6.035,2.499h494.933c2.263,0.001,4.434-0.898,6.035-
2.499c1.6-1.6,2.499-3.771,2.499-6.035      c0.249-41.103-31.914-75.112-72.967-
77.154c0.65-4.78,0.975-9.598,0.975-14.421c0.914-45.674-28.469-86.455-72.083-
100.045      c-43.615-13.59-90.962,3.282-
116.154,41.391C242.252,322.17,242.793,288.884,253.318,258.138L253.318,258.13
8z M87.519,238.092      c-55.35-38.567-67.358-129.25-69.833-
158.996c78.8,7.921,133.092,32.454,161.458,72.992
c15.333,22.503,22.859,49.414,21.423,76.606c-23.253-35.362-77.83-105.726-
162.473-140.577c-2.82-1.165-6.048-0.736-8.466,1.125      s-3.658,4.873-
3.252,7.897c0.406,3.024,2.395,5.602,5.218,6.761c89.261,36.751,144.772,117.77
6,161.392,144.874      C150.795,260.908,115.29,257.451,87.519,238.092z
M279.969,114.046c37.6-53.788,109.708-86.113,214.408-96.138      c-2.65,35.375-
17.158,159.05-91.892,211.175c-37.438,26.116-85.311,30.57-142.305,13.433
c19.284-32.09,92.484-142.574,212.405-191.954c2.819-1.161,4.805-3.738,5.209-
```

```

6.76c0.404-3.022-0.835-6.031-3.25-7.892    c-2.415-1.861-5.64-2.292-8.459-
1.131C351.388,82.01,279.465,179.805,252.231,222.711
C248.573,184.367,258.381,145.945,279.969,114.046L279.969,114.046z
M262.694,368.017c15.097-26.883,43.468-43.587,74.3-43.746
c47.906,0.521,86.353,39.717,85.95,87.625c-0.001,7.188-0.857,14.351-
2.55,21.337c-0.67,2.763,0.08,5.677,1.999,7.774
c1.919,2.097,4.757,3.1,7.568,2.676c1.994-0.272,4.005-0.393,6.017-
0.362c29.59,0.283,54.467,22.284,58.367,51.617H17.661    c3.899-
29.333,28.777-51.334,58.367-51.617c4-
0.004,7.989,0.416,11.9,1.254c4.622,0.985,9.447,0.098,13.417-2.467    c3.858-
2.519,6.531-6.493,7.408-11.017c7.793-40.473,43.043-69.838,84.258-
70.192c16.045-0.002,31.757,4.582,45.283,13.212
c4.01,2.561,8.897,3.358,13.512,2.205C256.422,375.165,260.36,372.163,262.694,
368.017L262.694,368.017z"/>    </g></g>',
    }
  },
  {
    id: 'syncfusion', style: { fill: 'none'},
    shape: {
      type: 'Native', content: '<g
xmlns="http://www.w3.org/2000/svg">' +
        '<rect height="256" width="256" fill="#34353F"/>' +
        '<path id="path1" transform="rotate(0,128,128)
translate(59,61.2230899333954) scale(4.3125,4.3125)  " fill="#FFFFFF"
d="M18.88501,23.042998L26.804993,23.042998 26.804993,30.969001
18.88501,30.969001z M9.4360352,23.042998L17.358032,23.042998
17.358032,30.969001 9.4360352,30.969001z
M0.014038086,23.042998L7.9360352,23.042998 7.9360352,30.969001
0.014038086,30.969001z M18.871033,13.609001L26.791016,13.609001
26.791016,21.535994 18.871033,21.535994z
M9.4219971,13.609001L17.342041,13.609001 17.342041,21.535994
9.4219971,21.535994z M0,13.609001L7.9219971,13.609001 7.9219971,21.535994
0,21.535994z M9.4219971,4.1859968L17.342041,4.1859968 17.342041,12.113998
9.4219971,12.113998z M0,4.1859968L7.9219971,4.1859968 7.9219971,12.113998
0,12.113998z M25.846008,0L32,5.2310026 26.773987,11.382995
20.619019,6.155998z"/>' +
        '</g>'
    }
  },
  {
    id: 'network', style: { fill: 'none'},
    shape: {
      type: 'Native', content: '<g
xmlns="http://www.w3.org/2000/svg">' +
        '<rect height="256" width="256" fill="#34353F"/>' +
        '<path id="path1" transform="rotate(0,128,128)
translate(59.1078108549118,59) scale(4.3125,4.3125)  " fill="#FFFFFF"
d="M12.12701,24.294998C12.75201,24.294998 13.258998,24.803009
13.258998,25.428009 13.258998,26.056 12.75201,26.563004 12.12701,26.563004
11.499019,26.563004 10.993007,26.056 10.993007,25.428009 10.993007,24.803009
11.499019,24.294998 12.12701,24.294998z
M7.9750035,24.294998C8.6010101,24.294998 9.1090057,24.803009
9.1090057,25.428009 9.1090057,26.056 8.6010101,26.563004 7.9750035,26.563004
7.3480199,26.563004 6.8399942,26.056 6.8399942,25.428009 6.8399942,24.803009
7.3480199,24.294998 7.9750035,24.294998z
M7.9750035,20.286011C8.6010101,20.286011 9.1090057,20.792999
9.1090057,21.419006 9.1090057,22.044006 8.6010101,22.552002

```

7.9750035,22.552002 7.3500035,22.552002 6.8420084,22.044006  
6.8420084,21.419006 6.8420084,20.792999 7.3500035,20.286011  
7.9750035,20.286011z  
M18.499994,19.317001C18.313013,19.317001,18.156,19.472,18.156,19.656006L18.1  
56,27.01001C18.156,27.195007,18.313013,27.350006,18.499994,27.350006L29.5219  
93,27.350006C29.707998,27.350006,29.865988,27.195007,29.865988,27.01001L29.8  
65988,19.656006C29.865988,19.472,29.707998,19.317001,29.521993,19.317001z  
M17.243006,17.443008L30.778003,17.443008C31.425007,17.445007,31.947986,17.96  
2006,31.950001,18.602997L31.950001,28.542007C31.947986,29.182999,31.425007,2  
9.702011,30.778003,29.703003L25.654012,29.703003C25.511007,29.703003  
25.399008,29.824997 25.413992,29.964996 25.430013,30.13501  
25.452993,30.360001 25.477011,30.559998 25.506002,30.809998  
25.727987,30.980011  
25.976003,31.033997L27.756002,31.419006C27.907003,31.452011 28.015005,31.584  
28.015005,31.738007 28.015005,31.883011 27.895986,32  
27.74999,32L27.571005,32 20.450004,32 20.318016,32C20.171013,32  
20.053001,31.883011 20.053001,31.738007 20.053001,31.585007  
20.161003,31.452011  
20.312004,31.419998L22.115989,31.033005C22.35601,30.98201  
22.572014,30.815002 22.596,30.574005 22.616997,30.363007 22.636009,30.130997  
22.648002,29.960007 22.658012,29.819 22.542015,29.70401  
22.399986,29.70401L17.243006,29.703003C16.596002,29.702011,16.072992,29.1829  
99,16.071008,28.542007L16.071008,18.602997C16.072992,17.962006,16.596002,17.  
445007,17.243006,17.443008z M7.9750035,16.133011C8.6020172,16.133011  
9.1100128,16.641006 9.1100128,17.268005 9.1100128,17.893997  
8.6020172,18.402008 7.9750035,18.402008 7.3489964,18.402008  
6.8410013,17.893997 6.8410013,17.268005 6.8410013,16.641006  
7.3489964,16.133011 7.9750035,16.133011z  
M24.027,13.762009C24.654014,13.762009 25.16201,14.270004 25.16201,14.895996  
25.16201,15.522003 24.654014,16.029999 24.027,16.029999 23.400993,16.029999  
22.892998,15.522003 22.892998,14.895996 22.892998,14.270004  
23.400993,13.762009 24.027,13.762009z M24.027,9.6110077C24.653007,9.6110077  
25.161003,10.119003 25.161003,10.74501 25.161003,11.37001  
24.653007,11.878006 24.027,11.878006 23.402,11.878006 22.894005,11.37001  
22.894005,10.74501 22.894005,10.119003 23.402,9.6110077 24.027,9.6110077z  
M24.027,5.6000061C24.654014,5.6000061 25.16201,6.1080017 25.16201,6.7350006  
25.16201,7.3610077 24.654014,7.8690033 24.027,7.8690033 23.400993,7.8690033  
22.892998,7.3610077 22.892998,6.7350006 22.892998,6.1080017  
23.400993,5.6000061 24.027,5.6000061z  
M19.876001,5.6000061C20.503013,5.6000061 21.011009,6.1080017  
21.011009,6.7350006 21.011009,7.3610077 20.503013,7.8690033  
19.876001,7.8690033 19.249994,7.8690033 18.743006,7.3610077  
18.743006,6.7350006 18.743006,6.1080017 19.249994,5.6000061  
19.876001,5.6000061z  
M2.4290157,1.8740082C2.2420037,1.8740082,2.0850215,2.029007,2.0850215,2.2140  
045L2.0850215,9.5680084C2.0850215,9.753006,2.2420037,9.9069977,2.4290157,9.9  
069977L13.451014,9.9069977C13.637995,9.9069977,13.795008,9.753006,13.795008,  
9.5680084L13.795008,2.2140045C13.795008,2.029007,13.637995,1.8740082,13.4510  
14,1.8740082z  
M1.1730042,0L14.706996,0C15.353999,0.0019989014,15.877009,0.51899719,15.8789  
93,1.1600037L15.878993,11.100006C15.877009,11.740005,15.353999,12.26001,14.7  
06996,12.26001L9.5830047,12.26001C9.4399994,12.26001 9.3290069,12.382004  
9.3420074,12.52301 9.3600128,12.692001 9.3829925,12.917999  
9.4060028,13.117004 9.4349945,13.367004 9.6570099,13.53801  
9.9049957,13.591003L11.684994,13.975998C11.835994,14.009003  
11.945003,14.141998 11.945003,14.294998 11.945003,14.440002  
11.826015,14.557007 11.679012,14.557007L11.499996,14.557007

```

4.3789966,14.557007 4.2470081,14.557007C4.1000049,14.557007
3.9819935,14.440002 3.9819937,14.294998 3.9819935,14.141998
4.0899952,14.009003
4.2409961,13.977005L6.0450113,13.589996C6.2860086,13.539001
6.501005,13.373001 6.5249918,13.130997 6.5460184,12.921005
6.5650003,12.688004 6.5769937,12.516998 6.5870035,12.376999
6.4710062,12.262009
6.3290079,12.262009L1.1730042,12.26001C0.52499391,12.26001,0.0020143806,11.7
40005,0,11.100006L0,1.1600037C0.0020143806,0.51899719,0.52499391,0.001998901
4,1.1730042,0z"/>' +
      '</g>'
    }
  }];
  let htmlshapes: NodeModel[] = [{
    id: 'HTML', borderColor: 'black', borderWidth: 1,
    shape: {
      type: 'HTML',
      content: '<div style="height:100%;width:100%;"><button
type="button" style="width:100%;overflow:hidden">HTML</button></div>'
    }
  },
  {
    id: 'Checkbox', borderColor: 'black', borderWidth: 1,
    shape: {
      type: 'HTML',
      content: '<div><div style="height:50%;width:100%;"><input
type="checkbox" value="Yes" style="width:25%">Yes</input></div>' +
        '<div style="height:50%;width:100%;"><input type="checkbox"
value="No" style="width:25%">No</input></div></div>'
    }
  },
  {
    id: 'Dropdown', borderColor: 'black', borderWidth: 1,
    shape: {
      type: 'HTML',
      content: '<div style="height:100%;width:100%;"><select
style="width:100%"><option>SVG</option><option>Canvas</option></select></div>'
    }
  }
  ]];
  let basicShapes: NodeModel[] = [{
    id: 'Rectangle',
    shape: {
      type: 'Basic',
      shape: 'Rectangle'
    }
  },
  {
    id: 'Ellipse',
    shape: {
      type: 'Basic',
      shape: 'Ellipse'
    }
  },
  {
    id: 'Hexagon',
    shape: {

```

```

        type: 'Basic',
        shape: 'Hexagon'
    }
}
];
//Initializes connector symbols for the symbol palette
let connectorSymbols: ConnectorModel[] = [{
    id: 'Link1',
    type: 'Orthogonal',
    sourcePoint: {
        x: 0,
        y: 0
    },
    targetPoint: {
        x: 40,
        y: 40
    },
    targetDecorator: {
        shape: 'Arrow'
    }
},
{
    id: 'Link21',
    type: 'Straight',
    sourcePoint: {
        x: 0,
        y: 0
    },
    targetPoint: {
        x: 40,
        y: 40
    },
    targetDecorator: {
        shape: 'Arrow'
    }
},
{
    id: 'link33',
    type: 'Bezier',
    sourcePoint: {
        x: 0,
        y: 0
    },
    targetPoint: {
        x: 40,
        y: 40
    },
    style: {
        strokeWidth: 2
    },
    targetDecorator: {
        shape: 'None'
    }
}
];
let palette: any;
let size: any;

```

```

let expand: any;
export default {
  name: 'app'
  data() {
    return {
      //Defines how many palettes can be at expanded mode at a
time
      expandMode: "Multiple",
      //Defines the palette collection
      palettes: [{
        //Sets the id of the palette
        id: 'flow',
        //Sets whether the palette expands/collapse its
children
        expanded: true,
        //Adds the palette items to palette
        symbols: flowshapes,
        //Sets the header text of the palette
        title: 'Flow Shapes',
        iconCss: 'e-ddb-icons e-flow'
      },
      {
        //Sets the id of the palette
        id: 'svg',
        //Sets whether the palette expands/collapse its
children
        expanded: true,
        //Adds the palette items to palette
        symbols: svgshapes,
        //Sets the header text of the palette
        title: 'SVG Shapes',
        iconCss: 'e-ddb-icons e-flow'
      },
      {
        //Sets the id of the palette
        id: 'html',
        //Sets whether the palette expands/collapse its
children
        expanded: true,
        //Adds the palette items to palette
        symbols: htmlshapes,
        //Sets the header text of the palette
        title: 'HTML Shapes',
        iconCss: 'e-ddb-icons e-flow'
      },
      {
        id: 'basic',
        expanded: true,
        symbols: basicShapes,
        title: 'Basic Shapes',
        iconCss: 'e-ddb-icons e-basic'
      },
      {
        id: 'connectors',
        expanded: true,
        symbols: connectorSymbols,
        title: 'Connectors',

```

```

        iconCss: 'e-ddb-icons e-connector'
      }
    ],
    palettewidth: "100%",
    paletteheight: "700px",
    symbolHeight: 60,
    symbolWidth: 60,
  };
}
}
</script>
<style>
  @import
  "https://ej2.syncfusion.com/vue/documentation/../../node_modules/@syncfusion/ej2-vue-diagrams/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/diagram/symbol-palette/palettes-cs1" %}

### Customize the Palette Header

Palettes can be annotated with its header texts.

The [title](#) displayed as the header text of palette.

The [expanded](#) property of palette allows to expand/collapse its palette items.

The [height](#) property of palette sets the height of the symbol group.

The [iconCss](#) property sets the content of the symbol group.

The [description](#) defines the text to be displayed and how that is to be handled in getSymbolInfo.

Also, we can embed any HTML element into a palette header by defining the getSymbolInfo property.

Following code example illustrates how to customize palette headers.

### APP.VUE

```

<template>
  <div id="app">
    <ejs-symbolpalette id="symbolpalette" :expandMode='expandMode'
    :palettes='palettes' :getSymbolInfo='getSymbolInfo' :width='palettewidth'
    :height='paletteheight' :symbolHeight='symbolHeight'
    :symbolWidth='symbolWidth'></ejs-symbolpalette>
  </div>
</template>
<script>
  import Vue from 'vue';
  import { DiagramPlugin,
    Diagram,
    NodeModel,
    UndoRedo,
    ConnectorModel,
    Node,
    Connector,
    SymbolPalette, SymbolPalettePlugin,
    SymbolInfo,
    ShapeModel} from '@syncfusion/ej2-vue-diagrams';

```

```

import { ExpandMode } from "@syncfusion/ej2-vue-navigations";
Vue.use(DiagramPlugin);
Vue.use(SymbolPalettePlugin);
let basicShapes: NodeModel[] = [{
  id: 'Rectangle',
  shape: {
    type: 'Basic',
    shape: 'Rectangle'
  }
},
{
  id: 'Ellipse',
  shape: {
    type: 'Basic',
    shape: 'Ellipse'
  }
},
{
  id: 'Hexagon',
  shape: {
    type: 'Basic',
    shape: 'Hexagon'
  }
}
];
let palette: any;
let size: any;
let expand: any;
export default {
  name: 'app'
  data() {
    return {
      //Defines how many palettes can be at expanded mode at a
time
      expandMode: "Multiple",
      //Defines the palette collection
      palettes: [{
        id: 'basic',
        expanded: true,
        symbols: basicShapes,
        title: 'Basic Shapes',
        iconCss: 'e-ddb-icons e-basic'
      }, ],
      getSymbolInfo: (symbol: NodeModel): SymbolInfo => {
        if (symbol['text'] !== undefined) {
          return {
            width: 75,
            height: 40,
            //Add or Remove the Text for Symbol palette
item.
            description: {
              //Defines the symbol description
              text: symbol['text'],
              //Defines how to handle the text when its
size exceeds the given symbol size
              overflow: 'Wrap'
            }
          }
        }
      }
    }
  }
}

```



```

        };
    }
    return {
        width: 75,
        height: 40,
        description: {
            text: symbol.shape['shape']
        }
    };
}
palettewidth: "100%",
paletteheight: "700px",
symbolHeight: 60,
symbolWidth: 60,
};
}
}
</script>
<style>
    @import "../node_modules/@syncfusion/ej2-vue-
    diagrams/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/diagram/symbol-palette/palettes-cs2" %}

### Restrict expansion of the palette panel

The symbol palette panel can be restricted from getting expanded. The `[cancel]` argument of the `[paletteExpanding]` property defines whether the palette's panel should be expanded or collapsed. By default, the panel is expanded. This restriction can be done for each of the palettes in the symbol palette as desired.

In the following code example, the basic shapes palette is restricted from getting collapsed whereas the swimlane shapes palette can be expanded or collapsed.

### APP.VUE

```

<template>
    <div id="app">
        <ejs-symbolpalette id="symbolpalette" :expandMode='expandMode'
        :palettes='palettes' :width='palettewidth' :height='paletteheight'
        :symbolPreview='symbolPreview' :symbolHeight='symbolHeight'
        :symbolWidth='symbolWidth' :getSymbolInfo='getSymbolInfo'
        :paletteExpanding='paletteExpanding'></ejs-symbolpalette>
    </div>
</template>
<script>
    import Vue from 'vue';
    import { DiagramPlugin,
        Diagram,
        NodeModel,
        UndoRedo,
        ConnectorModel,
        Node,
        Connector,
        SymbolPalette, SymbolPalettePlugin,
        SymbolInfo,

```

```

    ShapeModel, ExpandMode} from '@syncfusion/ej2-vue-diagrams';
Vue.use(DiagramPlugin);
Vue.use(SymbolPalettePlugin);
let basicShapes = [{
    id: 'Rectangle',
    shape: {
        type: 'Basic',
        shape: 'Rectangle'
    }
},
{
    id: 'Ellipse',
    shape: {
        type: 'Basic',
        shape: 'Ellipse'
    }
},
{
    id: 'Hexagon',
    shape: {
        type: 'Basic',
        shape: 'Hexagon'
    }
}
];
let swimlaneShapes = [
    {
        id: 'swimlaneShapes', expanded: true,
        title: 'Swimlane Shapes',
        symbols: [
            {
                id: 'stackCanvas1',
                shape: {
                    type: 'SwimLane', lanes: [
                        {
                            id: 'lane1',
                            style: { fill: '#f5f5f5'}, height: 60,
width: 150,
                            header: { width: 50, height: 50, style:
{fill: '#C7D4DF'} },
                        }
                    ],
                    orientation: 'Horizontal', isLane: true
                },
                height: 60,
                width: 140,
                style: { fill: '#f5f5f5'},
                offsetX: 70,
                offsetY: 30,
            }
        ]
    },
]
let palette;
let size;
let expand;
export default {
    name: 'app',

```

```

    data() {
      return {
        //Defines how many palettes can be at expanded mode at a
time
        expandMode: "Multiple",
        //Defines the palette collection
        palettes: [{
          id: 'basic',
          expanded: true,
          symbols: basicShapes,
          title: 'Basic Shapes',
          iconCss: 'e-ddb-icons e-basic'
        },
        {
          id: 'swimlane',
          expanded: true,
          symbols: swimlaneShapes,
          title: 'Swimlane Shapes',
          iconCss: 'e-ddb-icons e-basic'
        }
      ],
        palettewidth: "100%",
        paletteheight: "700px",
        symbolHeight: 50,
        symbolWidth: 50,
        symbolPreview: {
          height: 100,
          width: 100
        },
        getSymbolInfo: (symbol) => {
          return { fit: true };
        },
        paletteExpanding: (args) => {
          if(args.palette.id === 'basic') {
            // Basic shapes panel does not collapse or cexpand
            args.cancel = true;
          } else {
            if(args.palette.id === 'swimlane') {
              // Swimlane shapes panel collapse
              args.cancel = false;
            }
          }
        }
      };
    }
  }
</script>
<style>
  @import "../node_modules/@syncfusion/ej2-vue-
diagrams/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/diagram/symbol-palette/palettes-cs3" %}

### Stretch the symbols into the palette

The [fit](#) property defines whether the symbol has to be fit inside the size, that is defined by the symbol palette. For example, When we resize the rectangle in the symbol, ratio of the rectangle size has to be

maintained rather changing into square shape. The following code example illustrates how to customize the symbol size.

### APP.VUE

```
<template>
  <div id="app">
    <ejs-symbolpalette id="symbolpalette" :expandMode='expandMode'
    :palettes='palettes' :getSymbolInfo='getSymbolInfo':width='palettewidth'
    :height='paletteheight' :symbolHeight='symbolHeight'
    :symbolWidth='symbolWidth'></ejs-
symbolpalette>
  </div>
</template>
<script>
  import Vue from 'vue';
  import { DiagramPlugin,
    Diagram,
    NodeModel,
    UndoRedo,
    ConnectorModel,
    Node,
    Connector,
    SymbolPalette, SymbolPalettePlugin,
    SymbolInfo,
    ShapeModel} from '@syncfusion/ej2-vue-diagrams';
  import { ExpandMode } from "@syncfusion/ej2-vue-navigations";
  Vue.use(DiagramPlugin);
  Vue.use(SymbolPalettePlugin);
  let basicShapes: NodeModel[] = [{
    id: 'Rectangle',
    shape: {
      type: 'Basic',
      shape: 'Rectangle'
    }
  },
  {
    id: 'Ellipse',
    shape: {
      type: 'Basic',
      shape: 'Ellipse'
    }
  },
  {
    id: 'Hexagon',
    shape: {
      type: 'Basic',
      shape: 'Hexagon'
    }
  }
];
let palette: any;
let size: any;
let expand: any;
export default {
  name: 'app'
  data() {
```

```

        return {
            //Defines how many palettes can be at expanded mode at a
time
            expandMode: "Multiple",
            //Defines the palette collection
            palettes: [{
                id: 'basic',
                expanded: true,
                symbols: basicShapes,
                title: 'Basic Shapes',
                iconCss: 'e-ddb-icons e-basic'
            }, ],
            getSymbolInfo: (symbol: NodeModel): SymbolInfo => {
                // Enables to fit the content into the specified palette
item size
                return {
                    fit: true
                };
                // When it is set as false, the element is rendered with
actual node size
            },
            palettewidth: "100%",
            paletteheight: "700px",
            symbolHeight: 60,
            symbolWidth: 60,
        };
    }
}
</script>
<style>
    @import "../node_modules/@syncfusion/ej2-vue-
diagrams/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/diagram/symbol-palette/palettes-cs4" %}

### Add/Remove symbols to palette at runtime

- Symbols can be added to palette at runtime by using public method, [addPaletteltem](#).
- Symbols can be removed from palette at runtime by using public method, [removePaletteltem](#).

### Customize the size of symbols

You can customize the size of the individual symbol. The [symbolWidth](#) and [symbolHeight](#) property of node enables you to define the size of the symbols. The following code example illustrates how to change the size of a symbol.

#### APP.VUE

```

<template>
    <div id="app">
        <ejs-symbolpalette id="symbolpalette" :expandMode='expandMode'
:palettes='palettes' :symbolMargin='symbolMargin' :width='palettewidth'
:height='paletteheight' :symbolHeight='symbolHeight'
:symbolWidth='symbolWidth'></ejs-
symbolpalette>

```

```

</div>
</template>
<script>
  import Vue from 'vue';
  import { DiagramPlugin,
    Diagram,
    NodeModel,
    UndoRedo,
    ConnectorModel,
    Node,
    Connector,
    SymbolPalette, SymbolPalettePlugin,
    SymbolInfo,
    ShapeModel } from '@syncfusion/ej2-vue-diagrams';
  import { ExpandMode } from "@syncfusion/ej2-vue-navigations";
  Vue.use(DiagramPlugin);
  Vue.use(SymbolPalettePlugin);
  let basicShapes: NodeModel[] = [{
    id: 'Rectangle',
    shape: {
      type: 'Basic',
      shape: 'Rectangle'
    }
  },
  {
    id: 'Ellipse',
    shape: {
      type: 'Basic',
      shape: 'Ellipse'
    }
  },
  {
    id: 'Hexagon',
    shape: {
      type: 'Basic',
      shape: 'Hexagon'
    }
  }
  ];
  let palette: any;
  let size: any;
  let expand: any;
  export default {
    name: 'app'
    data() {
      return {
        //Defines how many palettes can be at expanded mode at a
time
        expandMode: "Multiple",
        //Defines the palette collection
        palettes: [{
          id: 'basic',
          expanded: true,
          symbols: basicShapes,
          title: 'Basic Shapes',
          iconCss: 'e-ddb-icons e-basic'
        }, ],
      };
    }
  };

```

```

        symbolMargin: {
            left: 15,
            right: 15,
            top: 15,
            bottom: 15
        },
        palettewidth: "100%",
        paletteheight: "700px",
        symbolHeight: 60,
        symbolWidth: 60,
    };
}
}
</script>
<style>
    @import
    "https://ej2.syncfusion.com/vue/documentation/../../node_modules/@syncfusion/ej2-vue-diagrams/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/diagram/symbol-palette/palettes-cs5" %}

The [symbolMargin](#) property is used to create the space around elements, outside of any defined borders.

### Symbol Preview

The Symbol Preview size of the palette items can be customized using [symbolPreview](#).

The [width](#) and [height](#) property of SymbolPalette enables you to define the preview size to all the symbol palette items.

[offset](#) of the dragging helper relative to the mouse cursor.

The following code example illustrates how to change the preview size of a palette item.

### APP.VUE

```

<template>
    <div id="app">
        <ejs-symbolpalette id="symbolpalette" :expandMode='expandMode'
        :palettes='palettes'
            :getSymbolInfo='getSymbolInfo' :symbolMargin='symbolMargin'
        :width='palettewidth' :symbolPreview='symbolPreview' :height='paletteheight'
        :symbolHeight='symbolHeight'
            :symbolWidth='symbolWidth'></ejs-
symbolpalette>
    </div>
</template>
<script>
    import Vue from 'vue';
    import { DiagramPlugin,
        Diagram,
        NodeModel,
        UndoRedo,
        ConnectorModel,
        Node,
        Connector,

```

```

    SymbolPalette, SymbolPalettePlugin,
    SymbolInfo,
    ShapeModel} from '@syncfusion/ej2-vue-diagrams';
import { ExpandMode } from "@syncfusion/ej2-vue-navigations";
Vue.use(DiagramPlugin);
Vue.use(SymbolPalettePlugin);
let basicShapes: NodeModel[] = [{
  id: 'Rectangle',
  shape: {
    type: 'Basic',
    shape: 'Rectangle'
  }
},
{
  id: 'Ellipse',
  shape: {
    type: 'Basic',
    shape: 'Ellipse'
  }
},
{
  id: 'Hexagon',
  shape: {
    type: 'Basic',
    shape: 'Hexagon'
  }
}
];
let palette: any;
let size: any;
let expand: any;
export default {
  name: 'app'
  data() {
    return {
      //Defines how many palettes can be at expanded mode at a
time
      expandMode: "Multiple",
      //Defines the palette collection
      palettes: [{
        id: 'basic',
        expanded: true,
        symbols: basicShapes,
        title: 'Basic Shapes',
        iconCss: 'e-ddb-icons e-basic'
      }, ],
      symbolMargin: {
        left: 15,
        right: 15,
        top: 15,
        bottom: 15
      },
      getSymbolInfo: (symbol: NodeModel): SymbolInfo => {
        return {
          fit: true
        };
      },
    },
  },

```



```

        symbolPreview: {
            height: 100,
            width: 100,
            offset: {
                x: 0.5,
                y: 0.5
            }
        },
        palettewidth: "100%",
        paletteheight: "700px",
        symbolHeight: 60,
        symbolWidth: 60,
    };
}
}
</script>
<style>
    @import "../node_modules/@syncfusion/ej2-vue-
    diagrams/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/diagram/symbol-palette/palettes-cs6" %}

### Default Settings

While adding more number of symbols such as nodes and connectors to the palette, you can define the default settings for those objects through [getNodeDefaults](#) property of defaultSettings allows to define the default settings for nodes and [getConnectorDefaults](#) property of defaultSettings allows to define the default settings for connectors.

### APP.VUE

```

<template>
    <div id="app">
        <ejs-symbolpalette id="symbolpalette" :expandMode='expandMode'
        :palettes='palettes'
            :getSymbolInfo='getSymbolInfo' :symbolMargin='symbolMargin'
        :width='palettewidth' :getNodeDefaults='palettegetNodeDefaults'
        :symbolPreview='symbolPreview' :height='paletteheight'
        :symbolHeight='symbolHeight'
                                :symbolWidth='symbolWidth'></ejs-
symbolpalette>
    </div>
</template>
<script>
    import Vue from 'vue';
    import { DiagramPlugin,
        Diagram,
        NodeModel,
        UndoRedo,
        ConnectorModel,
        Node,
        Connector,
        SymbolPalette, SymbolPalettePlugin,
        SymbolInfo,
        ShapeModel } from '@syncfusion/ej2-vue-diagrams';
    import { ExpandMode } from "@syncfusion/ej2-vue-navigations";

```

```

Vue.use(DiagramPlugin);
Vue.use(SymbolPalettePlugin);
let basicShapes: NodeModel[] = [{
  id: 'Rectangle',
  shape: {
    type: 'Basic',
    shape: 'Rectangle'
  }
},
{
  id: 'Ellipse',
  shape: {
    type: 'Basic',
    shape: 'Ellipse'
  }
},
{
  id: 'Hexagon',
  shape: {
    type: 'Basic',
    shape: 'Hexagon'
  }
}
];
let palette: any;
let size: any;
let expand: any;
export default {
  name: 'app'
  data() {
    return {
      //Defines how many palettes can be at expanded mode at a
time
      expandMode: "Multiple",
      //Defines the palette collection
      palettes: [{
        id: 'basic',
        expanded: true,
        symbols: basicShapes,
        title: 'Basic Shapes',
        iconCss: 'e-ddb-icons e-basic'
      }, ],
      symbolMargin: {
        left: 15,
        right: 15,
        top: 15,
        bottom: 15
      },
      palettegetNodeDefaults: (node: NodeModel): void => {
        node.width = 100;
        node.height = 100;
        node.style.strokeColor = '#3A3A3A';
      },
      getSymbolInfo: (symbol: NodeModel): SymbolInfo => {
        return {
          fit: true
        };
      };
    };
  }
};

```

```

        },
        symbolPreview: {
            height: 100,
            width: 100,
            offset: {
                x: 0.5,
                y: 0.5
            }
        },
        palettewidth: "100%",
        paletteheight: "700px",
        symbolHeight: 60,
        symbolWidth: 60,
    };
}
}
</script>
<style>
    @import "../node_modules/@syncfusion/ej2-vue-
    diagrams/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/diagram/symbol-palette/palettes-cs7" %}

#### Adding symbol description for symbols in the palette

The diagram provides support to add symbol description below each symbol of a palette. This descriptive representation of each symbol will enhance the details of the symbol visually. The height and width of the symbol description can also be set individually.

- The property `getSymbolInfo`, can be used to add the symbol description at runtime.

The following code is an example to set a symbol description for symbols in the palette.

#### APP.VUE

```

<template>
    <div id="app">
        <ejs-symbolpalette id="symbolpalette" :expandMode='expandMode'
        :palettes='palettes' :width='palettewidth' :height='paletteheight'
        :symbolHeight='symbolHeight' :symbolWidth='symbolWidth'
        :getSymbolInfo='getSymbolInfo'
        ></ejs-symbolpalette>
    </div>
</template>
<script>
    import Vue from 'vue';
    import { DiagramPlugin,
        Diagram,
        NodeModel,
        UndoRedo,
        ConnectorModel,
        Node,
        Connector,
        SymbolPalette, SymbolPalettePlugin,
        SymbolInfo,

```

```

    ShapeModel, ExpandMode} from '@syncfusion/ej2-vue-diagrams';
Vue.use(DiagramPlugin);
Vue.use(SymbolPalettePlugin);
let basicShapes = [{
  id: 'Rectangle',
  shape: {
    type: 'Basic',
    shape: 'Rectangle'
  }, style: { strokeWidth: 2 }
},
{
  id: 'Ellipse',
  shape: {
    type: 'Basic',
    shape: 'Ellipse'
  }, style: { strokeWidth: 2 }
},
{
  id: 'Hexagon',
  shape: {
    type: 'Basic',
    shape: 'Hexagon'
  }, style: { strokeWidth: 2 }
}
];
let swimlaneShapes = [
  {
    id: 'swimlaneShapes', expanded: true,
    title: 'Swimlane Shapes',
    symbols: [
      {
        id: 'stackCanvas1',
        shape: {
          type: 'SwimLane', lanes: [
            {
              id: 'lane1',
              style: { fill: '#f5f5f5'}, height: 60, width:
150,
              header: { width: 50, height: 50, style:
{fill: '#C7D4DF'} },
            }
          ],
          orientation: 'Horizontal', isLane: true
        },
        height: 60,
        width: 140,
        style: { fill: '#f5f5f5'},
        offsetX: 70,
        offsetY: 30,
      }
    ]
  },
]
let palette;
let size;
let expand;
export default {
  name: 'app',

```

```

    data() {
      return {
        //Defines how many palettes can be at expanded mode at a
time
        expandMode: "Multiple",
        //Defines the palette collection
        palettes: [{
          id: 'basic',
          expanded: true,
          symbols: basicShapes,
          title: 'Basic Shapes',
          iconCss: 'e-ddb-icons e-basic'
        },
        {
          id: 'swimlane',
          expanded: true,
          symbols: swimlaneShapes,
          title: 'Swimlane Shapes',
          iconCss: 'e-ddb-icons e-basic'
        }
      ],
        palettewidth: "100%",
        paletteheight: "700px",
        symbolHeight: 80,
        symbolWidth: 80,
        getSymbolInfo: (symbol) => {
          //Sets the description for symbols
          return { width: 75, height: 40, description: { text:
symbol.shape['shape'] } } };
        }
      }
    }
  </script>
  <style>
    @import "../node_modules/@syncfusion/ej2-vue-
diagrams/styles/material.css";
  </style>

```

{% previewsample "page.domainurl/code-snippet/diagram/symbol-palette/palettes-cs8" %}

### Appearance of symbol description

The appearance of a symbol description in the palette can be customized by changing its **color**, **fontSize**, **fontFamily**, **bold**, **italic**, **textDecoration**, and **margin**.

The following code is an example to change the color of a symbol description for symbols in the palette.

### APP.VUE

```

<template>
  <div id="app">
    <ejs-symbolpalette id="symbolpalette" :palettes='palettes'
:width='palettewidth' :height='paletteheight' :symbolMargin='symbolMargin'
:getNodeDefaults='palettegetNodeDefaults'
:symbolHeight='symbolHeight' :symbolWidth='symbolWidth'
:getSymbolInfo='getSymbolInfo'
></ejs-symbolpalette>

```

```

</div>
</template>
<script>
  import Vue from 'vue';
  import { DiagramPlugin,
    Diagram,
    NodeModel,
    UndoRedo,
    ConnectorModel,
    Node,
    Connector,
    SymbolPalette, SymbolPalettePlugin,
    SymbolInfo, NodeConstraints,
    ShapeModel, ExpandMode } from '@syncfusion/ej2-vue-diagrams';
  Vue.use(DiagramPlugin);
  Vue.use(SymbolPalettePlugin);
  let umlShapes = [
    {
      id: 'class',
      style: {
        fill: '#26A0DA',
      },
      borderColor: 'white',
      shape: {
        type: 'UmlClassifier',
        classShape: {
          attributes: [
            { name: 'accepted', type: 'Date', style: {
color: "red", fontFamily: "Arial", textDecoration: 'Underline', italic:
true }, isSeparator: true },
          ],
          methods: [{ name: 'getHistory', style: {}},
parameters: [{ name: 'Date', style: {} }], type: 'History' }],
          name: 'Patient'
        },
        classifier: 'Class'
      },
    },
    {
      id: 'Interface',
      style: {
        fill: '#26A0DA',
      },
      borderColor: 'white',
      shape: {
        type: 'UmlClassifier',
        interfaceShape: {
          name: "Bank Account",
        },
        classifier: 'Interface'
      },
    },
    {
      id: 'Enumeration',
      style: {
        fill: '#26A0DA',
      },
      borderColor: 'white',
      shape: {

```

```

        type: 'UmlClassifier',
        enumerationShape: {
          name: 'AccountType',
          members: [
            {
              name: 'Checking Account', style: {}
            },
          ],
        },
        classifier: 'Enumeration'
      },
    ],
  ];

  let palette;
  export default {
    name: 'app',
    data() {
      return {
        //Defines the palette collection
        palettes: [{
          id: 'uml',
          expanded: true,
          symbols: umlShapes,
          title: 'UML Shapes',
        },
        ],
        palettewidth: "100%",
        paletteheight: "700px",
        symbolHeight: 80,
        symbolWidth: 80,
        symbolMargin: { left: 12, right: 12, top: 12, bottom: 12 },
        palettegetNodeDefaults: (symbol) => {
          symbol.width = 100;
          symbol.height = 100;
        },
        getSymbolInfo: (symbol) => {
          //Sets the description for symbols
          return { fit: true, description: { text: symbol.id, } };
        }
      };
    }
  }
</script>
<style>
  @import
  "https://ej2.syncfusion.com/vue/documentation/../../node_modules/@syncfusion/ej2-vue-diagrams/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/diagram/symbol-palette/palettes-cs9" %}

### Tooltip for symbols in symbol palette

The Symbol palette supports displaying tooltips when mouse hovers over the symbols. You can customize the tooltip for each symbol in the symbol palette.

### Default tooltip for symbols

When hovering over symbols in the symbol palette, the default tooltip displays the symbol's ID.

Refer to the image below for an illustration of the tooltip behavior in the symbol palette.



### Custom tooltip for symbols

To customize the tooltips for symbols in the symbol palette, assign a custom tooltip to the 'Tooltip' content property of each symbol. Once you define the custom tooltip, enable the Tooltip constraints for each symbol, ensuring that the tooltips are displayed when users hover over them.

Here, the code provided below demonstrates how to define tooltip content to symbols within a symbol palette.

#### **APP.VUE**

```
<template>
  <div id="app">
    <ejs-symbolpalette id="symbolpalette" :palettes='palettes'
    :width='palettewidth' :height='paletteheight' :symbolMargin='symbolMargin'
    :getNodeDefaults='palettegetNodeDefaults'
    :symbolHeight='symbolHeight' :symbolWidth='symbolWidth'
    :getSymbolInfo='getSymbolInfo'
    ></ejs-symbolpalette>
  </div>
</template>
<script>
  import Vue from 'vue';
  import { DiagramPlugin,
    Diagram,
    NodeModel,
    UndoRedo,
    ConnectorModel,
    Node,
```



```

Connector,
SymbolPalette, SymbolPalettePlugin,
SymbolInfo, NodeConstraints,
ShapeModel, ExpandMode} from '@syncfusion/ej2-vue-diagrams';
Vue.use(DiagramPlugin);
Vue.use(SymbolPalettePlugin);
let umlShapes = [
  {
    id: 'class',
    style: {
      fill: '#26A0DA',
    },
    borderColor: 'white',
    shape: {
      type: 'UmlClassifier',
      classShape: {
        attributes: [
          { name: 'accepted', type: 'Date', style: {
color: "red", fontFamily: "Arial", textDecoration: 'Underline', italic:
true }, isSeparator: true },
        ],
        methods: [{ name: 'getHistory', style: {}},
parameters: [{ name: 'Date', style: {} }], type: 'History' }],
        name: 'Patient'
      },
      classifier: 'Class'
    },
  },
  {
    id: 'Interface',
    style: {
      fill: '#26A0DA',
    },
    borderColor: 'white',
    shape: {
      type: 'UmlClassifier',
      interfaceShape: {
        name: "Bank Account",
      },
      classifier: 'Interface'
    },
  },
  {
    id: 'Enumeration',
    style: {
      fill: '#26A0DA',
    },
    borderColor: 'white',
    shape: {
      type: 'UmlClassifier',
      enumerationShape: {
        name: 'AccountType',
        members: [
          {
            name: 'Checking Account', style: {}
          },
        ],
      },
      classifier: 'Enumeration'
    },
  },

```

```

    },
  },
];

let palette;
export default {
  name: 'app',
  data() {
    return {
      //Defines the palette collection
      palettes: [{
        id: 'uml',
        expanded: true,
        symbols: umlShapes,
        title: 'UML Shapes',
      }],
      palettewidth: "100%",
      paletteheight: "700px",
      symbolHeight: 80,
      symbolWidth: 80,
      symbolMargin: { left: 12, right: 12, top: 12, bottom: 12 },
      palettegetNodeDefaults: (symbol) => {
        symbol.width = 100;
        symbol.height = 100;
      },
      getSymbolInfo: (symbol) => {
        //Sets the description for symbols
        return { fit: true, description: { text: symbol.id, } };
      }
    };
  }
}
</script>
<style>
  @import
  "https://ej2.syncfusion.com/vue/documentation/../../node_modules/@syncfusion/ej2-vue-diagrams/styles/material.css";
</style>

```

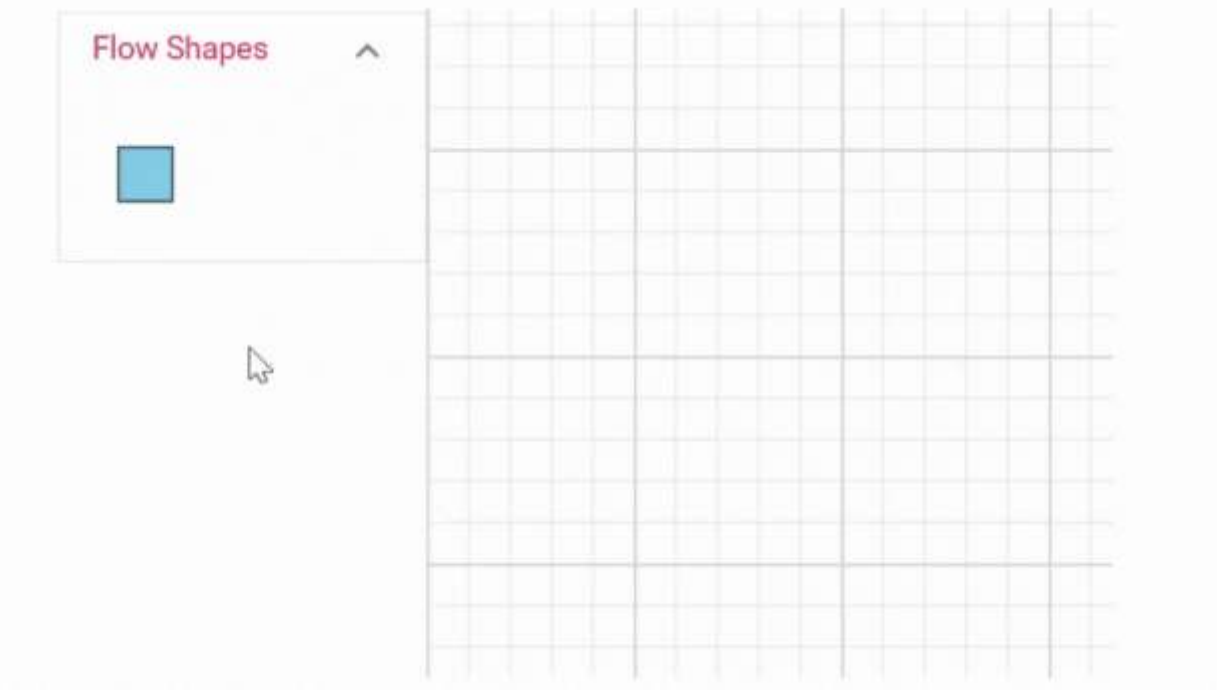
{% previewsample "page.domainurl/code-snippet/diagram/symbol-palette/palettes-cs9" %}

#### [How to provide different tooltip for Symbol palette and diagram elements.](#)

Differentiate the tooltips between symbols in the symbol palette and dropped nodes by utilizing the `dragEnter` event. When a custom tooltip is defined for a symbol, it will be displayed for both the symbol and the dropped node in the diagram canvas. However, to provide distinct tooltips for symbols in the palette and dropped nodes, capture the `dragEnter` event and assign specific tooltips dynamically.

When a symbol is dragged from the symbol palette and enters the diagram canvas, the `[DragEnter]` [IDragEnterEventArgs](#) event is triggered. Within this event, you can define a new tooltip for the dropped node. By assigning custom tooltip content to the `Tooltip` property of the node, you can provide a distinct tooltip that is specific to the dropped node.

The following image illustrates the differentiation of tooltips displayed in the Symbol Palette and the Diagram.



The following code snippet will demonstrate how to define two different tooltip for symbol in the symbol palette and dropped node in the diagram canvas.

```
`ts
let diagram: Diagram = new Diagram({
width: '100%', height: '500px',
connectors: connectors, nodes: nodes,
//event to change tooltip content while dragging symbols into Diagram
dragEnter: dragEnter,
});
diagram.appendTo('#diagram');
function dragEnter(args:IDragEnterEventArgs)
{
//enable tooltip constraints for the dragged symbol
args.dragItem.constraints = NodeConstraints.Default | NodeConstraints.Tooltip;
//change the tooltip content of the dragged symbol
args.dragItem.tooltip.content='This is Diagram Tooltip';
}
`
```

#### Palette Interaction

Palette interaction notifies the element enter, leave, and dragging of the symbols into the diagram.

### DragEnter

[DragEnter] [IDragEnterEventArgs](#) notifies when the element enter into the diagram from symbol palette.

### DragLeave

[DragLeave] [IDragLeaveEventArgs](#) notifies when the element leaves from the diagram.

### DragOver

[DragOver] [IDragOverEventArgs](#) notifies when an element drag over another diagram element.

Note: The diagram provides support to cancel the drag and drop operation from the symbol palette to the diagram when the ESC key is pressed.

### See Also

- [How to add the symbol to the diagram](#)

## Overview in Vue Diagram component

Overview control allows you to see a preview or an overall view of the entire content of a diagram. This helps you to look at the overall picture of a large diagram and also to navigate, pan, or zoom, on a particular position of the page.

When you work on a very large diagram, you may not know the part you are actually working on, or navigation from one part to another might be difficult. One solution for navigation is to zoom out the entire diagram and find where you are. Then, you can zoom in a particular area you want to. This solution is not suitable when you need some frequent navigation.

Overview control solves these problems by showing a preview, that is, an overall view of the entire diagram. A rectangle indicates viewport of the diagram. Navigation becomes easy by dragging this rectangle.

### Create overview

The `sourceID` property of overview should be set with the corresponding diagram ID for the overall view.

The `width` and `height` properties of the overview allow you to define the size of the overview.

The following code illustrates how to create overview.

### Zoom and Pan

In overview, the view port of the diagram is highlighted with a red colored rectangle. Diagram can be zoomed/panned by interacting with that. You can interact with overview as follows:

- Resize the rectangle: Zooms in/out the diagram.
- Drag the rectangle: Pans the diagram.
- Click at a position: Navigates to the clicked region.
- Choose a particular region by clicking and dragging: Navigates to the specified region.

The following image shows how the diagram is zoomed/panned with overview.

### APP.VUE

```
<template>
```

```

<div id="app" >
  <ejs-diagram id="diagram" :width='width' :height='height'
:getNodeDefaults='getNodeDefaults'
:getConnectorDefaults='getConnectorDefaults' :snapSettings='snapSettings'
:layout='layout' :dataSourceSettings='dataSourceSettings' ></ejs-
diagram>
  <div
style="right:0px;bottom:0px;background:#D5D5D5;position:absolute">
    <ejs-overview id="overview" style="top:30px"
      :sourceID='overviewsourceID'
      :width='overviewwidth'
      :height='overviewheight'></ejs-overview>
    <div>

      </div>
    </template>
  <script>
    import Vue from 'vue';
    import { DiagramPlugin, HierarchicalTree,
DataBinding,Diagram,OverviewPlugin,Overview,
ConnectorModel,
Overview,
OverviewModel,TreeInfo,
Node,
StackPanel,
ImageElement,
Container,
TextElement,
DataBinding,
HierarchicalTree } from '@syncfusion/ej2-vue-diagrams';
    import { DataManager,Query } from "@syncfusion/ej2-data";
    Vue.use(DiagramPlugin);
    Vue.use(OverviewPlugin);
    /**
     * Overview
     */
    let diagram: Diagram;
    let overview: Overview;
    Diagram.Inject(DataBinding, HierarchicalTree);
    let data: object[] = [{
      'Id': 'parent',
      'Name': 'Maria Anders',
      'Designation': 'Managing Director',
      'IsExpand': 'true',
      'RatingColor': '#C34444'
    },
    {
      'Id': 1,
      'Name': 'Ana Trujillo',
      'Designation': 'Project Manager',
      'IsExpand': 'false',
      'RatingColor': '#68C2DE',
      'ReportingPerson': 'parent'
    },
    {
      'Id': 2,
      'Name': 'Anto Moreno',
      'Designation': 'Project Lead',

```

```

        'IsExpand': 'false',
        'RatingColor': '#93B85A',
        'ReportingPerson': 'parent'
    },
    {
        'Id': 3,
        'Name': 'Thomas Hardy',
        'Designation': 'Senior S/w Engg',
        'IsExpand': 'false',
        'RatingColor': '#68C2DE',
        'ReportingPerson': 1
    },
    {
        'Id': 4,
        'Name': 'Christina kaff',
        'Designation': 'S/w Engg',
        'IsExpand': 'false',
        'RatingColor': '#93B85A',
        'ReportingPerson': 2
    },
    {
        'Id': 5,
        'Name': 'Hanna Moos',
        'Designation': 'Project Trainee',
        'IsExpand': 'true',
        'RatingColor': '#D46E89',
        'ReportingPerson': 2
    }
],
let items: DataManager = new DataManager(data as JSON[], new
Query().take(7));
export default {
    name: 'app',
    data () {
        return {
            width: "100%",
            height: "590px",
            snapSettings: {
                constraints: 0
            },
        },
        layout: {
            type: 'OrganizationalChart',
            margin: {
                top: 20
            },
        },
        getLayoutInfo: (node: Node, tree: TreeInfo) => {
            if (!tree.hasSubTree) {
                tree.orientation = 'Vertical';
                tree.type = 'Alternate';
            }
        }
    },
    dataSourceSettings: {
        id: 'Id',
        parentId: 'ReportingPerson',
        dataManager: items
    },
},

```

```
overviewsourceID: "diagram",
overviewwidth: "100%",
overviewheight: "150px",
getNodeDefaults: (node: NodeModel) => {
  node.height = 50;
  node.style.fill = '#6BA5D7';
  node.style.borderColor = 'white';
  node.style.strokeColor = 'white';
  return node;
},
getConnectorDefaults: (obj: ConnectorModel): ConnectorModel => {
  obj.style.strokeColor = '#6BA5D7';
  obj.style.fill = '#6BA5D7';
  obj.style.strokeWidth = 2;
  obj.targetDecorator.style.fill = '#6BA5D7';
  obj.targetDecorator.style.strokeColor = '#6BA5D7';
  obj.targetDecorator.shape = 'None';
  obj.type = 'Orthogonal';
  return obj;
},
setNodeTemplate: (obj: Node, diagram: Diagram): Container => {
  let content: StackPanel = new StackPanel();
  content.id = obj.id + '_outerstack';
  content.style.strokeColor = 'darkgreen';
  content.style.fill = '#6BA5D7';
  content.orientation = 'Horizontal';
  content.padding = {
    left: 5,
    right: 10,
    top: 5,
    bottom: 5
  };
  let innerStack: StackPanel = new StackPanel();
  innerStack.style.strokeColor = 'none';
  innerStack.style.fill = '#6BA5D7';
  innerStack.margin = {
    left: 5,
    right: 0,
    top: 0,
    bottom: 0
  };
  innerStack.id = obj.id + '_innerstack';
  let text: TextElement = new TextElement();
  text.content = obj.data['Name'];
  text.style.color = 'white';
  text.style.strokeColor = 'none';
  text.style.fill = 'none';
  text.id = obj.id + '_text1';
  let desigText: TextElement = new TextElement();
  desigText.margin = {
    left: 0,
    right: 0,
    top: 5,
    bottom: 0
  };
  desigText.content = obj.data['Designation'];
  desigText.style.color = 'white';
```

```

        desigText.style.strokeColor = 'none';
        desigText.style.fill = 'none';
        desigText.style.textWrapping = 'Wrap';
        desigText.id = obj.id + '_desig';
        innerStack.children = [text, desigText];
        content.children = [innerStack];
        return content;
    }
    },
    provide: {diagram: [DataBinding, HierarchicalTree]},
  }
</script>
<style>
  @import "../node_modules/@syncfusion/ej2-vue-
  diagrams/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/diagram/overview/overview-cs1" %}

### How to load EJ1 diagram in EJ2 diagram

To load EJ1 JSON data in an EJ2 diagram, follow these steps.

1. Import and inject the EJ1SerializationModule as shown in the following code example.

```

`html
<template>
<div id="app">
<ejs-diagram ref="diagramObject" id="diagram" :width='width':height='height'></ejs-diagram>
</div>
</template>
<script>
export default {
name: 'app'
data() {
return {
width: "100%",
height: "350px",
}
},
}
</script>
`

```



2. Load the EJ1 JSON data using the diagram loadDiagram method and set the second parameter to true.

```
`html
<script>
export default {
  mounted: function() {
    let diagram = document.getElementById("diagram").ej2_instances[0];
    let ej1Data = {"JSONData"}; //Replace JSONData with your EJ1 JSON data
    //Load the EJ1 JSON and pass a boolean value as true.
    diagram.loadDiagram(ej1Data, true);
  }
}
</script>
`
```

## Dialog

### Getting Started

This section explains how to create a simple Dialog and how to configure the Dialog component.

#### Prerequisites

[System requirements for Syncfusion Vue UI components](#)

#### Dependencies

The list of dependencies required to use the Dialog component in your application is given below:

```
`javascript
|-- @syncfusion/ej2-vue-popups
|-- @syncfusion/ej2-vue-base
|-- @syncfusion/ej2-vue-buttons
|-- @syncfusion/ej2-popups
|-- @syncfusion/ej2-base
|-- @syncfusion/ej2-buttons
`
```

#### Get Started with Vue CLI

You can use [Vue CLI](#) to setup your vue applications.

To install Vue CLI use the following command.

```
`bash
npm install -g @vue/cli
`
```

Start a new project using below Vue CLI command.

```
`bash
vue init webpack-simple quickstart
cd quickstart
npm install
`
```

### Adding Syncfusion packages

All the available Essential JS 2 packages are published in [npmjs.com](https://www.npmjs.com) registry. You can choose the component that you want to install. For this application, we are going to use Dialog component.

To install Dialog component, use the following command

```
`bash
npm install @syncfusion/ej2-vue-popups --save
`
```

### Registering Vue Component

For Registering Vue Component two ways are available. They are as follows.

- `Vue.use()`
- `Vue.component()`

#### Using `Vue.use()`

Import the Component Plugin from the EJ2 Vue Package and register the same using `Vue.use()` with Component Plugin as its argument.

Refer the code snippet given below.

```
`ts
import { DialogPlugin } from '@syncfusion/ej2-vue-popups';
Vue.use(DialogPlugin);
`
```

By Registering Component Plugin in Vue, all child directives are also globally registered.

#### Using `Vue.component()`

Import the Component and Component Plugin from EJ2 Vue Package, register the same using the `Vue.component()` with name of Component from Component Plugin and the EJ2 Vue Component as its arguments.

Refer the code snippet given below.

```
`ts
import { DialogComponent, DialogPlugin } from '@syncfusion/ej2-vue-popups';
Vue.component(DialogPlugin.name, DialogComponent);
`
```

By using `Vue.component()`, only the EJ2 Vue Component is registered. Child directives needs to be registered separately.

#### Creating Vue sample

Add the EJ2 Vue Dialog using `<ejs-dialog>` to the `<template>` section of the `App.vue` file in `src` directory, the content attribute of the Dialog component is provided as name in data option in the `<script>` section.

```
,  
  
<template>  
<div id="app">  
<ejs-dialog :header="Dialog" :content="content" ></ejs-dialog>  
</div>  
</template>  
  
<script>  
import Vue from 'vue';  
import { DialogPlugin } from '@syncfusion/ej2-vue-popups';  
Vue.use(DialogPlugin);  
export default {  
  name: 'app',  
  data () {  
    return {  
      content: 'The dialog component is used to display information and get input from the user.'  
    }  
  }  
}  
</script>  
,
```

#### Adding CSS reference

Add Dialog component's styles as given below in `<style>` section of the `App.vue` file.

```
,  
  
<style>  
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";  
@import "../node_modules/@syncfusion/ej2-vue-buttons/styles/material.css";  
@import "../node_modules/@syncfusion/ej2-vue-popups/styles/material.css";  
</style>
```

The [Custom Resource Generator \(CRG\)](#) is an online web tool, which can be used to generate the custom script and styles for a set of specific components.

This web tool is useful to combine the required component scripts and styles in a single file.

### Running the application

Now run the `npm run dev` command in the console, it will build your application and open in the browser.

### APP.VUE

```
<template>
  <div>
    <div id="target" class="control-section">
      <ejs-dialog :target='target' :width='width' :content='content'>
        </ejs-dialog>
      </div>
    </div>
  </template>
<script>
import Vue from "vue";
import { DialogPlugin } from '@syncfusion/ej2-vue-popups';
Vue.use(DialogPlugin);
export default {
  data: function() {
    return {
      target: "#target",
      width: '335px',
      content: 'This is a Dialog with content.'
    }
  }
}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-vue-popups/styles/material.css";
.control-section {
  height: 100%;
  min-height: 200px;
}
</style>
```

```
{% previewsample "page.domainurl/code-snippet/dialog/getting-started-cs1" %}
```

In the dialog control, max-height is calculated based on the dialog target element height. If the target property is not configured, the document.body is considered as a target. Therefore, to show a dialog in proper height, you need to add min-height to the target element.

If the dialog is rendered based on the body, then the dialog will get the height based on its body element height. If the height of the dialog is larger than the body height, then the dialog's height will not be set. For this scenario, we can set the CSS style for the html and body to get the dialog height.

```
html, body {
```

height: 100%;

}

,

### Modal dialog

A modal shows an overlay behind the Dialog. So, the user should interact the Dialog compulsory before interacting with the remaining content in an application.

While the user clicks the overlay, the action can be handled through the [overlayClick](#) event. In the below sample, the Dialog close action is performed while clicking on the overlay.

When the modal dialog is opened, the Dialog's target scrolling will be disabled. The scrolling will be enabled again once close the Dialog.

### APP.VUE

```
<template>
  <div>
    <div id="modalTarget" class="control-section; position:relative"
style="height:350px;">
      <!-- Render Button to open the modal Dialog -->
      <ejs-button id='modalbtn' v-
on:click.native="modalBtnClick">Open</ejs-button>
      <!-- Render modal Dialog -->
      <ejs-dialog ref="modalDialog" :isModal='isModal' :header='header'
:target='target' :width='width' :animationSettings='animationSettings'
:content='content' :open="modalDlgOpen" :close="modalDlgClose"
:overlayClick="overlayClick">
        </ejs-dialog>
      </div>
    </div>
  </template>
<script>
import Vue from 'vue';
import { DialogPlugin } from '@syncfusion/ej2-vue-popups';
import { ButtonPlugin } from '@syncfusion/ej2-vue-buttons';
Vue.use(DialogPlugin);
Vue.use(ButtonPlugin);
export default {
  data: function() {
    return {
      target: "#modalTarget",
      width: '335px',
      header: 'Software Update',
      content: 'Your current software version is up to date.',
      isModal: true,
      animationSettings: { effect: 'None' }
    }
  },
  mounted: function(){
    document.getElementById('modalbtn').focus();
  },
  methods: {
    modalBtnClick: function() {
      this.$refs.modalDialog.show();
    }
  }
}
```

```

        modalDlgClose: function() {
            document.getElementById('modalbtn').style.display = '';
        },
        modalDlgOpen: function() {
            document.getElementById('modalbtn').style.display = 'none';
        },
        overlayClick: function() {
            this.$refs.modalDialog.hide();
        }
    }
}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-vue-popups/styles/material.css";
#app {
    color: #008cff;
    height: 40px;
    left: 45%;
    position: absolute;
    top: 45%;
    width: 30%;
}
.control-section {
    height: 100%;
    min-height: 200px;
}
</style>

```

{% previewsample "page.domainurl/code-snippet/dialog/modal-cs1" %}

### Enable header

The Dialog header can be enabled by adding the header content as text or HTML content through the [header](#) property.

### APP.VUE

```

<template>
  <div>
    <div id="target" class="control-section; position:relative"
    style="height:350px;">
      <!-- Render Dialog -->
      <ejs-dialog ref="headerDialog" :header='header' :target='target'
      :width='width' :content='content'>
        </ejs-dialog>
    </div>
  </div>
</template>
<script>
import Vue from "vue";
import { DialogPlugin } from '@syncfusion/ej2-vue-popups';
import { ButtonPlugin } from '@syncfusion/ej2-vue-buttons';
Vue.use(DialogPlugin);
Vue.use(ButtonPlugin);
export default {
  data: function() {
    return {

```

```

        target: "#target",
        width: '335px',
        header: 'Dialog header',
        content: 'This is a Dialog with header.'
    }
}
}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-vue-popups/styles/material.css";
#app {
    color: #008cff;
    height: 40px;
    left: 45%;
    position: absolute;
    top: 45%;
    width: 30%;
}
.control-section {
    height: 100%;
    min-height: 200px;
}
</style>

```

{% previewsample "page.domainurl/code-snippet/dialog/header-cs1" %}

### Enable footer

The Dialog provides built-in support to render the buttons on the footer (for ex: 'OK' or 'Cancel' buttons). Each Dialog button allows the user to perform any action while clicking on it.

The primary button will be focused automatically on open the Dialog, and add the `click` event to handle the actions

When the Dialog initialize with more than one primary buttons, the first primary button gets focus on open the Dialog.

The below sample is rendered with button and its click event.

### APP.VUE

```

<template>
  <div>
    <div id="target" class="control-section; position:relative"
    style="height:350px;">
      <!-- Render Button to open the Dialog -->
      <ejs-button id='dlgbtn' v-on:click.native="btnClick">Open</ejs-
button>
      <!-- Render Dialog -->
      <ejs-dialog ref="footerDialog" :header='header' :target='target'
:width='width' :buttons='buttons' :content='content' :open="dlgOpen"
:close="dlgClose">
        </ejs-dialog>
    </div>
  </div>
</template>
<script>

```

```

import Vue from "vue";
import { DialogPlugin } from '@syncfusion/ej2-vue-popups';
import { ButtonPlugin } from '@syncfusion/ej2-vue-buttons';
Vue.use(DialogPlugin);
Vue.use(ButtonPlugin);
export default {
  data: function() {
    return {
      target: "#target",
      width: '335px',
      header: 'Dialog',
      content: 'This is a Dialog with button and primary button.',
      buttons: [{ click: this.dlgButtonClick, buttonModel: { content:
'OK', isPrimary: true } },
        { click: this.dlgButtonClick, buttonModel: { content: 'Cancel'
}}]
    }
  },
  mounted: function() {
    document.getElementById('dlgbtn').focus();
  },
  methods: {
    btnClick: function() {
      this.$refs.footerDialog.show();
    },
    dlgClose: function() {
      document.getElementById('dlgbtn').style.display = '';
    },
    dlgOpen: function() {
      document.getElementById('dlgbtn').style.display = 'none';
    },
    dlgButtonClick: function() {
      this.$refs.footerDialog.hide();
    }
  }
}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-vue-popups/styles/material.css";
#app {
  color: #008cff;
  height: 40px;
  left: 45%;
  position: absolute;
  top: 45%;
  width: 30%;
}
.control-section {
  height: 100%;
  min-height: 200px;
}
</style>

```

{% previewsample "page.domainurl/code-snippet/dialog/footer-cs1" %}



### Draggable

The Dialog supports to drag within its target container by grabbing the Dialog header, which allows the user to reposition the Dialog dynamically.

The Dialog can be draggable only when the Dialog header is enabled. From **16.2.x** version, enabled draggable support for modal dialog also.

### APP.VUE

```
<template>
  <div id="target" class="control-section">
    <ejs-dialog :header="header" :content="content"
:allowDragging="draggable" :target='target' :width='width'> </ejs-dialog>
  </div>
</template>
<script>
import Vue from 'vue';
import { DialogPlugin } from '@syncfusion/ej2-vue-popups';
Vue.use(DialogPlugin);
export default {
  data : function() {
    return {
      target: '#target',
      width: '300px',
      header: 'Dialog',
      draggable: true,
      content: 'This is a Dialog with drag enabled.'
    }
  }
}
</script>
<style>
@import "../../../node_modules/@syncfusion/ej2-vue-popups/styles/material.css";
#app {
  color: #008cff;
  height: 40px;
  left: 45%;
  position: absolute;
  top: 45%;
  width: 30%;
}
.control-section {
  min-height: 355px;
  margin: 10px;
}
</style>
```

```
{% previewsample "page.domainurl/code-snippet/dialog/draggable-cs1" %}
```

### Positioning

The Dialog can be positioned using the [position](#) property by providing the X and Y co-ordinates. It can be positioned inside the target of the container or **<body>** of the element based on the given X and Y values.

for X is: left, center, right (or) any offset value

for Y is: top, center, bottom (or) any offset value

The below example demonstrates the different Dialog positions.

### APP.VUE

```
<template>
  <div>
    <div id="target" class="control-section">
      <ejs-button id="dialogBtn" v-on:click.native="buttonClick">Open
Dialog</ejs-button>
      <ejs-dialog id='defaultDialog' header='Choose a Dialog Position'
showCloseIcon='true' :position='position' :footerTemplate='footerTemplate'
width='406px' ref='dialogObj'
      target='#target' :open='dialogOpen' :close='dialogClose'
closeOnEscape='false'>
        <table style='width:371px' id='poschange'>
          <tr>
            <td><ejs-radiobutton id='radio1' label='Left Top'
value='left top' name='xy' :change='changePosition' ></ejs-radiobutton></td>
            <td><ejs-radiobutton id='radio2' label='Center Top'
value='center top' name='xy' :change='changePosition'></ejs-
radiobutton></td>
            <td><ejs-radiobutton id='radio3' label='Right Top'
value='right top' name='xy' :change='changePosition'></ejs-radiobutton></td>
          </tr>
          <tr>
            <td><ejs-radiobutton id='radio4' label='Left Center'
value='left center' name='xy' :change='changePosition' ></ejs-
radiobutton></td>
            <td><ejs-radiobutton id='radio5' checked='true'
label='Center Center' value='center center' name='xy'
:change='changePosition' ></ejs-radiobutton></td>
            <td><ejs-radiobutton id='radio6' label='Right Center'
value='right center' name='xy' :change='changePosition' ></ejs-
radiobutton></td>
          </tr>
          <tr>
            <td><ejs-radiobutton id='radio7' label='Left Bottom'
value='left bottom' name='xy' :change='changePosition' ></ejs-
radiobutton></td>
            <td><ejs-radiobutton id='radio8' label='Center Bottom'
value='center bottom' name='xy' :change='changePosition' ></ejs-
radiobutton></td>
            <td><ejs-radiobutton id='radio9' label='Right Bottom'
value='right bottom' name='xy' :change='changePosition' ></ejs-
radiobutton></td>
          </tr>
        </table>
      </ejs-dialog>
    </div>
  </div>
</template>
<script>
import Vue from "vue";
import { DialogPlugin } from '@syncfusion/ej2-vue-popups';
import { ButtonPlugin } from '@syncfusion/ej2-vue-buttons';
```

```

import { RadioButtonPlugin } from '@syncfusion/ej2-vue-buttons';
Vue.use(DialogPlugin);
Vue.use(ButtonPlugin);
Vue.use(RadioButtonPlugin);
export default {
  data: function() {
    return {
      footerTemplate: '<span id="posvalue" style="float:left;margin-left:8px;padding:10px;">Position: { X: "Center", Y: "Center" }</span>',
      position: { X: 'center', Y: 'center' }
    },
    methods: {
      buttonClick: function(args){
        this.$refs.dialogObj.show();
      },
      changePosition: function(event) {
        this.$refs.dialogObj.position = { X: event.value.split(" ")[0], Y: event.value.split(" ")[1] };
        document.getElementById('posvalue').innerHTML = 'Position: {X: "' + event.value.split(" ")[0] + '", Y: "' + event.value.split(" ")[1] + '"}';
        var txt = event.event.target.parentElement.querySelector('.e-label').innerText.split(" ");
        document.getElementById('posvalue').innerHTML = 'Position: { X: "' + txt[0] + '", Y: "' + txt[1] + '" }';
      },
      dialogClose() {
        document.querySelector('#dialogBtn').style.display='inline-block';
      },
      dialogOpen() {
        document.querySelector('#dialogBtn').style.display='none';
      }
    }
  }
}
</script>
<style>
  html,
  body,
  #container {
    height: 100%;
    overflow: hidden;
    width: 100%;
  }
  .control-section {
    min-height: 350px;
    max-width: 840px;
    margin: 10px;
  }
  #defaultDialog table,
  #defaultDialog th,
  #defaultDialog td {
    border: 1px solid #D8D8D8;
    border-collapse: collapse;
  }
  #container {

```

```
        height: 365px;
    }
    #defaultDialog.e-dialog .e-footer-content {
        padding: 0px 1px 4px ! important;
    }
    .tableStyle {
        margin: 17px;
        width: 304px;
    }
    .e-dialog .e-dlgcontent{
        padding: 10px 16px 10px;
    }
    .e-radio +label .e-label {
        line-height:18px;
    }
    td {
        padding: 6px;
    }
}
</style>
```

{% previewsample "page.domainurl/code-snippet/dialog/position-cs1" %}

See Also

- [Load dialog content using AJAX](#)
- [How to position the dialog on center of the page on scrolling](#)
- [Prevent closing of modal dialog](#)
- [Close dialog while click on outside of dialog](#)
- [How to make a reusable alert and confirm dialog](#)

## Getting Started with the Vue Dialog Component in Vue 3

This article provides a step-by-step guide for setting up a [Vite](#) project with a JavaScript environment and integrating the Syncfusion Vue Dialog component using the [Composition API](#) / [Options API](#).

The **Composition API** is a new feature introduced in Vue.js 3 that provides an alternative way to organize and reuse component logic. It allows developers to write components as functions that use smaller, reusable functions called composition functions to manage their properties and behavior.

The **Options API** is the traditional way of writing Vue.js components, where the component logic is organized into a series of options that define the component's properties and behavior. These options include data, methods, computed properties, watchers, lifecycle hooks, and more.

Prerequisites

[System requirements for Syncfusion Vue UI components](#)

Set up the Vite project

A recommended approach for beginning with Vue is to scaffold a project using [Vite](#). To create a new Vite project, use one of the commands that are specific to either NPM or Yarn.

```
`bash
```

```
npm create vite@latest
```

```
`
```

or

```
`bash
```

```
yarn create vite
```

```
,
```

Using one of the above commands will lead you to set up additional configurations for the project as below:

1. Define the project name: We can specify the name of the project directly. Let's specify the name of the project as `my-project` for this article.

```
`bash
```

```
? Project name: » my-project
```

```
,
```

2. Select `Vue` as the framework. It will create a Vue 3 project.

```
`bash
```

```
? Select a framework: » - Use arrow-keys. Return to submit.
```

```
Vanilla
```

```
Vue
```

```
React
```

```
Preact
```

```
Lit
```

```
Svelte
```

```
Others
```

```
,
```

3. Choose `JavaScript` as the framework variant to build this Vite project using JavaScript and Vue.

```
`bash
```

```
? Select a variant: » - Use arrow-keys. Return to submit.
```

```
JavaScript
```

```
TypeScript
```

```
Customize with create-vue ↗
```

```
Nuxt ↗
```

```
,
```

4. Upon completing the aforementioned steps to create the `my-project`, run the following command to install its dependencies:

```
`bash
```

```
cd my-project
```

```
npm install
`
or
`bash
cd my-project
yarn install
`
```

Now that `my-project` is ready to run with default settings, let's add Syncfusion components to the project.

#### Add Syncfusion Vue packages

Syncfusion Vue component packages are available at [npmjs.com](https://www.npmjs.com). To use Syncfusion Vue components in the project, install the corresponding npm package.

This article uses the [Vue Dialog component](#) as an example. To use the Vue Dialog component in the project, the `@syncfusion/ej2-vue-popups` package needs to be installed using the following command:

```
`bash
npm install @syncfusion/ej2-vue-popups --save
`
or
`bash
yarn add @syncfusion/ej2-vue-popups
`
```

#### Import Syncfusion CSS styles

You can import themes for the Syncfusion Vue component in various ways, such as using CSS or SASS styles from npm packages, CDN, [CRG](#) and [Theme Studio](#). Refer to [themes topic](#) to know more about built-in themes and different ways to refer to themes in a Vue project.

In this article, `Material` theme is applied using CSS styles, which are available in installed packages. The necessary `Material` CSS styles for the Dialog component and its dependents were imported into the `<style>` section of `src/App.vue` file.

#### ~/SRC/APP.VUE

```
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-buttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-popups/styles/material.css";
</style>
```

The order of importing CSS styles should be in line with its dependency graph.

### Add Syncfusion Vue component

Follow the below steps to add the Vue Dialog component using **Composition API** or **Options API**:

1.First, import and register the Dialog component in the **script** section of the **src/App.vue** file. If you are using the **Composition API**, you should add the **setup** attribute to the **script** tag to indicate that Vue will be using the **Composition API**.

#### **COMPOSITION API (~SRC/APP.VUE)**

```
<script setup>
import { DialogComponent as EjsDialog } from "@syncfusion/ej2-vue-popups";
</script>
```

#### **OPTIONS API (~SRC/APP.VUE)**

```
<script>
import { DialogComponent } from "@syncfusion/ej2-vue-popups";
//Component registration
export default {
  name: "App",
  components: {
    "ejs-dialog": DialogComponent,
  }
}
</script>
```

2.In the **template** section, define the Dialog component with the [dataSource](#) property and column definitions.

#### **~/SRC/APP.VUE**

```
<template>
<div>
<div id="target" class="control-section">
<ejs-dialog :target="target" :width="width" :content="content">
</ejs-dialog>
</div>
</div>
</template>
```

Here is the summarized code for the above steps in the **src/App.vue** file:

#### **COMPOSITION API (~SRC/APP.VUE)**

```
<template>
<div>
<div id="target" class="control-section">
<ejs-dialog :target="data[0].target" :width="data[0].width"
:content="data[0].content">
</ejs-dialog>
</div>
</div>
</template>
<script setup>
import { DialogComponent as EjsDialog } from "@syncfusion/ej2-vue-popups";
```

```
const data = [{ target: "#target",
width: "335px",
content: "This is a Dialog with content."}],
</script>
<style>
@import "../node_modules/@syncfusion/ej2-vue-popups/styles/material.css";
.control-section {
height: 100%;
min-height: 200px;
}
</style>
```

### **OPTIONS API (~SRC/APP.VUE)**

```
<template>
<div>
<div id="target" class="control-section">
<ejs-dialog :target="target" :width="width" :content="content">
</ejs-dialog>
</div>
</div>
</template>
<script>
import { DialogComponent } from "@syncfusion/ej2-vue-popups";
export default {
name: "App",
components: {
"ejs-dialog": DialogComponent,
},
data: function () {
return {
target: "#target",
width: "335px",
content: "This is a Dialog with content.",
};
},
methods: {},
};
</script>
<style>
@import "../node_modules/@syncfusion/ej2-vue-popups/styles/material.css";
.control-section {
height: 100%;
min-height: 200px;
}
</style>
```

### **Run the project**

To run the project, use the following command:

```
`bash
```

```
npm run dev
```

```
`
```




or

```
`bash
```

```
yarn run dev
```

```
`
```

The output will appear as follows:



This is a Dialog with content

You can refer to our [Vue dialog](#) feature tour page for its groundbreaking feature representations. You can also explore our [Vue dialog example](#) that shows you how to render the dialog in Vue.

### Template in Vue Dialog component

In Dialog the template support is provided to the header, content and footer sections. So any text or HTML content can be appending in these sections.

#### Header

The Dialog header content can be provided through the [header](#) property, and it will allow both text and any HTML content as a string. Also in header, close button is provided as built-in support, and this can be enabled through the [showCloseIcon](#) property.

#### Footer

The Dialog footer can be enabled by adding built-in [buttons](#) or providing any HTML string through the [footerTemplate](#).

The [buttons](#) and [footerTemplate](#) properties can't be used at the same time.

#### Content

The Dialog content can be updated by providing any HTML string through the [content](#).

The below example demonstrates the usage of header, footer and content template in the Dialog.

#### **APP.VUE**

```
<template>
  <div>
    <div id="target" class="control-section; position:relative"
style="height:350px;">
      <!-- Render Button to open the Dialog -->
      <ejs-button id='modalBtn' v-on:click.native="btnClick">Open
Dialog</ejs-button>
```

```

        <!-- Render Dialog -->
        <ejs-dialog id='dialog' ref="templateDialog"
:header='headerTemplate' :target='target' :height='height' :width='width'
:footerTemplate='footerTemplate' :showCloseIcon='true'
:animationSettings='animationSettings' :content='contentTemplate'
:open="dlgOpen" :close="dlgClose">
        </ejs-dialog>
    </div>
</div>
</template>
<script>
import Vue from "vue";
import { DialogPlugin } from '@syncfusion/ej2-vue-popups';
import { ButtonPlugin } from '@syncfusion/ej2-vue-buttons';
import { isNullOrUndefined } from '@syncfusion/ej2-base';
Vue.use(DialogPlugin);
Vue.use(ButtonPlugin);
var footerTemplateVue = Vue.component("demo", {
    template: '<span> <input id="inVal" class="e-input" type="text"
placeholder="Enter your message here!"/><button id="sendButton" class="e-
control e-btn e-primary sendButton" data-ripple="true">Send</button>
</span>',
    data() {
        return {
            data: {}
        };
    }
});
var headerTemplateVue = Vue.component("demo1", {
    template: '<span> <div title="Nancy"
class="e-icon-settings dlg-template"> Nancy </div></span>',
    data() {
        return {
            data: {}
        };
    }
});
var contentTemplateVue = Vue.component("demo2", {
    template: '<div><div class="dialogContent"> <span
class="dialogText">Greetings Nancy! When will you share me the source files
of the project?</span></div></div>',
    data() {
        return {
            data: {}
        };
    }
});
export default {
    data: function() {
        return {
            target: '#target',
            height: '75%',
            width: '435px',
            footerTemplate: function () {
                return { template: footerTemplateVue }
            }
        };
    }
};

```

```

        },
        headerTemplate: function () {
            return { template: headerTemplateVue }
        },
        contentTemplate: function () {
            return { template: contentTemplateVue }
        },
        animationSettings: { effect: 'None' }
    }
},
mounted: function() {
    document.getElementById('modalBtn').focus();
},
methods: {
    btnClick: function() {
        this.$refs.templateDialog.show();
    },
    dlgClose: function() {
        document.getElementById('modalBtn').style.display = 'block';
    },
    dlgOpen: function() {
        document.getElementById('sendButton').keypress = (e) => {
            if (e.keyCode === 13) { this.updateTextValue(); }
        };
        document.getElementById('inVal').onkeydown = (e) => {
            if (e.keyCode === 13) { this.updateTextValue(); }
        };
        document.getElementById('sendButton').onclick = () => {
            this.updateTextValue();
        };
        document.getElementById('modalBtn').style.display = 'none';
    },
    updateTextValue: function() {
        var enteredVal = document.getElementById('inVal');
        var dialogTextElement =
document.getElementsByClassName('dialogText')[0];
        var dialogTextWrap =
document.getElementsByClassName('dialogContent')[0];
        if
(!isNullOrUndefined(document.getElementsByClassName('contentText')[0])) {
            detach(document.getElementsByClassName('contentText')[0]);
        }
        if (enteredVal.value !== '') {
            dialogTextElement.innerHTML = enteredVal.value;
        }
        enteredVal.value = '';
    },
    dlgButtonClick: function() {
        this.$refs.templateDialog.hide();
    }
}
}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-vue-popups/styles/material.css";
#app {
    color: #008cff;

```

```
        height: 40px;
        left: 45%;
        position: absolute;
        top: 45%;
        width: 30%;
    }
    .control-section {
        height: 100%;
        min-height: 200px;
    }
    html,
    body,
    #dialog-container {
        display: block;
        height: 100%;
        overflow: hidden;
        width: 100%;
    }
    #container {
        visibility: hidden;
    }
    .e-dialog .e-dlg-header-content {
        background-color: #007DD1;
    }
    #target {
        min-height: 350px;
    }
    .e-dialog .e-dlg-header-content .e-btn.e-dlg-closeicon-btn {
        top: 3px;
        left: -11px;
    }
    .e-dialog .e-dlg-header {
        position: relative;
        padding: 0;
    }
    .e-dialog .e-dlg-header-content {
        padding: 6px;
    }
    #dialog.e-dialog .e-dlg-header-content {
        padding: 13px;
    }
    .target-element .e-open-icon::before {
        content: '\e782';
    }
    .e-dialog .e-dlg-header img.img2 {
        height: 36px;
        width: 36px;
        border-radius: 50%;
        vertical-align: middle;
        display: inline-block;
    }
    .e-dialog .e-dlg-header .dlg-template {
        display: inline-block;
        padding: 0px 10px;
        vertical-align: middle;
        height: 30px;
        line-height: 30px;
    }
```

```

        color: #fff;
    }
    .e-dialog .e-footer-content input.e-input {
        width: 75%;
        float: left;
    }
    .e-icon-settings.e-icons {
        float: left;
        position: relative;
        left: 14%;
        top: -33px;
    }
    #dialog.e-dialog .e-footer-content {
        padding: 15px;
    }
    .dialogContent .dialogText {
        font-size: 13px;
        padding: 5%;
        word-wrap: break-word;
        border-radius: 6px;
        text-align: justify;
        font-style: initial;
        display: block;
    }
    .e-dialog .e-dlg-header .e-icon-settings, .target-element .e-icon-btn {
        color: #fff;
    }
    .dialogContent .dialogText {
        background-color: #f5f5f5;
    }
    #dialog.e-dialog .e-footer-content {
        border-top: 0.5px solid rgba(0, 0, 0, 0.42);
        padding-left: 35px;
    }
    .e-dialog .e-dlg-content .dialogContent {
        display: block;
        font-size: 15px;
        word-wrap: break-word;
        text-align: center;
        font-style: italic;
        border-radius: 6px;
        padding: 3%;
        position: relative;
        top: 6px;
    }
    .control-wrapper .e-control.e-dialog {
        width: 30%;
    }
    .e-dialog .e-dlg-header-content .e-icon-dlg-close {
        color: #fff;
    }
    .e-dialog .e-btn.e-dlg-closeicon-btn:hover span{
        color: #8ECBFF;
    }
    .e-dialog .e-dlg-header-content .e-btn.e-dlg-closeicon-btn:hover,
    .e-dialog .e-dlg-header-content .e-btn.e-dlg-closeicon-btn:focus {
        background-color: rgba(255, 255, 255, 0.10);
    }

```

```

    }
    .e-dialog .e-dlg-header-content .e-btn.e-dlg-closeicon-btn:active .e-
icon-dlg-close ,
    .e-dialog .e-dlg-header-content .e-btn.e-dlg-closeicon-btn:focus .e-
icon-dlg-close,
    .e-dialog .e-dlg-header-content .e-btn.e-dlg-closeicon-btn:hover .e-
icon-dlg-close {
        color: #fff;
    }
</style>

```

{% previewsample "page.domainurl/code-snippet/dialog/template-cs1" %}

See Also

- [How to add an icon to dialog buttons](#)
- [How to customize the dialog appearance](#)

### Animation in Vue Dialog component

The Dialog can be animated during the open and close actions. Also, user can customize animation's **delay**, **duration** and **effect**.

<!-- markdownlint-disable MD033 -->

delay	The Dialog animation will start with the mentioned delay
duration	Specifies the animation duration to complete with one animation cycle
effect	<p>Specifies the animation effects of Dialog open and close actions effect.</p> <p>List of supported animation effects:            'Fade'   'FadeZoom'   'FlipLeftDown'   'FlipLeftUp'   'FlipRightDown'   'FlipRightUp'              'FlipXDown'   'FlipXUp'   'FlipYLeft'   'FlipYRight'   'SlideBottom'   'SlideLeft'   'SlideRight'              'SlideTop'   'Zoom'   'None'</p> <p>If the user sets 'Fade' effect, then the Dialog will open with 'FadeIn' effect and close with 'FadeOut' effect</p>

In the below sample, **Zoom** effect is enabled. So, The Dialog will open with **ZoomIn** and close with **ZoomOut** effects.

#### APP.VUE

```

<template>
  <div>
    <div id="target" class="control-section">
      <div id='customization'>
        <div class='animate'>
          <ejs-button v-on:click.native='buttonClick'
id='Zoom'>Zoom</ejs-button>
        </div>
      </div>
    </div>
  </div>

```

```

        <div class='animate'>
            <ejs-button v-on:click.native='buttonClick'
id='FlipXDown'>FlipX Down</ejs-button>
        </div>
        <div class='animate'>
            <ejs-button v-on:click.native='buttonClick'
id='FlipXUp'>FlipX Up</ejs-button>
        </div>
        <div class='animate'>
            <ejs-button v-on:click.native='buttonClick'
id='FlipYLeft'>FlipY Left</ejs-button>
        </div>
        <div class='animate'>
            <ejs-button v-on:click.native='buttonClick'
id='FlipYRight'>FlipY Right</ejs-button>
        </div>
    </div>
    <ejs-dialog id='dialog' header='Animation Dialog' content='The
dialog is configured with animation effect. It is opened or closed with
"Zoom In or Out" animation.' showCloseIcon='true' :isModal='isModal'
:animationSettings='animationSettings' width='285px' ref='dialogObj'
target='#target' :buttons='dlgButton'>
    </ejs-dialog>
</div>
</div>
</template>
<script>
import Vue from "vue";
import { DialogPlugin } from '@syncfusion/ej2-vue-popups';
import { ButtonPlugin } from '@syncfusion/ej2-vue-buttons';
Vue.use(DialogPlugin);
Vue.use(ButtonPlugin);
export default {
    data: function() {
        return {
            dlgButton: [{ 'click': () => { this.$refs.dialogObj.hide(); } },
buttonModel: { content: 'OK', isPrimary: true }},
            isModal: true,
            animationSettings: { effect: 'Zoom' }
        }
    },
    methods: {
        buttonClick: function(e){
            var effect = e.target.getAttribute('id');
            var txt = e.target.parentElement.innerText;
            txt = (txt === 'Zoom In/Out') ? 'Zoom In or Out' : txt;
            this.$refs.dialogObj.content = 'The dialog is configured with
animation effect. It is opened or closed with "' + txt + '" animation.';
            this.$refs.dialogObj.animationSettings = { effect: effect,
duration: 400};
            setTimeout(() => {
                this.$refs.dialogObj.show();
            }, 400);
        }
    }
}
</script>

```

```

<style>
  @import "../node_modules/@syncfusion/ej2-vue-
  popups/styles/material.css";
  #customization {
    display: table;
    margin: 0 auto;
  }
  #app {
    color: #008cff;
    height: 40px;
    left: 45%;
    position: absolute;
    top: 45%;
    width: 30%;
  }
  .control-section {
    height: 100%;
    min-height: 340px;
  }
  .animate {
    display: table-cell;
    padding-left: 20px;
  }
  #customization .e-btn.e-outline:hover,
  #customization .e-css.e-btn.e-outline:hover,
  #customization .e-btn.e-outline:hover:focus,
  #customization .e-css.e-btn.e-outline:hover:focus,
  #customization .e-btn.e-outline:focus,
  #customization .e-css.e-btn.e-outline:focus {
    background-color: #ffffff;
    border-color: #0078d6;
    color: #0078d6;
    outline: none;
  }
</style>

```

```
{% previewsample "page.domainurl/code-snippet/dialog/animation-cs1" %}
```

## Resize in Vue Dialog component

The Dialog supports resizing feature. To resize the dialog, we have to select and resize it by using its handle (grip) or hovering on any of the edges or borders of the dialog within the sample container.

The resizable dialog can be created by setting the [enableResize](#) property to true, which is used to change the size of a dialog dynamically and view its content with expanded mode. The [resizeHandles](#) property can also be configured for all the which directions in which the dialog should be resized. When you configure the target property along with the [enableResize](#) property, the dialog can be resized within its specified target container.

### APP.VUE

```

<template>
  <div id="target" class="control-section">
    <ejs-dialog :header="header" :content="content" :enableResize='true'
    :resizeHandles='resizeHandles' :allowDragging="draggable" :target='target'
    :width='width'> </ejs-dialog>
  </div>

```



```

</template>
<script>
import Vue from 'vue';
import { DialogPlugin } from '@syncfusion/ej2-vue-popups';
Vue.use(DialogPlugin);
export default {
  data : function() {
    return {
      target: '#target',
      width: '300px',
      header: 'Dialog',
      resizeHandles: ['All'],
      draggable: true,
      content: 'This is a Dialog with resize enabled.'
    }
  }
}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-vue-popups/styles/material.css";
#app {
  color: #008cff;
  height: 40px;
  left: 45%;
  position: absolute;
  top: 45%;
  width: 30%;
}
.control-section {
  min-height: 355px;
  margin: 10px;
}
</style>

```

{% previewsample "page.domainurl/code-snippet/dialog/resize-cs1" %}

### Dialog utility in Vue Dialog component

The dialog component provides built-in utility functions to render the alert and confirm dialogs with the minimal code.

The following options are used as an argument on calling the utility functions:

Options	Description
title	Specifies the title of dialog like the <a href="#">header</a> property.
content	Specifies the value that can be displayed in dialog's content area like the <a href="#">content</a> property.
isModal	Specifies the Boolean value whether the dialog can be displayed as modal or non-modal. For more details, refer to the <a href="#">isModal</a> property.
position	Specifies the value where the alert or confirm dialog is positioned within the document. For more details, refer to the <a href="#">position</a> property { X: 'center', Y: 'center'}

| okButton | Configures the OK button that contains button properties with the click events.

okButton:{ icon:'prefix icon to the button', cssClass:'custom class to the button', click: 'action for OK button click', text: 'Yes' // <-- Default value is 'OK' }

| cancelButton | Configures the Cancel button that contains button properties with the click events.

cancelButton:{ icon:'prefix icon to the button', cssClass:'custom class to the button', click: 'action for 'Cancel' button click', text: 'No' // <-- Default value is 'Cancel' }

| isDraggable | Specifies the value whether the alert or confirm dialog can be dragged by the user. |

| showCloseIcon | When set to true, the close icon is shown in the dialog component. |

| closeOnEscape | When set to true, you can close the dialog by pressing ESC key. |

| animationSettings | Specifies the animation settings of the dialog component. |

| cssClass | Specifies the CSS class name that can be appended to the dialog. |

| zIndex | Specifies the order of the dialog, that is displayed in front or behind of another component. |

| open | Event which is triggered after the dialog is opened. |

| close | Event which is triggered after the dialog is closed. |

### Alert dialog

An alert dialog box is used to display warning like messages to the users. Use the following code to render a simple alert dialog in an application.

#### APP.VUE

```
<template>
  <div>
    <center><ejs-button ref='button' id="dialogbtn" cssClass="e-info" v-
on:click.native="dialogBtnClick">Open Alert Dialog</ejs-button></center>
  </div>
</template>
<script>
import Vue from 'vue';
import { DialogUtility } from '@syncfusion/ej2-popups';
import { ButtonPlugin } from '@syncfusion/ej2-vue-buttons';
Vue.use(ButtonPlugin);
Vue.use(DialogUtility);
export default {
  data: function() {
    return { }
  },
  methods: {
    dialogBtnClick: function() {
      DialogUtility.alert('This is an Alert Dialog!');
    }
  }
}
</script>
<style>
@import "../..../node_modules/@syncfusion/ej2-vue-
popups/styles/material.css";
#app {
  color: #008cff;
}
```

```

        height: 40px;
        left: 45%;
        position: absolute;
        top: 45%;
        width: 30%;
    }
</style>

```

{% previewsample "page.domainurl/code-snippet/dialog/dialog-utility-alert-cs2" %}

*Render an alert dialog with options*

### **APP.VUE**

```

<template>
  <div>
    <center><ejs-button ref='button' id="dialogbtn" cssClass="e-info" v-
on:click.native="dialogBtnClick">Open Alert Dialog</ejs-button></center>
  </div>
</template>
<script>
import Vue from 'vue';
import { DialogUtility } from '@syncfusion/ej2-popups';
import { ButtonPlugin } from '@syncfusion/ej2-vue-buttons';
Vue.use(ButtonPlugin);
Vue.use(DialogUtility);
export default {
  data: function() {
    return { }
  },
  methods: {
    dialogBtnClick: function() {
      DialogUtility.alert({
        title: 'Alert Dialog',
        content: "This is an Alert Dialog!",
        okButton: { text: 'OK', click: this.okClick },
        showCloseIcon: true,
        closeOnEscape: true,
        animationSettings: { effect: 'Zoom' }
      });
    },
    okClick: function() {
      alert('you clicked OK button');
    }
  }
}
</script>
<style>
  @import "../node_modules/@syncfusion/ej2-vue-
popups/styles/material.css";
  #app {
    color: #008cff;
    height: 40px;
    left: 45%;
    position: absolute;
    top: 45%;
    width: 30%;
  }

```

```
}
</style>
```

{% previewsample "page.domainurl/code-snippet/dialog/dialog-utility-alert1-cs2" %}

### Confirm dialog

A confirm dialog displays a specified message along with 'OK' and 'Cancel' button.

#### APP.VUE

```
<template>
  <div>
    <center><ejs-button ref='button' id="dialogbtn" cssClass="e-info" v-
on:click.native="dialogBtnClick">Open Confirm Dialog</ejs-button></center>
  </div>
</template>
<script>
import Vue from 'vue';
import { DialogUtility } from '@syncfusion/ej2-popups';
import { ButtonPlugin } from '@syncfusion/ej2-vue-buttons';
Vue.use(ButtonPlugin);
Vue.use(DialogUtility);
export default {
  data: function() {
    return { }
  },
  methods: {
    dialogBtnClick: function() {
      DialogUtility.confirm('This is a Confirmation Dialog!');
    }
  }
}
</script>
<style>
@import "../../node_modules/@syncfusion/ej2-vue-
popups/styles/material.css";
#app {
  color: #008cff;
  height: 40px;
  left: 45%;
  position: absolute;
  top: 45%;
  width: 30%;
}
</style>
```

{% previewsample "page.domainurl/code-snippet/dialog/dialog-utility-confirm-cs2" %}

### Render a confirmation dialog with options

#### APP.VUE

```
<template>
  <div>
    <center><ejs-button ref='button' id="dialogbtn" cssClass="e-info" v-
on:click.native="dialogBtnClick">Open Confirm Dialog</ejs-button></center>
  </div>
```

```

</template>
<script>
import Vue from 'vue';
import { DialogUtility } from '@syncfusion/ej2-popups';
import { ButtonPlugin } from '@syncfusion/ej2-vue-buttons';
Vue.use(ButtonPlugin);
Vue.use(DialogUtility);
export default {
  data: function() {
    return { }
  },
  methods: {
    dialogBtnClick: function() {
      // Initialize and render Confirm dialog with options
      DialogUtility.confirm({
        title: ' Confirmation Dialog',
        content: "This is a Confirmation Dialog!",
        okButton: { text: 'OK', click: this.okClick },
        cancelButton: { text: 'Cancel', click: this.cancelClick },
        showCloseIcon: true,
        closeOnEscape: true,
        animationSettings: { effect: 'Zoom' }
      });
    },
    okClick: function() {
      alert('you clicked OK button');
    },
    cancelClick: function() {
      alert('you clicked Cancel button');
    }
  }
}
</script>
<style>
  @import "../../node_modules/@syncfusion/ej2-vue-popups/styles/material.css";
  #app {
    color: #008cff;
    height: 40px;
    left: 45%;
    position: absolute;
    top: 45%;
    width: 30%;
  }
</style>

```

{% previewsample "page.domainurl/code-snippet/dialog/dialog-utility-confirm1-cs2" %}

### Close utility dialog

When rendering an Alert and Confirmation dialog through utility methods, You can close the dialog using the following ways.

- By pressing the escape key if the [closeOnEscape](#) property is enabled.
- By clicking the close button if the [showCloseIcon](#) property is enabled.

You can also manually close the Dialogs by creating an instance to the dialog and call the [hide](#) method.

Below sample demonstrates the different ways of hiding the utility dialog.

#### APP.VUE

```
<template>
  <div>
    <center><ejs-button ref='button' id="dialogbtn" cssClass="e-info" v-
on:click.native="dialogBtnClick">Open Confirm Dialog</ejs-button></center>
  </div>
</template>
<script>
import Vue from 'vue';
import { DialogUtility } from '@syncfusion/ej2-popups';
import { ButtonPlugin } from '@syncfusion/ej2-vue-buttons';
Vue.use(ButtonPlugin);
Vue.use(DialogUtility);
let DialogObj = undefined;
export default {
  data: function() {
    return { }
  },
  methods: {
    dialogBtnClick: function() {
      // Initialize and render Confirm dialog with options
      DialogObj = DialogUtility.confirm({
        title: 'Confirmation Dialog',
        content: "This is a Confirmation Dialog!",
        okButton: { text: 'OK', click: this.okClick },
        cancelButton: { text: 'Cancel', click: this.cancelClick },
        showCloseIcon: true,
        closeOnEscape: true,
        animationSettings: { effect: 'Zoom' }
      });
    },
    okClick: function() {
      alert('you clicked OK button');
    },
    cancelClick: function() {
      //Hide the dialog
      DialogObj.hide();
    }
  }
}
</script>
<style>
  @import "../../node_modules/@syncfusion/ej2-vue-
popups/styles/material.css";
  #app {
    color: #008cff;
    height: 40px;
    left: 45%;
    position: absolute;
    top: 45%;
    width: 30%;
  }
</style>
```

```
</style>
```

```
{% previewsample "page.domainurl/code-snippet/dialog/dialog-utility-confirm1-cs3" %}
```

### Style in Vue Dialog component

The following content provides the exact CSS structure that can be used to modify the control's appearance based on the user preference.

#### Customizing the dialog header

Use the following CSS to customize the dialog header properties.

```
`css
.e-dialog .e-dlg-header {
color: green;
font-size: 20px;
font-weight: normal;
}
`
```

#### Customizing the dialog content

Use the following CSS to customize the dialog content properties.

```
`css
.e-dialog .e-dlg-content {
color: red;
font-size: 10px;
font-weight: normal;
line-height: normal;
}
`
```

#### Customizing modal dialog overlay

Use the following CSS to customize the modal dialog overlay.

```
`css
.e-dlg-overlay {
background-color: slategray;
opacity: 0.6;
}
`
```

#### Customizing the dialog resize icon

Use the following CSS to customize the dialog resize icon.

```
`css
/ To change the icon content /
.e-dialog .e-south-east::before, .e-dialog .e-south-west::before {
content: '\f047';
}
/ To set the icon pack /
.e-dialog .e-resize-handle {
font: normal normal normal 14px/1 FontAwesome;
}
`
```

The above CSS demonstration uses the font awesome icon.

#### Customizing the dialog close button

Use the following CSS to customize the dialog close button.

```
`css
/ To specify font size and color /
.e-dialog .e-btn .e-btn-icon.e-icon-dlg-close {
font-size: 12px;
color: red;
}
`
```

#### Customizing the dialog footer button

Use the following CSS to customize the dialog footer button.

```
`css
/ To specify font color, background color and border color /
.e-btn.e-flat.e-primary, .e-css.e-btn.e-flat.e-primary {
background-color: transparent;
border-color: transparent;
color: blue;
}
`
```

### Localization in Vue Dialog component

**Localization** library allows to localize the default text content of Dialog. In Dialog, The close button's tooltip text alone will be localize based on culture.

| Locale key | en-US (default) |



|-----|-----|

| close | Close |

### Loading translations

To load translation object in an application use `load` function of `L10n` class.

In the below sample, `French` culture is set to Dialog and change the close button's tooltip text.

### APP.VUE

```
<template>
  <div>
    <div id="target" class="control-section; position:relative"
style="height:350px;">
      <!-- Render Button to open the Dialog -->
      <ejs-button id='modalbtn' v-on:click.native="btnClick">Open</ejs-
button>
      <!-- Render Dialog -->
      <ejs-dialog ref="localeDialog" :header='header' :target='target'
:width='width' :locale='locale' :showCloseIcon='true'
:animationSettings='animationSettings' :content='content' :open="dlgOpen"
:close="dlgClose">
        </ejs-dialog>
      </div>
    </div>
  </div>
</template>
<script>
import Vue from "vue";
import { DialogPlugin } from '@syncfusion/ej2-vue-popups';
import { ButtonPlugin } from '@syncfusion/ej2-vue-buttons';
import { L10n } from '@syncfusion/ej2-base';
Vue.use(DialogPlugin);
Vue.use(ButtonPlugin);
L10n.load({
  'fr-BE': {
    'dialog': {
      'close': "Fermer"
    }
  }
});
export default {
  data: function() {
    return {
      target: '#target',
      width: '335px',
      header: 'Dialogue',
      content: 'Dialogue avec la culture française.',
      animationSettings: { effect: 'None' },
      locale: 'fr-BE'
    }
  },
  mounted: function(){
    document.getElementById('modalbtn').focus();
  },
  methods: {
    btnClick: function() {
      this.$refs.localeDialog.show();
    }
  }
}
```

```

    },
    dlgClose: function() {
      document.getElementById('modalbtn').style.display = '';
    },
    dlgOpen: function() {
      document.getElementById('modalbtn').style.display = 'none';
    }
  }
}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-vue-popups/styles/material.css";
#app {
  color: #008cff;
  height: 40px;
  left: 45%;
  position: absolute;
  top: 45%;
  width: 30%;
}
.control-section {
  height: 100%;
  min-height: 200px;
}
</style>

```

{% previewsample "page.domainurl/code-snippet/dialog/locale-cs1" %}

### Accessibility in Vue Dialog component

The Dialog component followed the accessibility guidelines and standards, including [ADA](#), [Section 508](#), [WCAG 2.2](#) standards, and [WCAG roles](#) that are commonly used to evaluate accessibility.

The accessibility compliance for the Dialog component is outlined below.

| Accessibility Criteria | Compatibility |

| -- | -- |

| [WCAG 2.2 Support](#) |  |

| [Section 508 Support](#) |  |

| [Screen Reader Support](#) |  |

| [Right-To-Left Support](#) |  |

| [Color Contrast](#) |  |

| [Mobile Device Support](#) |  |

```
| Keyboard Navigation Support |  |  
| Accessibility Checker Validation |  |  
| Axe-core Accessibility Validation |  |  
<style>  
.post .post-content img {  
display: inline-block;  
margin: 0.5em 0;  
}  
</style>  
<div> - All  
features of the component meet the requirement.</div>  
<div> - Some features of the component do not meet the requirement.</div>  
<div> - The  
component does not meet the requirement.</div>
```

### WAI-ARIA attributes

The Dialog characterized with complete ARIA Accessibility support which helps to accessible by on-screen readers and other assistive technology devices. This component designed with the reference of the guidelines document given in [WAI ARAI Accessibility Practices](#).

The Dialog component uses the **Dialog** role and following ARIA properties to its element based on its state.

### | Property | Functionalities |

| --- | --- |

| aria-describedby | It indicates the Dialog content description is notified to the user through assistive technologies. |

| aria-labelledby | The Dialog title is notified to the user through assistive technologies. |

| aria-modal | For modal dialog it's value is true and non-modal dialog its value is false |

| aria-grabbed | Enable the draggable Dialog and starts dragging it is value is true and stopping the drag its value is false |

### Keyboard interaction

Keyboard interaction of Dialog component has designed based on [WAI-ARIA Practices](#) described for Dialog. User can use the following shortcut keys to interact with the Dialog.

<!-- markdownlint-disable MD033 -->

Keyboard shortcuts	Actions
--------------------	---------

Esc	Closes the Dialog. This functionality can be controlled by using <a href="#">closeOnEscape</a>
Enter	When the Dialog button or any input (except text area) is in focus state, when pressing the Enter key, the click event associated with the primary button or button will trigger. Enter key is not working when the Dialog content contains any text area with initial focus
Ctrl + Enter	When the Dialog content contains text area and it is in focus state, and press the Ctrl + Enter key to call the click event function associated with primary button
Tab	Focus will be changed within the Dialog elements
Shift + Tab	The Focus will be changed previous focusable element within the Dialog elements. When focusing a first focusable element in the Dialog, then press the shift + tab key, it will change the focus to last focusable element

**APP.VUE**

```

<template>
  <div>
    <div id="target" class="control-section; position:relative"
    style="height:350px;">
      <ejs-button id='modalbtn' v-on:click.native="btnClick">Open</ejs-
      button>
      <ejs-dialog ref="accessDialog" :header='header' :target='target'
      :width='width' :height='height' :showCloseIcon='true' :buttons='buttons'
      :content='content' :open="dlgOpen" :close="dlgClose">
        </ejs-dialog>
      </div>
    </div>
  </template>
<script>
import Vue from "vue";
import { DialogPlugin } from '@syncfusion/ej2-vue-popups';
import { ButtonPlugin } from '@syncfusion/ej2-vue-buttons';
Vue.use(DialogPlugin);
Vue.use(ButtonPlugin);
export default {
  data: function() {
    return {
      target: '#target',
      width: '335px',
      height: '350px',
      header: 'Feedback',
      content: "<form><div class='form-group'><label
for='email'>Email:</label>" +
        "<input type='email' class='form-control' id='email'>" +
        "</div><div class='form-group'>" +
        "</div><div class='form-group'><label
for='comment'>Comments:</label>" +
        "<textarea style='resize: none;' class='form-control'
rows='4' id='comment'></textarea>" +
        "</div>" +
        "</form>",
    }
  }
}

```

```

        buttons: [{ click: this.dlgButtonClick, buttonModel: { content:
'OK', isPrimary: true } },
        { click: this.dlgButtonClick, buttonModel: { content: 'Cancel'
}}]
    },
    },
    mounted: function() {
        document.getElementById('modalbtn').focus();
    },
    methods: {
        btnClick: function() {
            this.$refs.accessDialog.show();
        },
        dlgClose: function() {
            document.getElementById('modalbtn').style.display = '';
        },
        dlgOpen: function() {
            document.getElementById('modalbtn').style.display = 'none';
        },
        dlgButtonClick: function() {
            this.$refs.accessDialog.hide();
        }
    }
}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-vue-popups/styles/material.css";
@import
"https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css";
#app {
    color: #008cff;
    height: 40px;
    left: 45%;
    position: absolute;
    top: 45%;
    width: 30%;
}
.control-section {
    height: 100%;
    min-height: 200px;
}
</style>

```

{% previewsample "page.domainurl/code-snippet/dialog/accessibility-cs1" %}

See Also

- [Show dialog with full-screen](#)

### Ensuring accessibility

The Dialog component's accessibility levels are ensured through an [accessibility-checker](#) and [axe-core](#) software tools during automated testing.

The accessibility compliance of the Dialog component is shown in the following sample. Open the [sample](#) in a new window to evaluate the accessibility of the Dialog component with accessibility tools.

{% previewsample "https://ej2.syncfusion.com/accessibility/dialog.html" %}

See also

- [Accessibility in Syncfusion React components](#)

## How To

Create nested dialog in Vue Dialog component

A Dialog can be nested within another Dialog. The below sample contains parent and child Dialog (inner Dialog).

### Step 1:

Create two div elements with id `#dialog` and `#innerDialog`.

### Step 2:

Initialize the Dialog as mentioned in the below sample.

### Step 3:

Set the inner Dialog target as `#dialog`.

### APP.VUE

```
<template>
  <div>
    <div id="target" class="col-lg-12 control-section"
      style="padding:10px;position:relative;">
      <ejs-button id='dlgbtn' v-on:click.native="buttonClick">Open
      Dialog</ejs-button>
      <ejs-dialog ref='dialogObj' :visible="true" header='First Dialog'
      showCloseIcon='true' :animationSettings='animationSettings' width='330px'
      height='260px'
        target='#target' content='<p>This is the first dialog and acts
as a parent dialog, you can open the second (child) dialog by clicking
"Next".</p>' :buttons='dlgButton' :open='dialogOpen' :close='dialogClose'>
        </ejs-dialog>
      <ejs-dialog ref='secondDialog' :visible="false" header='Second
      Dialog' :isModal='isModal' showCloseIcon='true' allowDragging='true'
      :animationSettings='animationSettings' width='285px' height='215px'
        target='#target' content='<p>This is the second dialog and act
as a child dialog.</p>' :buttons='dlg2Button'>
        </ejs-dialog>
      </div>
    </div>
  </template>
<style>
  #dlgbtn {
    margin-right: 5%;
  }
  .control-section {
    height: 100%;
    min-height: 340px;
  }
  #app {
    color: #008cff;
  }
</style>
```

```

        height: 40px;
        left: 45%;
        position: absolute;
        top: 45%;
        width: 30%;
    }
</style>
<script>
import Vue from "vue";
import { DialogPlugin } from '@syncfusion/ej2-vue-popups';
import { ButtonPlugin } from '@syncfusion/ej2-vue-buttons';
Vue.use(DialogPlugin);
Vue.use(ButtonPlugin);
export default {
    data: function() {
        return {
            dlgButton: [{ 'click': () => { this.$refs.secondDialog.show(); },
buttonModel: { isPrimary: 'true', content: 'Next' } }],
            dlg2Button: [{ 'click': () => { this.$refs.secondDialog.hide();
},buttonModel: { isPrimary: 'true', content: 'Close' } }],
            animationSettings: { effect: 'None' },
            isModal: true,
        }
    },
    methods: {
        buttonClick: function(args){
            this.$refs.dialogObj.show();
        },
        dialogClose: function() {
            document.querySelector('#dlgbtn').style.display='block';
        },
        dialogOpen: function() {
            document.querySelector('#dlgbtn').style.display='none';
        }
    }
};
</script>

```

{% previewsample "page.domainurl/code-snippet/dialog/inner-dialog-cs1" %}

### Position the dialog on center of the page on scrolling in Vue Dialog component

By default, when scroll the page/container Dialog also scrolled along with the page/container. When a user expects to display the Dialog in the same position without scrolling achieving this in sample level as like below. Here added **e-fixed** class to Dialog element and prevent the scrolling.

#### APP.VUE

```

<template>
  <div>
    <div id="target" class="control-section; position:relative">
      <div>
        <b>JavaScript:</b><br />
        JavaScript is a high-level, dynamic, untyped, and interpreted
        programming language. It has been standardized in the ECMAScript
        language specification. Alongside HTML and CSS, it is one of the
        three essential technologies of World Wide Web
      </div>
    </div>
  </div>
</template>

```

content production; the majority of websites employ it and it **is** supported by all modern Web browsers without plug-ins. JavaScript **is** prototype-based with first-class **functions**, making it a multi-paradigm language, supporting **object** - oriented , imperative, and functional programming styles.

```
<br /><br /><br />
```

```
<b>MVC:</b><br />
```

Model-view-controller (MVC) **is** a software architecture pattern which separates the representation of information **from** the user's interaction with it. The model consists of application data, business rules, logic, and functions. A view can be any output representation of data, such **as** a chart or a diagram. Multiple views of the same data are possible, such **as** a bar chart **for** management and a tabular view **for** accountants. The controller mediates input, converting it to commands **for** the model or view. The central ideas behind MVC are code reusability and **in** addition to dividing the application **into** three kinds of components, the MVC design defines the interactions between them.

A controller can send commands to its associated view to change the view's presentation of the model (e.g., by scrolling through a document). It can also send commands to the model to update the model's state (e.g., editing a document).

A model notifies its associated views and controllers when there has been a change **in** its state. This notification allows the views to produce updated output, and the controllers to change the available **set** of commands. A passive implementation of MVC omits these notifications, because the application does not require them or the software platform does not support them.

A view requests **from** the model the information that it needs to generate an output representation to the user.

```
</div>
<!-- Render Dialog -->
<ejs-dialog ref="scrollDialog" :header='header' :target='target'
:width='width' :showCloseIcon='true' :content='content' :open="dlgOpen"
:close="dlgClose">
  </ejs-dialog>
</div>
</div>
</template>
<script>
import Vue from "vue";
import { DialogPlugin } from '@syncfusion/ej2-vue-popups';
import { ButtonPlugin } from '@syncfusion/ej2-vue-buttons';
Vue.use(DialogPlugin);
Vue.use(ButtonPlugin);
export default {
  data: function() {
    return {
      target: '#target',
      width: '335px',
      header: 'Dialog',
      content: '<button class="e-control e-btn" id="targetButton"
role="button" >Prevent Dialog Scroll</button>',
    }
  },
  mounted: function(){
    document.getElementById('targetButton').onclick = (): void => {
```



```

        this.$refs.scrollDialog.cssClass = 'e-fixed';
    },
    methods: {
        btnClick: function() {
            this.$refs.scrollDialog.show();
        },
        dlgClose: function() {
            document.getElementById('modalbtn').style.display = '';
        },
        dlgOpen: function() {
            document.getElementById('modalbtn').style.display = 'none';
        }
    }
}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-vue-popups/styles/material.css";
#app {
    color: #008cff;
    height: 40px;
    left: 45%;
    position: absolute;
    top: 45%;
    width: 30%;
}
html,
body,
.control-section {
    height: 120%;
    min-height: 200px;
}
body {
    overflow-y: scroll;
}
#loader {
    color: #008cff;
    height: 40px;
    left: 45%;
    position: absolute;
    top: 45%;
    width: 30%;
}
.e-fixed {
    position: fixed;
}
</style>

```

{% previewsample "page.domainurl/code-snippet/dialog/scroll-cs1" %}

### Load dialog content using ajax in Vue Dialog component

You can load dialog's content dynamically from external source like external file using AJAX library. The AJAX library can make the request and load dialog's content using its `success` event. Refer the following link to learn about how to load dialog content using AJAX.

#### [AJAX Content](#)

### Render a dialog without header in Vue Dialog component

The dialog can be rendered without header by setting the [header](#) property value as empty string or null. By default, dialog is rendered without header.

#### APP.VUE

```
<template>
  <div>
    <div id="target" class="control-section; position:relative"
    style="height:350px;">
      <!-- Render Button to open the Dialog -->
      <ejs-button id='dlgbtn' v-on:click.native="btnClick">Open
Dialog</ejs-button>
      <!-- Render Dialog -->
      <ejs-dialog ref="footerDialog" :target='target' :width='width'
:buttons='buttons' :content='content' :close="dlgClose">
      </ejs-dialog>
    </div>
  </div>
</template>
<script>
import Vue from 'vue';
import { DialogPlugin } from '@syncfusion/ej2-vue-popups';
import { ButtonPlugin } from '@syncfusion/ej2-vue-buttons';
Vue.use(DialogPlugin);
Vue.use(ButtonPlugin);
export default {
  data: function() {
    return {
      target: "#target",
      width: '250px',
      content: 'This is a dialog without header.',
      buttons: [{ click: this.dlgButtonClick, buttonModel: { content:
'OK', isPrimary: true } },
        { click: this.dlgButtonClick, buttonModel: { content: 'Cancel'
}}],
      animationSettings: { effect: 'None' }
    }
  },
  mounted: function(){
    document.getElementById('dlgbtn').focus();
  },
  methods: {
    btnClick: function() {
      this.$refs.footerDialog.show();
    },
    dlgClose: function() {
      document.getElementById('dlgbtn').style.display = '';
    },
    dlgButtonClick: function() {
      this.$refs.footerDialog.hide();
    }
  }
}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-vue-popups/styles/material.css";
```

```
#app {
  color: #008cff;
  height: 40px;
  left: 45%;
  position: absolute;
  top: 45%;
  width: 30%;
}
.control-section {
  height: 100%;
  min-height: 200px;
}
</style>
```

{% previewsample "page.domainurl/code-snippet/dialog/dlg-header-cs1" %}

Show dialog with full screen in Vue Dialog component

You can show the dialog in fullscreen by passing `true` as argument to the dialog [show](#) method.

### APP.VUE

```
<template>
  <div>
    <div id="target" class="control-section; position:relative"
style="height:360px;">
      <!-- Render Button to open the Dialog -->
      <ejs-button id='dlgbtn' v-on:click.native="btnClick">Open
Dialog</ejs-button>
      <!-- Render Dialog -->
      <ejs-dialog ref="footerDialog" :header='header' :target='target'
:width='width' :buttons='buttons' :visible='visible'
:showCloseIcon='showCloseIcon' :content='content' :close="dlgClose">
        </ejs-dialog>
    </div>
  </div>
</template>
<script>
import Vue from 'vue';
import { DialogPlugin } from '@syncfusion/ej2-vue-popups';
import { ButtonPlugin } from '@syncfusion/ej2-vue-buttons';
Vue.use(DialogPlugin);
Vue.use(ButtonPlugin);
export default {
  data: function() {
    return {
      target: "#target",
      width: '250px',
      header: 'Dialog',
      visible: false,
      showCloseIcon: true,
      content: 'This is a Dialog with fullscreen display.',
      buttons: [{ click: this.dlgButtonClick, buttonModel: { content:
'OK', isPrimary: true } },
        { click: this.dlgButtonClick, buttonModel: { content: 'Cancel'
} } ],
      animationSettings: { effect: 'None' }
    }
  }
}
```

```

    }
  },
  mounted: function() {
    document.getElementById('dlgbtn').focus();
  },
  methods: {
    btnClick: function() {
      this.$refs.footerDialog.show(true);
    },
    dlgClose: function() {
      document.getElementById('dlgbtn').style.display = '';
    },
    dlgButtonClick: function() {
      this.$refs.footerDialog.hide();
    }
  }
}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-vue-popups/styles/material.css";
#app {
  color: #008cff;
  height: 40px;
  left: 45%;
  position: absolute;
  top: 45%;
  width: 30%;
}
.control-section {
  height: 100%;
  max-height: 380px;
  overflow: hidden;
}
</style>

```

{% previewsample "page.domainurl/code-snippet/dialog/dlg-fullscreen-cs1" %}

### Display a dialog with custom position in Vue Dialog component

By default, the dialog is displayed in the center of the target container. The dialog position can be set using the [position](#) property by providing custom X and Y coordinates.

The dialog can be positioned inside the target based on the given X and Y values.

#### APP.VUE

```

<template>
  <div>
    <div id="target" class="control-section">
      <ejs-dialog id='firstDialog' header='Position-01'
        :position='position' width='360px' ref='dialogObj'
        target='#target' :content='content' :closeOnEscape='false'>
      </ejs-dialog>
      <ejs-dialog ref="secondDialog" header='Position-02'
        :height='height1' :target='target' width='360px' :position='position1'
        :content='content1'>
      </ejs-dialog>
    </div>
  </div>
</template>

```

```

    </div>
  </div>
</template>
<script>
import Vue from 'vue';
import { DialogPlugin } from '@syncfusion/ej2-vue-popups';
import { ButtonPlugin } from '@syncfusion/ej2-vue-buttons';
Vue.use(DialogPlugin);
Vue.use(ButtonPlugin);
export default {
  data: function() {
    return {
      target: "#target",
      height: '120px',
      content: 'The dialog is positioned at {X: 160, Y: 14}
coordinates.',
      position: { X: 420, Y: 14 },
      height1: '120px',
      content1: 'The dialog is positioned at {X: 160, Y: 240}
coordinates',
      position1: { X: 420, Y: 240 }
    }
  },
}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-vue-popups/styles/material.css";
#app {
  color: #008cff;
  height: 40px;
  left: 45%;
  position: absolute;
  top: 45%;
  width: 30%;
}
.control-section {
  height: 100%;
  min-height: 200px;
}
#defaultDialog table,
#defaultDialog th,
#defaultDialog td {
  border: 1px solid #D8D8D8;
  border-collapse: collapse;
}
#container {
  height: 365px;
}
#defaultDialog.e-dialog .e-footer-content {
  padding: 0px 1px 4px ! important;
}
.tableStyle {
  margin: 17px;
  width: 304px;
}
.e-dialog .e-dlgcontent{
  padding: 10px 16px 10px;

```

```

    }
    .e-radio +label .e-label {
        line-height:18px;
    }
    td {
        padding: 6px;
    }
}
</style>

```

{% previewsample "page.domainurl/code-snippet/dialog/dlg-position-cs1" %}

### Prevent closing of modal dialog in Vue Dialog component

You can prevent closing of modal dialog by setting the [beforeClose](#) event argument cancel value to true.

In the following sample, the dialog is closed when you enter the username value with minimum 4 characters. Otherwise, it will not be closed.

#### APP.VUE

```

<template>
  <div>
    <div id="dialogTarget" class="control-section; position:relative"
    style="height:350px;">
      <!-- Render Button to open the modal Dialog -->
      <ejs-button id='dlgbtn' v-on:click.native="dlgBtnClick">Open
Dialog</ejs-button>
      <!-- Render modal Dialog -->
      <ejs-dialog ref="modalDialog"
:isModal='isModal':beforeClose="validation" :header='header'
:target='target' :width='width' :buttons='buttons'
:animationSettings='animationSettings' :close="modalDlgClose" >
        <div class='wrap'>
          <div class="e-float-input">
            <input id="textvalue" type="text" required/>
            <span class="e-float-line"></span>
            <label class="e-float-text">Username</label>
          </div>
          <div class="e-float-input">
            <input id="textvalue2" type="password" required/>
            <span class="e-float-line"></span>
            <label class="e-float-text">Password</label>
          </div>
        </div>
      </ejs-dialog>
    </div>
  </div>
</template>
<script>
import Vue from 'vue';
import { DialogPlugin } from '@syncfusion/ej2-vue-popups';
import { ButtonPlugin } from '@syncfusion/ej2-vue-buttons';
Vue.use(DialogPlugin);
Vue.use(ButtonPlugin);
export default {
  data: function() {
    return {

```

```

        target: "#dialogTarget",
        width: '300px',
        header: 'Sign in',
        buttons: [{ click: this.dlgButtonClick, buttonModel: { content:
'log in', isPrimary: true } }],
        isModal: true,
        animationSettings: { effect: 'None' }
    }
},
mounted: function(){
    document.getElementById('dlgbtn').focus();
},
methods: {
    dlgBtnClick: function() {
        this.$refs.modalDialog.show();
    },
    modalDlgClose: function() {
        document.getElementById('dlgbtn').style.display = '';
    },
    dlgButtonClick: function() {
        this.$refs.modalDialog.hide();
    },
    validation: function(args) {
        let text = document.getElementById('textvalue');
        let text1 = document.getElementById('textvalue2');
        if (text.value === "" && text1.value === "") {
            args.cancel= true;
            alert("Enter the username and password")
        } else if (text.value === "") {
            args.cancel= true;
            alert("Enter the username")
        } else if (text1.value === "") {
            args.cancel= true;
            alert("Enter the password")
        } else if (text.value.length < 4) {
            args.cancel= true;
            alert("Username must be minimum 4 characters")
        } else {
            args.cancel= false;
            document.getElementById("textvalue").value = "";
            document.getElementById("textvalue2").value = "";
        }
    }
}
}
</script>
<style>
@import "../..../node_modules/@syncfusion/ej2-vue-popups/styles/material.css";
#app {
    color: #008cff;
    height: 40px;
    left: 45%;
    position: absolute;
    top: 45%;
    width: 30%;
}
.control-section {

```

```

        height: 100%;
        min-height: 200px;
    }
    .wrap {
    box-sizing: border-box;
    margin: 0 auto;
    width: 260px;
    }
    .e-dlg-container .e-float-input {
    margin: 17px 0;
    }
    .e-dlg-container .wrap .e-control .e-btn {
        margin: 3% 26%;
    }
    .text-center {
        text-align: center;
    }
    #content {
        margin-top: 12px;
    }
    .e-dlg-container .e-footer-content {
        padding: 20px 0 0;
        width: 100%;
    }
    .e-dlg-container .e-dialog .e-footer-content .e-btn {
        width: 100%;
        height: 36px;
        margin-left: 0px;
    }
    .e-dlg-container .e-dialog .e-footer-content {
        padding: 0 18px 18px;
    }
    .e-dlg-container .e-dialog .e-dlg-header-content .e-dlg-header {
        color: #333;
        font-weight: bold;
        text-align: center;
        width: 100%;
        font-size: 20px;
    }
}
</style>

```

{% previewsample "page.domainurl/code-snippet/dialog/dlg-validation-cs1" %}

### Prevent the focus on the first element in Vue Dialog component

By default, the dialog focuses on the first elements of the content area which can be active and focusable. You can prevent this default focusing behavior using the [open](#) event and by enabling the `preventFocus` argument.

Bind the open event and enable the `preventFocus` argument within an event like the below sample.

#### APP.VUE

```

<template>
  <div>
    <div id="dialogTarget" class="control-section; position:relative"
      style="height:300px;">

```



```

        <!-- Render Button to open the Dialog -->
        <ejs-button id='dlgbtn' v-on:click.native="dialogBtnClick">Open
Dialog</ejs-button>
        <!-- Render Dialog -->
        <ejs-dialog id="dialog" ref="Dialog" :header='header'
:target='target' :width='width' :buttons='buttons' :open="onOpen">
            <div class='form-group'><label for='email'>Email:</label>
                <input type='email' class='form-control' id='email'>
            </div>
            <div class='form-group'>
                <label for='comment'>Password:</label>
                <input type='password' class='form-control' id='password'>
            </div>
        </ejs-dialog>
    </div>
</div>
</template>
<script>
import Vue from 'vue';
import { DialogPlugin } from '@syncfusion/ej2-vue-popups';
import { ButtonPlugin } from '@syncfusion/ej2-vue-buttons';
Vue.use(DialogPlugin);
Vue.use(ButtonPlugin);
export default {
    data: function() {
        return {
            target: "#dialogTarget",
            width: '300px',
            header: 'Sign In',
            buttons: [{ click: this.dlgButtonClick, buttonModel: { content:
'Ok', isPrimary: true } }, { click: this.dlgButtonClick, buttonModel: {
content: 'Cancel' } }]
        }
    },
    methods: {
        dialogBtnClick: function() {
            this.$refs.Dialog.show();
        },
        dlgButtonClick: function() {
            this.$refs.Dialog.hide();
        },
        onOpen: function(args) {
            args.preventDefault = true;
        }
    }
}
</script>
<style>
@import "../../node_modules/@syncfusion/ej2-vue-popups/styles/material.css";
#app {
    color: #008cff;
    height: 40px;
    left: 45%;
    position: absolute;
    top: 45%;
    width: 30%;
}

```

```
.control-section {
  height: 100%;
  min-height: 200px;
}
</style>
```

```
{% previewsample "page.domainurl/code-snippet/dialog/dlg-focus-cs1" %}
```

### Prevent opening of the dialog in Vue Dialog component

You can prevent opening of the dialog by setting the [beforeOpen](#) event argument cancel value to true.

In the following sample, the success dialog is opened when you enter the username value with minimum 4 characters. Otherwise, it will not be opened.

### APP.VUE

```
<template>
  <div>
    <div id="dialogTarget" class="control-section; position:relative"
style="height:300px;">
      <div class="login-form">
        <div class='wrap'>
          <div id="heading">Sign in</div>
          <div class="e-float-input">
            <input id="textvalue" type="text" required/>
            <span class="e-float-line"></span>
            <label class="e-float-text">Username</label>
          </div>
          <div class="e-float-input">
            <input id="textvalue2" type="password" required/>
            <span class="e-float-line"></span>
            <label class="e-float-text">Password</label>
          </div>
          <div class="button-contain">
            <ejs-button id="dialogbtn" cssClass="e-info" v-
on:click.native="dialogBtnClick" >Log in</ejs-button>
          </div>
        </div>
      </div>
      <ejs-dialog id="dialog" ref="Dialog" :header='header'
:isModal='isModal' :target='target' :width='width' :buttons='buttons'
:animationSettings='animationSettings' :visible='visible' :content='content'
:beforeOpen="validation" :close="modalDlgClose">
        </ejs-dialog>
      </div>
    </div>
  </template>
<script>
import Vue from 'vue';
import { DialogPlugin } from '@syncfusion/ej2-vue-popups';
import { ButtonPlugin } from '@syncfusion/ej2-vue-buttons';
Vue.use(DialogPlugin);
Vue.use(ButtonPlugin);
export default {
  data: function() {
    return {
```

```

        target: "#dialogTarget",
        width: '250px',
        visible: false,
        header: 'Success',
        isModal: true,
        content: 'Congratulations! Login Success',
        buttons: [{ click: this.dlgButtonClick, buttonModel: { content:
'Dismiss', isPrimary: true } }],
        isModal: true,
        animationSettings: { effect: 'None' }
    }
},
mounted: function(){
    document.getElementById('dialogbtn').focus();
},
methods: {
    dialogBtnClick: function() {
        this.$refs.Dialog.show();
    },
    modalDlgClose: function() {
        document.getElementById('dialogbtn').style.display = '';
    },
    dlgButtonClick: function() {
        this.$refs.Dialog.hide();
    },
    validation: function(args) {
        let text = document.getElementById('textvalue');
        let text1 = document.getElementById('textvalue2');
        if (text.value === "" && text1.value === "") {
            args.cancel= true;
            alert("Enter the username and password")
        } else if (text.value === "") {
            args.cancel= true;
            alert("Enter the username")
        } else if (text1.value === "") {
            args.cancel= true;
            alert("Enter the password")
        } else if (text.value.length < 4) {
            args.cancel= true;
            alert("Username must be minimum 4 characters")
        } else {
            args.cancel= false;
            document.getElementById("textvalue").value = "";
            document.getElementById("textvalue2").value = "";
        }
    }
}
}
</script>
<style>
@import "../../../node_modules/@syncfusion/ej2-vue-popups/styles/material.css";
#app {
    color: #008cff;
    height: 40px;
    left: 45%;
    position: absolute;
    top: 45%;

```

```

        width: 30%;
    }
    .control-section {
        height: 100%;
        min-height: 200px;
    }
    .wrap {
        box-sizing: border-box;
        margin: 0 auto;
        padding: 20px 30px;
        width: 340px;
        background: #f7f7f7;
    }
    .wrap .e-float-input { /* csslint allow: adjoining-classes */
        margin: 17px 0;
    }
    .text-center {
        text-align: center;
    }
    #content {
        margin-top: 12px;
    }
    .button-contain {
        padding: 20px 0 0;
        width: 100%;
    }
    .button-contain .e-btn { /* csslint allow: adjoining-classes */
        width: 100%;
        height: 36px;
    }
    #heading {
        color: #333;
        font-weight: bold;
        margin: 0 0 15px;
        text-align: center;
        font-size: 20px;
    }
    .login-form {
        width: 340px;
        margin: 50px auto;
    }
    #dialog.e-dialog .e-footer-content {
        text-align: center;
    }
}
</style>

```

{% previewsample "page.domainurl/code-snippet/dialog/dlg-check-cs1" %}

[Read all the values from dialog on button click in Vue Dialog component](#)

You can read the dialog element values by binding the action handler to the footer buttons. The buttons property provides the options to bind events to the action buttons. For detailed information about buttons, refer to the [footer](#) section.

In the below sample, value of input elements within the dialog has been checked in the footer button click event and send the values as the content of confirmation dialog.

**APP.VUE**

```

<template>
  <div>
    <div id="target">
      <center> <ejs-button id="dialogbtn" v-
on:click.native="btnClick">Open</ejs-button></center>
      <ejs-dialog id="dialog" ref="Dialog" :header='header'
:target='target' :width='width' :buttons='buttons' :visible='visible'
:content='content' :animationSettings='animateSettings'
:showCloseIcon="showCloseIcon">
        </ejs-dialog>
        <ejs-dialog id="modalDialog" ref="ModalDialog"
:header='model_header' :isModal='isModal' :target='target'
:buttons='mbuttons' :animationSettings='animationSettings'
:visible='visible' >
          </ejs-dialog>
        </div>
      </div>
    </div>
  </template>
<script>
import Vue from "vue";
import { DialogPlugin } from '@syncfusion/ej2-vue-popups';
import { ButtonPlugin } from '@syncfusion/ej2-vue-buttons';
Vue.use(DialogPlugin);
Vue.use(ButtonPlugin);
export default {
  data: function() {
    return {
      header: 'User details',
      content: "<form><div class='form-group'><label
for='name'>Name:</label>" +
        "<input type='name' value='' class='form-control' id='name'>" +
        "</div><div class='form-group'><label for='email'>Email
Id:</label>" +
        "<input type='email' value='user@syncfusion.com' class='form-
control' id='email'>" +
        "</div><div class='form-group'><label for='contact'>Contact
Number:</label>" +
        "<input type='contact' class='form-control' id='contact'>" +
        "</div><div class='form-group'><label
for='address'>Address:</label>" +
        "<textarea class='form-control' rows='2'
id='address'></textarea>" +
        "</div></form>",
      showCloseIcon: true,
      visible: false,
      buttons: [{
        buttonModel: { isPrimary: true, content: 'Submit' }, click:
this.createModalDialog }
    ],
      target: document.querySelector('body'),
      width: '400px',
      animationSettings: { effect: 'Zoom' },
      animateSettings: { effect: 'Zoom' },
      model_header: 'User details',
      showCloseIcon: true,

```

```

        isModal: true,
        mbuttons: [{
            buttonModel: {isPrimary: true, content: 'Yes'}, click:
function() {
            this.hide();
        }, {
            buttonModel: {isPrimary: false, content: 'No'}, click:
this.showDialog
        }
    ],
    },
    methods: {
        btnClick: function () {
            this.$refs.Dialog.show();
        },
        createModalDialog: function() {
            this.$refs.Dialog.hide();
            this.$refs.ModalDialog.content = this.getDynamicContent();
            this.$refs.ModalDialog.show();
        },
        showDialog: function() {
            this.$refs.ModalDialog.hide();
            this.$refs.Dialog.show();
        }
        getDynamicContent: function() {
            if (!document.getElementById('dialog')) {return;}
            let input =
document.getElementById('dialog').querySelector('#name');
            let email =
document.getElementById('dialog').querySelector('#email');
            let contact =
document.getElementById('dialog').querySelector('#contact');
            let address =
document.getElementById('dialog').querySelector('#address');
            let template = "<div class='row'><div class='col-xs-6 col-sm-6
col-lg-6 col-md-6'><b>Confirm your details</b></div>" +
                "</div><div class='row'><div class='col-xs-6 col-sm-6 col-lg-6
col-md-6'><span id='name'> Name: </span>" +
                "</div><div class='col-xs-6 col-sm-6 col-lg-6 col-md-6'><span
id='nameValue'>"+ input.value + "</span> </div></div>" +
                "<div class='row'><div class='col-xs-6 col-sm-6 col-lg-6 col-md-
6'><span id='email'> Email: </span>" +
                "</div><div class='col-xs-6 col-sm-6 col-lg-6 col-md-6'><span
id='emailValue'>"+ email.value +
                "</span></div></div><div class='row'><div class='col-xs-6 col-
sm-6 col-lg-6 col-md-6'>"+
                "<span id='Contact'> Contact number: </span></div><div
class='col-xs-6 col-sm-6 col-lg-6 col-md-6'>"+
                "<span id='contactValue'>"+ contact.value +
                "</span></div></div><div class='row'><div class='col-xs-6 col-sm-6 col-lg-6
col-md-6'>"+
                "<span id='Address'> Address: </span> </div><div class='col-xs-6
col-sm-6 col-lg-6 col-md-6'><span id='AddressValue'>"+ address.value
                + "</span></div></div>"
            return template;
        },
    },

```

```

    }
  }
</script>
<style>
@import ".../node_modules/@syncfusion/ej2-vue-popups/styles/material.css";
@import
"https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css";
#app {
  color: #008cff;
  height: 40px;
  left: 45%;
  position: absolute;
  top: 45%;
  width: 30%;
}
#modalDialog {
  margin: 20px;
}
html,
body {
  height: 100%;
  width: 100%;
  overflow: hidden;
}
.row {
  padding: 10px 3px;
}
</style>

```

```
{% previewsample "page.domainurl/code-snippet/dialog/read-all-cs1" %}
```

### Customize the dialog appearance in Vue Dialog component

You can customize the dialog appearance by providing dialog template as string or HTML element to the [content](#) property. In the following sample, dialog is customized as error window appearance.

#### APP.VUE

```

<template>
  <div>
    <div id="dialogTarget" class="control-section; position:relative"
style="height:300px;">
      <center><ejs-button ref='button' id="dialogbtn" cssClass="e-info" v-
on:click.native="dialogBtnClick">Open</ejs-button></center>
      <ejs-dialog id="dialog" ref="Dialog" :header='header'
:showCloseIcon='showCloseIcon' :target='target' :width='width'
:buttons='buttons' :animationSettings='animationSettings' :visible='visible'
:content='content'>
        </ejs-dialog>
      </div>
    </div>
  </template>
<script>
import Vue from 'vue';
import { DialogPlugin } from '@syncfusion/ej2-vue-popups';
import { ButtonPlugin } from '@syncfusion/ej2-vue-buttons';
Vue.use(DialogPlugin);

```

```

Vue.use(ButtonPlugin);
export default {
  data: function() {
    return {
      header: 'File and Folder Rename',
      content: "<div class = 'dialog-content'><div class='msg-wrapper col-lg-12'><span class='e-icons close-icon col-lg-2'></span>" +
        "<span class='error-msg col-lg-10'>Can not rename 'pictures' because a file or folder with that name already exists </span>" +
        "</div><div class='error-detail col-lg-8'><span>Specify a different name</span> </div></div>",
      showCloseIcon: true,
      visible: false,
      buttons: [{
        buttonModel: { isPrimary: true, content: 'close' }, click:
function() {
          this.hide();
        }
      }],
      target: document.querySelector('body'),
      width: '400px',
      animationSettings: { effect: 'Zoom' }
    }
  },
  methods: {
    dialogBtnClick: function() {
      this.$refs.Dialog.show();
    }
  }
}
</script>
<style>
@import "../../../node_modules/@syncfusion/ej2-vue-popups/styles/bootstrap.css";
@import
"https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css";
#app {
  color: #008cff;
  height: 40px;
  left: 45%;
  position: absolute;
  top: 45%;
  width: 30%;
}
.row {
  padding: 10px 3px;
}
span.close-icon {
  width: 24px;
  height: 24px;
  position: relative;
  /* background-color: yellow; */
  display: inline-block;
  top: 17px;
}
span.error-msg {
  color: #66afe9;

```



```

        display: inline-block;
        position: relative;
        top: 18px;
        left: 20px;
        width: 80%;
        padding: 0px 16px 27px;
    }
    .error-detail {
        position: relative;
        left: 45px;
        margin: 20px 0px 21px;
    }
    .close-icon:before {
        content: '\e7e9';
        font-size: 26px;
        color: #d9534f;
        position: relative;
        left: 5px;
    }
    .e-dialog .e-footer-content {
        background-color: #f8f8f8;
    }
    .e-dialog, .e-dialog .e-dlg-header-content {
        background-color: #d9edf7;
    }
    .e-dialog {
        padding: 3px;
    }
    .e-dialog .e-dlg-header-content {
        padding: 10px;
    }
    .e-dialog .e-footer-content {
        padding: 8px 12px;
    }
    .e-dialog .e-dlg-content {
        padding: 15px 0px 0px;
    }
</style>

```

{% previewsample "page.domainurl/code-snippet/dialog/custom-dialog-cs1" %}

### Close dialog while click on outside of dialog in Vue Dialog component

By default, dialog can be closed by pressing Esc key and clicking the close icon on the right of dialog header. It can also be closed by clicking outside of the dialog using hide method.

Set the [CloseOnEscape](#) property value to false to prevent closing of the dialog when pressing Esc key.

In the following sample, dialog is closed when clicking outside the dialog area using [hide](#) method.

### APP.VUE

```

<template>
  <div>
    <div id="dialogTarget" class="control-section; position:relative" v-
on:click.self="btnClick" style="height:350px;">
      <center><ejs-button ref='button' id="dialogbtn" cssClass="e-info" v-
on:click.native="dialogBtnClick">Open</ejs-button></center>

```

```

    <ejs-dialog id="dialog" ref="Dialog" :header='header'
:showCloseIcon='showCloseIcon' :target='target' :width='width'
:buttons='buttons' :animationSettings='animationSettings' :visible='visible'
:content='content' :closeOnEscape='closeOnEscape'>
    </ejs-dialog>
  </div>
</div>
</template>
<script>
import Vue from 'vue';
import { DialogPlugin } from '@syncfusion/ej2-vue-popups';
import { ButtonPlugin } from '@syncfusion/ej2-vue-buttons';
Vue.use(DialogPlugin);
Vue.use(ButtonPlugin);
export default {
  data: function() {
    return {
      header: 'Delete Multiple Items',
      content: "Are you sure you want to permanently delete all of
these items?",
      showCloseIcon: true,
      visible: false,
      buttons: [{ buttonModel: { isPrimary: true, content: 'Yes' },
click: this.btnClick }, { buttonModel: { content: 'No' }, click:
this.btnClick }],
      target: document.body,
      height: 'auto',
      width: '300px',
      animationSettings: { effect: 'Zoom' },
      closeOnEscape: true
    }
  },
  methods: {
    dialogBtnClick: function() {
      this.$refs.Dialog.show();
    },
    btnClick: function() {
      this.$refs.Dialog.hide();
    }
  }
}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-vue-popups/styles/material.css";
#app {
  color: #008cff;
  height: 40px;
  left: 45%;
  position: absolute;
  top: 45%;
  width: 30%;
}
</style>

```

{% previewsample "page.domainurl/code-snippet/dialog/outside-click-cs1" %}

### Add an icons to dialog buttons in Vue Dialog component

You can add icons to the dialog buttons using the [buttons](#) property or [footerTemplate](#) property . For detailed information about dialog buttons, refer to the [documentation](#) section.

In the following sample, dialog footer buttons are customized with icons using `buttons` property.

#### APP.VUE

```
<template>
  <div>
    <div id="target">
      <center><ejs-button ref='button' id="dialogbtn" cssClass="e-info" v-
on:click.native="dialogBtnClick">Open</ejs-button></center>
      <ejs-dialog id="dialog" ref="Dialog" :header='header'
:showCloseIcon='showCloseIcon' :target='target' :width='width'
:buttons='buttons' :animationSettings='animationSettings' :visible='visible'
:content='content' :closeOnEscape='closeOnEscape'>
        </ejs-dialog>
      </div>
    </div>
  </template>
<script>
import Vue from 'vue';
import { DialogPlugin } from '@syncfusion/ej2-vue-popups';
import { ButtonPlugin } from '@syncfusion/ej2-vue-buttons';
Vue.use(DialogPlugin);
Vue.use(ButtonPlugin);
export default {
  data: function() {
    return {
      header: 'Delete Multiple Items',
      content: "Are you sure you want to permanently delete all of
these items?",
      showCloseIcon: true,
      visible: false,
      buttons: [{ buttonModel: { isPrimary: true, content: 'Yes',
iconCss: 'e-icons e-ok-icon' }, click: this.btnClick }, { buttonModel: {
content: 'No', iconCss: 'e-icons e-close-icon' }, click: this.btnClick }],
      target: document.body,
      height: 'auto',
      width: '300px',
      animationSettings: { effect: 'Zoom' },
      closeOnEscape: true
    }
  },
  methods: {
    dialogBtnClick: function() {
      this.$refs.Dialog.show();
    },
    btnClick: function() {
      this.$refs.Dialog.hide();
    }
  }
}
</script>
<style>
@import "../../../node_modules/@syncfusion/ej2-vue-popups/styles/material.css";
```

```
#app {
  color: #008cff;
  height: 40px;
  left: 45%;
  position: absolute;
  top: 45%;
  width: 30%;
}
#target {
  height: 100%;
  min-height: 350px;
}
.e-ok-icon::before {
  content: '\e7ff';
}
.e-close-icon::before {
  content: '\e825';
}
</style>
```

{% previewsample "page.domainurl/code-snippet/dialog/template-buttons-cs1" %}

In the following sample, dialog footer buttons are customized with icons using `footerTemplate` property.

### APP.VUE

```
<template>
  <div>
    <div id="target">
      <center><ejs-button ref='button' id="dialogbtn" cssClass="e-info" v-
on:click.native="dialogBtnClick">Open</ejs-button></center>
      <ejs-dialog id="dialog" ref="Dialog" :header='header'
:showCloseIcon='showCloseIcon' :target='target' :width='width'
:animationSettings='animationSettings' :footerTemplate='footer'
:visible='visible' :content='content' :closeOnEscape='closeOnEscape'>
        </ejs-dialog>
      </div>
    </div>
  </div>
</template>
<script>
import Vue from 'vue';
import { DialogPlugin } from '@syncfusion/ej2-vue-popups';
import { ButtonPlugin } from '@syncfusion/ej2-vue-buttons';
Vue.use(DialogPlugin);
Vue.use(ButtonPlugin);
export default {
  data: function() {
    return {
      header: 'Delete Multiple Items',
      content: "Are you sure you want to permanently delete all of
these items?",
      showCloseIcon: true,
      visible: false,
      footer: '<div><button id="Button1" class="e-control e-btn e-
primary e-flat" data-ripple="true"><span class="e-btn-icon e-icons e-ok-icon
```

```

e-icon-left"></span>Yes</button><button id="Button2" class="e-control e-btn
e-flat" data-ripple="true"><span class="e-btn-icon e-icons e-close-icon e-
icon-left"></span>No</button></div>',
    target: document.body,
    height: 'auto',
    width: '300px',
    animationSettings: { effect: 'Zoom' },
    closeOnEscape: true
  }
},
mounted: function() {
  document.getElementById('Button1').onclick = () => {
    this.$refs.Dialog.hide();
  };
  document.getElementById('Button2').onclick = () => {
    this.$refs.Dialog.hide();
  };
},
methods: {
  dialogBtnClick: function() {
    this.$refs.Dialog.show();
  },
  btnClick: function() {
    this.$refs.Dialog.hide();
  }
}
}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-vue-popups/styles/material.css";
#app {
  color: #008cff;
  height: 40px;
  left: 45%;
  position: absolute;
  top: 45%;
  width: 30%;
}
#target {
  height: 100%;
  min-height: 350px;
}
.e-ok-icon::before {
  content: '\e7ff';
}
.e-close-icon::before {
  content: '\e825';
}
</style>

```

{% previewsample "page.domainurl/code-snippet/dialog/template-footer-cs1" %}

### Add a minimize maximize buttons in Vue Dialog component

Vue Dialog allows end users to either minimize or maximize the Dialog component. You can add minimize and maximize custom buttons near the close icon in the Dialog header using the

[headerTemplate](#) property and handle the actions in the button click events, as shown in the following sample.

#### APP.VUE

```
<template>
  <div>
    <div id="target">
      <center><ejs-button ref='button' id="dialogbtn" cssClass="e-info" v-
on:click.native="dialogBtnClick">Open</ejs-button></center>
      <ejs-dialog id="dialog" ref="Dialog" :header='headerTemplate'
:showCloseIcon='showCloseIcon' :target='target' :width='width'
:buttons='buttons' :animationSettings='animationSettings' :visible='visible'
:content='content' :closeOnEscape='closeOnEscape' :created="created">
        </ejs-dialog>
      </div>
    </div>
  </template>
<script>
import Vue from 'vue';
import { DialogPlugin } from '@syncfusion/ej2-vue-popups';
import { ButtonPlugin } from '@syncfusion/ej2-vue-buttons';
Vue.use(DialogPlugin);
Vue.use(ButtonPlugin);
var headerTemplateVue = Vue.component("demo1", {
  template: '<span><span class="title">Dialog</span><span class="e-icons
sf-icon-Maximize" id="max-btn" title="Maximize" ></span><span class="e-icons
sf-icon-Minimize" id="min-btn" title="Minimize"></span></span>',
  data() {
    return {
      data: {}
    };
  }
});
var isFullScreen;
var dialogOldPosition;
export default {
  data: function() {
    return {
      headerTemplate: function () {
        return { template: headerTemplateVue }
      },
      content: "This is a dialog with minimize and maximize buttons",
      showCloseIcon: true,
      visible: false,
      buttons: [{ buttonModel: { isPrimary: true, content: 'Yes',
iconCss: 'e-icons e-ok-icon' }, click: this.btnClick }, { buttonModel: {
content: 'No', iconCss: 'e-icons e-close-icon' }, click: this.btnClick }],
      target: document.body,
      height: 'auto',
      width: '300px',
      animationSettings: { effect: 'Zoom' },
      closeOnEscape: true,
    }
  },
  methods: {
    created: function () {
```

```

        document.getElementById("min-btn").addEventListener("click",
this.minimize);
        document.getElementById("max-btn").addEventListener("click",
this.maximize);
    },
    dialogBtnClick: function() {
        this.$refs.Dialog.show();
    },
    btnClick: function() {
        this.$refs.Dialog.hide();
    }
    maximize: function(){
        var maximizeIcon;
        if (this.$refs.Dialog.$el.classList.contains('dialog-
minimized')) {
            this.$refs.Dialog.$el.querySelector('#min-
btn').classList.add('sf-icon-Minimize');
            this.$refs.Dialog.$el.querySelector('#min-
btn').classList.remove('sf-icon-Restore');
            this.$refs.Dialog.$el.querySelector('#min-
btn').setAttribute('title', 'Minimize');
        }
        if (!this.$refs.Dialog.$el.classList.contains('dialog-
maximized') && !this.isFullScreen) {
            maximizeIcon = this.$refs.Dialog.$el.querySelector(".e-dlg-
header-content .sf-icon-Maximize");
        } else {
            maximizeIcon = this.$refs.Dialog.$el.querySelector(".e-dlg-
header-content .sf-icon-Restore");
        }
        if (!this.$refs.Dialog.$el.classList.contains('dialog-
maximized')) {
            this.$refs.Dialog.$el.classList.add('dialog-maximized');
            this.$refs.Dialog.show(true);
            maximizeIcon.classList.add('sf-icon-Restore');
            maximizeIcon.setAttribute('title', 'Restore');
            maximizeIcon.classList.remove('sf-icon-Maximize');
            this.$refs.Dialog.$el.querySelector('.e-dlg-
content').classList.remove('hide-content');
            this.isFullScreen = true;
        } else {
            this.$refs.Dialog.$el.classList.remove('dialog-maximized');
            this.$refs.Dialog.show(false);
            maximizeIcon.classList.remove('sf-icon-Restore');
            maximizeIcon.classList.add('sf-icon-Maximize');
            maximizeIcon.setAttribute('title', 'Maximize');
            this.$refs.Dialog.$el.querySelector('.e-dlg-
content').classList.remove('hide-content');
            this.$refs.Dialog.position = this.dialogOldPositions;
            this.$refs.Dialog.dataBind();
            this.isFullScreen = false;
        }
    }
    minimize: function(){
        var minimizeIcon = this.$refs.Dialog.$el.querySelector(".e-dlg-
header-content .sf-icon-Minimize");

```

```

        if (!this.$refs.Dialog.$el.classList.contains('e-dlg-
fullscreen')) {
            if (!this.$refs.Dialog.$el.classList.contains('dialog-
minimized')) {
                this.dialogOldPositions = { X:
this.$refs.Dialog.$el.ej2_instances[0].position.X, Y:
this.$refs.Dialog.$el.ej2_instances[0].position.Y }
                this.$refs.Dialog.$el.classList.add('dialog-minimized');
                this.$refs.Dialog.$el.classList.remove('dialog-maximized');
                this.$refs.Dialog.$el.querySelector('.e-dlg-
content').classList.add('hide-content');
                this.$refs.Dialog.$el.ej2_instances[0].position = { X:
'center', Y: 'bottom' };
                this.$refs.Dialog.dataBind();
                minimizeIcon.classList.add('sf-icon-Restore');
                minimizeIcon.setAttribute('title', 'Restore');
            } else {
                this.$refs.Dialog.$el.classList.remove('dialog-minimized');
                this.$refs.Dialog.$el.querySelector('.e-dlg-
content').classList.remove('hide-content');
                minimizeIcon.classList.add('sf-icon-Minimize');
                minimizeIcon.setAttribute('title', 'Minimize');
                minimizeIcon.classList.remove('sf-icon-Restore');
                this.$refs.Dialog.$el.ej2_instances[0].position =
this.dialogOldPositions;
                this.$refs.Dialog.dataBind();
            }
        } else {
            this.$refs.Dialog.show(false);
            this.$refs.Dialog.$el.classList.remove('dialog-maximized');
            this.$refs.Dialog.$el.classList.add('dialog-minimized');
            this.$refs.Dialog.$el.querySelector('.e-dlg-
content').classList.add('hide-content');
            minimizeIcon.classList.remove('sf-icon-Minimize');
            minimizeIcon.removeAttribute('title');
            this.$refs.Dialog.$el.ej2_instances[0].position = { X: 'center',
Y: 'bottom' };
            this.$refs.Dialog.dataBind();
            this.isFullScreen = true;
        }
    }
}
}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-vue-popups/styles/material.css";
#app {
    color: #008cff;
    height: 40px;
    left: 45%;
    position: absolute;
    top: 45%;
    width: 30%;
}
#target {
    height: 100%;
    min-height: 350px;

```



```

}
.e-ok-icon::before {
    content: '\e7ff';
}
.e-close-icon::before {
    content: '\e825';
}
.control-section {
    height: 100%;
    min-height: 350px;
}
.dialog-minimized {
    top: 261px !important;
}
.control-section {
    width: 450px;
    height: 500px;
    border: 1px solid black;
}
.e-dialog .e-dlg-header {
    width: auto;
}
.e-dialog .e-dlg-header .e-icons.sf-icon-Maximize::before,
.e-dialog .e-dlg-header .e-icons.sf-icon-Minimize::before,
.e-dialog .e-dlg-header .e-icons.sf-icon-Restore::before {
    position: relative;
}
.e-dialog .e-dlg-header .e-icons.sf-icon-Minimize,
.e-dialog .e-dlg-header .e-icons.sf-icon-Maximize,
.e-dialog .e-dlg-header .e-icons.sf-icon-Restore {
    font-size: 14px;
    width: 30px;
    height: 30px;
    line-height: 30px;
    float: right;
    position: relative;
    text-align: center;
    cursor: pointer;
}
.e-dialog .e-dlg-header .e-icons.sf-icon-Minimize:hover, .e-dialog .e-dlg-
header .e-icons.sf-icon-Maximize:hover,
.e-dialog .e-dlg-header .e-icons.sf-icon-Restore:hover {
    background-color: #e0e0e0;
    border-color: transparent;
    box-shadow: 0 0 0 transparent;
    border-radius: 50%;
}
.e-dialog .e-dlg-header .e-icons.sf-icon-Minimize,
.e-dialog .e-dlg-header .e-icons.sf-icon-Restore {
    padding-left: 5px;
    padding-right: 5px;
}
.e-dialog .e-dlg-header {
    position: relative;
    top: 1px;
}

```

```

.e-dialog .e-dlg-content.hide-content, .e-dialog .e-footer-content.hide-
content {
    display: none;
}

.e-dialog .e-dlg-header span.title {
    width: 60px;
    display: inline-block;
}

@font-face {
    font-family: 'Min-Max_FONT';
    src: url(data:application/x-font-ttf;charset=utf-
8;base64,AAEAAAAKAIAAAwAgTlMvMjltSfUAAAEoAAAVmNtYXNlbnE+dkAAABlAAAAADxnbHlmQCK
X6AAAAdwAAADkaGVhZBK7D5EAAADQAAAAANmhoZWEIVQQGAAAArAAAACRobXR4FAAAAAAAYAAAAA
UbG9jYQBaAJwAAAHQAAAAADG1heHABEGAgAAABCAAAACBuYw1l8Rnd5AAAAAsAAAAJhcG9zdDbKxec
AAAUkAAAAATwABAAAEAAAAAFwEAAAAAAD+AABAAAAAAAAAAAAAAAAAAAAABQABAAAAAQAAK4KTXV8
PPPUACwQAAAAANfSZU4AAAAA19JlTgAAAAAD+AP4AAAACAACAAAAAAAAAAAAEAAAAFABQAAwAAAAA
AAgAAAAAoACgAAAP8AAAAAAAAAAAAQQAQAAZAAABQAAAokCzAAAAI8CiQLMAAAB6wAyAQgAAAIABQMAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAUGZFZABA5wDnAwQAAAAAXAQAAAAAAAAABAAAAAAAABAAAAAQAAAA
EAAAABAAAAAQAAAAAAAACAAAAAwAAABQAAwABAAAAFAAEACgAAAAEAAQAAQAA5wP//wAA5wD//wA
AAAEABAAAAEAAgADAAQAAAAAAAA4AKgBMAHIAAQAAAAADkwIyAAMAABMhNSFtAyb82gHOZAAAAAM
AAAAAA/gD+AADAACACwAAAREhESUVITUDIREha5P82gMm/Np1A/D8EALK/aMCXcl1Zfx1A/AAAQA
AAAAADkwOSAAsAABMJARcJATcJAScJAWwBTf6zRwFNAU1H/rMBTUf+s/6zA0v+tf61RwFL/rVHAUs
BS0f+tQFLAADAAAAA4A/gABQALABMAABMzIREhESURIXEhNQcjESE1MxEh0mQB1P2jAyZ1/gh
kygMmyvzaAsr9owJdyf2jAfhlZfzaygMmAAAAAAAAASAN4AAQAAAAAAAAABAAAAAQAAAAAAAAQAMAAE
AAQAAAAAAAAAgAHAA0AAQAAAAAAAAAwAMABQAAQAAAAAAAAABAAACAAAQAAAAAAAAABQALACwAAQAAAAAABgA
MADCAAAQAAAAAACgAsAEMAAQAAAAAACwASAG8AAwABBAkAAAACATEAAwABBAkAAQAYAIMAAwABBAk
AAgAOAJsAAwABBAkAAwAYAKkAAwABBAkABAAYAMEAAwABBAkABQAWANKAAwABBAkABgAYA08AAwA
BBakACgBYAQCAAwABBAkACwAkAV8gTWluLU1heF9GT05UUmVndWxhc1lpci1NYXhfrk9OVE1pci1
NYXhfrk9OVFZ1cnNpb24gMS4wTWluLU1heF9GT05URm9udCBnZW51cmF0ZWQgdXNpbmcgU3luY2Z
1c2l1bviBNZXRYbyBTdHVkaW93d3cuc3luY2Z1c2l1bvi5jb20AIABNAGkAbgAtAE0AYQB4AF8ARgB
PAE4AVABSAGUAZwB1AGwAYQByAE0AaQBucAC0ATQBhAHgAXwBGAE8ATgBUAE0AaQBucAC0ATQBhAHg
AXwBGAE8ATgBUAFYAZQByAHMAaQBvAG4AIAAxAAC4AMABNAGkAbgAtAE0AYQB4AF8ARgBPAAE4AVAB
GAG8AbgB0ACAAZwB1AG4AZQByAGEAdABlAGQAIAB1AHMAaQBucAGcAIABTAHkAbgBjAGYAdQBzAGk
AbwBuACAATQBlAHQACgBvACAAUwB0AHUAZABpAG8AdwB3AHcALgBzAHkAbgBjAGYAdQBzAGkAbwB
uAC4AYwBvAG0AAAAAAGAAAAAAAAAKAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAFAQIBAwEEAQUBBgA
ITWluaW1pemUITWF4aW1pemUFQ2xvc2UHUHVzdG9yZQAAAA==) format('truetype');
    font-weight: normal;
    font-style: normal;
}

[class^="sf-icon-"], [class*=" sf-icon-"] {
    font-family: 'Min-Max_FONT' !important;
    speak: none;
    font-size: 55px;
    font-style: normal;
    font-weight: normal;
    font-variant: normal;
    text-transform: none;
    line-height: 1;
    -webkit-font-smoothing: antialiased;
    -moz-osx-font-smoothing: grayscale;
}

.sf-icon-Minimize:before {
    content: "\e700";
}

.sf-icon-Maximize:before {

```

```

        content: "\e701";
    }
    .sf-icon-Close:before {
        content: "\e702";
    }
    .sf-icon-Restore:before {
        content: "\e703";
    }
}
</style>

```

{% previewsample "page.domainurl/code-snippet/dialog/min-max-buttons-cs1" %}

### Setting max height to the dialog in Vue Dialog component

By default, the maxHeight for the Dialog is calculated based on the target. If the target is not specified externally, the Dialog consider the body as target and will calculate the maxHeight based on it. We have an option to set the maxHeight of the Dialog in the [beforeOpen](#) event.

#### APP.VUE

```

<template>
  <div>
    <div id="target" class="control-section; position:relative"
    style="height:360px;">
      <!-- Render Button to open the Dialog -->
      <ejs-button id='dlgbtn' v-on:click.native="btnClick">Open
Dialog</ejs-button>
      <!-- Render Dialog -->
      <ejs-dialog ref="footerDialog" :header='header' :target='target'
:width='width' :buttons='buttons' :visible='visible'
:showCloseIcon='showCloseIcon' :content='content' :close="dlgClose"
:beforeOpen='onBeforeOpen'>
      </ejs-dialog>
    </div>
  </div>
</template>
<script>
import Vue from 'vue';
import { DialogPlugin } from '@syncfusion/ej2-vue-popups';
import { ButtonPlugin } from '@syncfusion/ej2-vue-buttons';
Vue.use(DialogPlugin);
Vue.use(ButtonPlugin);
export default {
  data: function() {
    return {
      target: "#target",
      width: '800px',
      header: 'Dialog',
      visible: false,
      showCloseIcon: true,
      content: 'This is a Dialog with max-height',
      buttons: [{ click: this.dlgButtonClick, buttonModel: { content:
'OK', isPrimary: true } },
        { click: this.dlgButtonClick, buttonModel: { content: 'Cancel'
} }],
      animationSettings: { effect: 'None' }
    }
  }
}

```

```

    },
    mounted: function() {
        document.getElementById('dlgbtn').focus();
    },
    methods: {
        btnClick: function() {
            this.$refs.footerDialog.show();
        },
        dlgClose: function() {
            document.getElementById('dlgbtn').style.display = '';
        },
        dlgButtonClick: function() {
            this.$refs.footerDialog.hide();
        },
        onBeforeOpen: function(args: beforeOpenEventArgs) {
            args.maxHeight = '300px';
        }
    }
}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-vue-popups/styles/material.css";
#app {
    color: #008cff;
    height: 40px;
    left: 45%;
    position: absolute;
    top: 45%;
    width: 30%;
}
.control-section {
    height: 100%;
    max-height: 380px;
    overflow: hidden;
}
</style>

```

{% previewsample "page.domainurl/code-snippet/dialog/max-height-cs1" %}

## DocumentEditor

### Overview

The Document Editor component is used to create, edit, view, and print Word documents in web applications. All the user interactions and editing operations that run purely in the client-side provides much faster editing experience to the users.

### Key Features

- [Opens](#) the native Syncfusion Document Text (\*.sfdt) format documents in the client-side.
- [Saves the documents](#) in the client-side as Syncfusion Document Text (.sfdt) and Word document (.docx).
- Supports document elements like text, [image](#), [table](#), fields, [bookmark](#), [shapes](#), [section](#), [header](#) and [footer](#).

- Supports the commonly used fields like [hyperlink](#), page number, page count, and table of contents.
- Supports formats like [text](#), [paragraph](#), [bullets and numbering](#), [table](#), [page settings](#), etc.
- Provides support to create, edit, and apply [paragraph and character styles](#).
- Provides support to [find and replace](#) text within the document.
- Supports all the common editing and formatting operations along with [undo and redo](#).
- Provides support to [cut](#), [copy](#), and [paste](#) rich text contents within the component. Also allows pasting simple text to and from other applications.
- Provides support to insert, and edit [form fields](#).
- Provides support to insert, and edit [comments](#).
- Provides support to perform [spell checking](#) for any input text
- Allows user interactions like [zoom](#), [scroll](#), select contents through touch, mouse, and keyboard.
- Provides intuitive UI options like context menu, [dialogs](#), and [navigation pane](#).
- [Localizes](#) all the static text to any desired language.
- Allows to create a lightweight Word viewer using module injection to view and [prints](#) Word documents.
- Provides a [server-side helper library](#) to open the Word documents like DOCX, DOC, WordML, RTF, and Text, by converting it to SFDT file format.

### Supported Web platforms

- [Javascript\(ES5\)](#)
- [Javascript](#)
- [Angular](#)
- [React](#)
- [Vue](#)
- [ASP.NET Core](#)
- [ASP.NET MVC](#)
- [Blazor](#)

### *Supported platforms for server-side dependencies*

You can deploy web APIs for server-side dependencies of Document Editor component in the following platforms.

- [ASP.NET Core](#)
- [ASP.NET MVC](#)
- [Java](#)

To know more about server-side dependencies, refer this [page](#).

### Which operations require server-side interaction

- Open file formats other than SFDT
- Paste with formatting
- Restrict editing
- Spellcheck
- Save as file formats other than SFDT and DOCX

Note: If you don't require the above functionalities then you can deploy as pure client-side component without any server-side interactions.

### Getting Started with the Vue DocumentEditor Component in Vue 2

This article provides a step-by-step guide for setting up a Vue 2 project using [Vue-CLI](#) and integrating the Syncfusion Vue Document Editor component

#### Prerequisites

##### [System requirements for Syncfusion Vue UI components](#)

#### Dependencies

The list of dependencies required to use the Document Editor component in your application is given below:

```
`javascript
|-- @syncfusion/ej2-vue-documenteditor
|-- @syncfusion/ej2-vue-base
|-- @syncfusion/ej2-documenteditor
|-- @syncfusion/ej2-base
|-- @syncfusion/ej2-buttons
|-- @syncfusion/ej2-compression
|-- @syncfusion/ej2-data
|-- @syncfusion/ej2-dropdowns
|-- @syncfusion/ej2-file-utils
|-- @syncfusion/ej2-inputs
|-- @syncfusion/ej2-lists
|-- @syncfusion/ej2-navigations
|-- @syncfusion/ej2-popups
|-- @syncfusion/ej2-splitbuttons
|-- @syncfusion/ej2-charts
`
```

#### *Server side dependencies*

The Document Editor component requires server-side interactions for the following operations:

- [Open file formats other than SFDT](#)
- [Paste with formatting](#)
- [Restrict editing](#)
- [Spellcheck](#)
- [Save as file formats other than SFDT and DOCX](#)

Note: If you don't require the above functionalities then you can deploy as pure client-side component without any server-side interactions.

To know about server-side dependencies, please refer this [page](#).

### Setting up the Vue 2 project

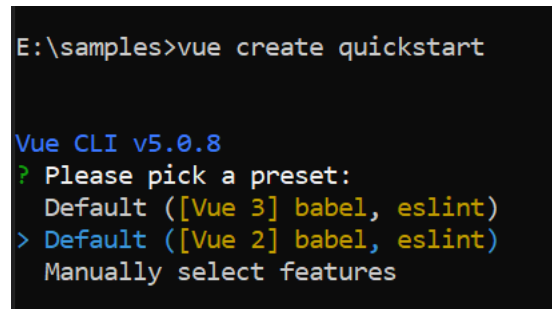
To generate a Vue 2 project using Vue-CLI, use the [vue create](#) command. Follow these steps to install Vue CLI and create a new project:

```
`bash
npm install -g @vue/cli
vue create quickstart
cd quickstart
npm run serve
`
```

or

```
`bash
yarn global add @vue/cli
vue create quickstart
cd quickstart
yarn run serve
`
```

When creating a new project, choose the option `Default ([Vue 2] babel, eslint)` from the menu.



```
E:\samples>vue create quickstart

Vue CLI v5.0.8
? Please pick a preset:
  Default ([Vue 3] babel, eslint)
> Default ([Vue 2] babel, eslint)
  Manually select features
```

Once the `quickstart` project is set up with default settings, proceed to add Syncfusion components to the project.

### Add Syncfusion Vue packages

Syncfusion packages are available at [npmjs.com](https://www.npmjs.com). To use Vue components, install the required npm package.

This article uses the [Vue Document editor component](#) as an example. Install the `@syncfusion/ej2-vue-documenteditor` package by running the following command:

```
`bash
npm install @syncfusion/ej2-vue-documenteditor --save
`
```

or

```
`bash
```

```
yarn add @syncfusion/ej2-vue-documenteditor
```

,

### Import Syncfusion CSS styles

You can import themes for the Syncfusion Vue component in various ways, such as using CSS or SASS styles from npm packages, CDN, [CRG](#) and [Theme Studio](#). Refer to [themes topic](#) to know more about built-in themes and different ways to refer to themes in a Vue project.

In this article, the **Material** theme is applied using CSS styles, which are available in installed packages. The necessary Material CSS styles for the Document Editor component and its dependents were imported into the `<style>` section of **src/App.vue** file.

,

```
@import '../node_modules/@syncfusion/ej2-base/styles/material.css';
@import '../node_modules/@syncfusion/ej2-buttons/styles/material.css';
@import '../node_modules/@syncfusion/ej2-inputs/styles/material.css';
@import '../node_modules/@syncfusion/ej2-popups/styles/material.css';
@import '../node_modules/@syncfusion/ej2-lists/styles/material.css';
@import '../node_modules/@syncfusion/ej2-navigations/styles/material.css';
@import '../node_modules/@syncfusion/ej2-splitbuttons/styles/material.css';
@import '../node_modules/@syncfusion/ej2-dropdowns/styles/material.css';
@import "../node_modules/@syncfusion/ej2-vue-documenteditor/styles/material.css";
```

,

Document editor has different themes, please refer to [Supported themes](#) section.

### Add Syncfusion Vue component

You can add **DocumentEditorContainer** component with predefined toolbar and properties pane options or **DocumentEditor** component with customize options.

Starting from **v19.3.0.x**, we have optimized the accuracy of text size measurements such as to match Microsoft Word pagination for most Word documents. This improvement is included as default behavior along with an optional API [to disable it and retain the document pagination behavior of older versions](#).

### DocumentEditor Component

Document Editor Component is used to create, view and edit word documents. In this, you can customize the UI options based on your requirements to modify the document.

### Registering DocumentEditor Component

You can register the Document Editor component in your application by using the **Vue.use()**.

Refer to the code example given below.

```
`ts
```



```
import { DocumentEditorPlugin } from '@syncfusion/ej2-vue-documenteditor';
Vue.use(DocumentEditorPlugin);
,
```

Registering `DocumentEditorPlugin` in vue, will register the Document Editor component along with its required child directives globally.

#### Adding DocumentEditor Component

Add the Vue Document Editor by using `<ejs-documenteditor>` selector in `<template>` section of the `App.vue` file.

```
,
<template>
<div id="app">
<ejs-documenteditor :serviceUrl='serviceUrl' :isReadOnly='false' :enablePrint='true'
:enableSfdtExport='true' :enableSelection='true' :enableContextMenu='true' :enableSearch='true'
:enableOptionsPane='true' :enableWordExport='true' :enableTextExport='true' :enableEditor='true'
:enableImageResizer='true' :enableEditorHistory='true' :enableHyperlinkDialog='true'
:enableTableDialog='true' :enableBookmarkDialog='true' :enableTableOfContentsDialog='true'
:enablePageSetupDialog='true' :enableStyleDialog='true' :enableListDialog='true'
:enableParagraphDialog='true' :enableFontDialog='true' :enableTablePropertiesDialog='true'
:enableBordersAndShadingDialog='true' :enableTableOptionsDialog='true' height="370px"> </ejs-
documenteditor>
</div>
</template>
<script>
import Vue from 'vue';
import { DocumentEditorPlugin, DocumentEditorComponent, Print, SfdtExport, WordExport, TextExport,
Selection, Search, Editor, ImageResizer, EditorHistory, ContextMenu, OptionsPane, HyperlinkDialog,
TableDialog, BookmarkDialog, TableOfContentsDialog, PageSetupDialog, StyleDialog, ListDialog,
ParagraphDialog, BulletsAndNumberingDialog, FontDialog, TablePropertiesDialog,
BordersAndShadingDialog, TableOptionsDialog, CellOptionsDialog, StylesDialog } from '@syncfusion/ej2-
vue-documenteditor';
Vue.use(DocumentEditorPlugin);
export default {
data () {
return {
serviceUrl:'https://ej2services.syncfusion.com/production/web-services/api/documenteditor/'
},
provide: {
//Inject require modules.
```

DocumentEditor: [Print, SfdtExport, WordExport, TextExport, Selection, Search, Editor, ImageResizer, EditorHistory, ContextMenu, OptionsPane, HyperlinkDialog, TableDialog, BookmarkDialog, TableOfContentsDialog, PageSetupDialog, StyleDialog, ListDialog, ParagraphDialog, BulletsAndNumberingDialog, FontDialog, TablePropertiesDialog, BordersAndShadingDialog, TableOptionsDialog, CellOptionsDialog, StylesDialog]

```
}
}
}
</script>
```

The Document Editor requires server-side interactions for the following operations:

- \* Paste with formatting
- \* Restrict editing
- \* Spell check

Refer to this [link](#) to configure the web service and set the [serviceUrl](#).

Run the DocumentEditor application

The Vue Document Editor application is configured to compile and run the application in a browser. Use the following command to run the application.

```
`bash
npm run dev
```

Output will be displayed as follows.

#### APP.VUE

```
<template>
  <div id="app">
    <ejs-documenteditor id="container6" height="370px" style="width:
100%" :serviceUrl='serviceUrl' :isReadOnly='false' :enablePrint='true'
:enableSfdtExport='true' :enableSelection='true' :enableContextMenu='true'
:enableSearch='true' :enableOptionsPane='true' :enableWordExport='true'
:enableTextExport='true' :enableEditor='true' :enableImageResizer='true'
:enableEditorHistory='true' :enableHyperlinkDialog='true'
:enableTableDialog='true' :enableBookmarkDialog='true'
:enableTableOfContentsDialog='true' :enablePageSetupDialog='true'
:enableStyleDialog='true' :enableListDialog='true'
:enableParagraphDialog='true' :enableFontDialog='true'
:enableTablePropertiesDialog='true' :enableBordersAndShadingDialog='true'
:enableTableOptionsDialog='true'></ejs-documenteditor>
  </div>
</template>
<script>
```

```

import Vue from 'vue'
import { DocumentEditorPlugin, DocumentEditorComponent, Print,
SfdtExport, WordExport, TextExport, Selection, Search, Editor, ImageResizer,
EditorHistory, ContextMenu, OptionsPane, HyperlinkDialog, TableDialog,
BookmarkDialog, TableOfContentsDialog, PageSetupDialog, StyleDialog,
ListDialog, ParagraphDialog, BulletsAndNumberingDialog, FontDialog,
TablePropertiesDialog, BordersAndShadingDialog, TableOptionsDialog,
CellOptionsDialog, StylesDialog } from '@syncfusion/ej2-vue-documenteditor';
Vue.use(DocumentEditorPlugin);
export default {
  data() {
    return {
      serviceUrl: 'https://ej2services.syncfusion.com/production/web-
services/api/documenteditor/'
    };
  },
  provide: {
    //Inject require modules.
    DocumentEditor: [Print, SfdtExport, WordExport, TextExport,
Selection, Search, Editor, ImageResizer, EditorHistory, ContextMenu,
OptionsPane, HyperlinkDialog, TableDialog, BookmarkDialog,
TableOfContentsDialog, PageSetupDialog, StyleDialog, ListDialog,
ParagraphDialog, BulletsAndNumberingDialog, FontDialog,
TablePropertiesDialog, BordersAndShadingDialog, TableOptionsDialog,
CellOptionsDialog, StylesDialog]
  }
}
</script>
<style>
  @import "../../node_modules/@syncfusion/ej2-vue-
documenteditor/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/document-editor/getting-started-cs2" %}

#### [DocumentEditorContainer Component](#)

DocumentEditorContainer Component is also used to create, view and edit word document. But here, you can use predefined toolbar and properties pane to view and modify word document.

#### [Registering DocumentEditorContainer Component](#)

You can register the DocumentEditorContainer component in your application by using the `Vue.use()`.

Refer to the code example given below.

`ts

```
import { DocumentEditorContainerPlugin } from '@syncfusion/ej2-vue-documenteditor';
```

```
Vue.use(DocumentEditorContainerPlugin);
```

,

Registering `DocumentEditorContainerPlugin` in vue, will register the DocumentEditorContainer component along with its required child directives globally.

### Adding DocumentEditorContainer Component

Add the Vue DocumentEditorContainer by using `<ejs-documenteditorcontainer>` selector in `<template>` section of the `App.vue` file.

```
,

<template>
<div id="app">
<ejs-documenteditorcontainer height="590px" :serviceUrl='serviceUrl' :enableToolbar='true'> </ejs-
documenteditorcontainer>
</div>
</template>
<script>
import Vue from 'vue';

import { DocumentEditorContainerPlugin, DocumentEditorContainerComponent,Toolbar } from
'@syncfusion/ej2-vue-documenteditor';
Vue.use(DocumentEditorContainerPlugin);
export default {
data(){
return { serviceUrl:'https://ej2services.syncfusion.com/production/web-services/api/documenteditor/' }
},
provide: {
//Inject require modules.
DocumentEditorContainer: [Toolbar]
}
}
</script>
,
```

### Run the DocumentEditorContainer application

The Vue DocumentEditorContainer application is configured to compile and run the application in a browser. Use the following command to run the application.

```
`bash
npm run dev
,
```

DocumentEditorContainer output will be displayed as follows.

### **APP.VUE**

```
<template>
```

```

<div id="app">
  <ejs-documenteditorcontainer ref='documenteditor'
:serviceUrl='serviceUrl' height="590px" id='container'
:enableToolbar='true'></ejs-documenteditorcontainer>
</div>
</template>
<script>
  import Vue from 'vue';
  import { DocumentEditorContainerPlugin,
DocumentEditorContainerComponent,Toolbar} from '@syncfusion/ej2-vue-
documenteditor';
  Vue.use(DocumentEditorContainerPlugin);
  export default {
    data() {
      return {
serviceUrl:'https://ej2services.syncfusion.com/production/web-
services/api/documenteditor/' };
    },
    provide: {
      //Inject require modules.
      DocumentEditorContainer: [Toolbar]
    }
  }
</script>
<style>
  @import "../../node_modules/@syncfusion/ej2-vue-
documenteditor/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/document-editor/getting-started-cs3" %}

## Frequently Asked Questions

- [How to localize the Documenteditor container.](#)
- [How to load the document by default.](#)
- [How to customize tool bar.](#)
- [How to resize Document editor component.](#)

## Getting Started with Syncfusion Document Editor Component in Vue 3

This section explains how to use Document Editor component in Vue 3 application.

### Prerequisites

#### [System requirements for Syncfusion Vue UI components](#)

#### Creating Vue application using Vue CLI

The easiest way to create a Vue application is to use the [Vue CLI](#). Vue CLI versions above [4.5.0](#) are mandatory for creating applications using Vue 3. Use the following command to uninstall older versions of the Vue CLI.

```
`bash
```

```
npm uninstall vue-cli -g
```

```
,
```

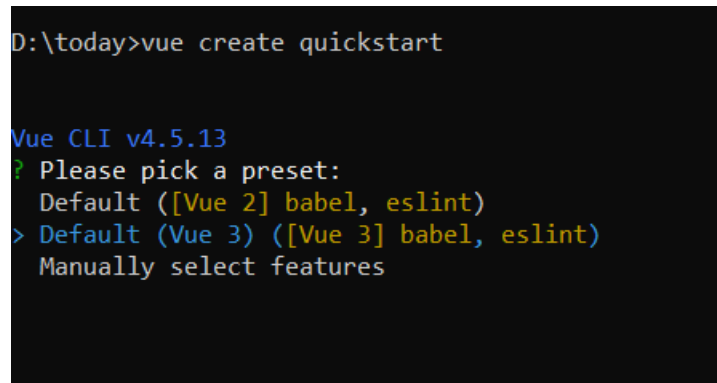
Use the following commands to install the latest version of Vue CLI.

```
`bash
npm install -g @vue/cli
npm install -g @vue/cli-init
`
```

Create a new project using the command below.

```
`bash
vue create quickstart
`
```

Initiating a new project prompts us to choose the type of project to be used for the current application. Select the option **Default (Vue 3)** from the menu.



```
D:\today>vue create quickstart

Vue CLI v4.5.13
? Please pick a preset:
  Default ([Vue 2] babel, eslint)
> Default (Vue 3) ([Vue 3] babel, eslint)
  Manually select features
```

### Adding Syncfusion DocumentEditor package in the application

Syncfusion Vue packages are maintained in the [npmjs.com](https://www.npmjs.com) registry.

The Document Editor component will be used in this example. To install it use the following command.

```
`bash
npm install @syncfusion/ej2-vue-documenteditor --save
`
```

The **--save** will instruct NPM to include the Document Editor package inside the **dependencies** section of the **package.json**.

### Adding CSS reference

Add Document editor component's styles as given below in **<style>** section of the **src/App.vue** file.

```
`
<style>

@import '../node_modules/@syncfusion/ej2-base/styles/material.css';
@import '../node_modules/@syncfusion/ej2-buttons/styles/material.css';
@import '../node_modules/@syncfusion/ej2-inputs/styles/material.css';

`
```

```

@import '../node_modules/@syncfusion/ej2-popups/styles/material.css';
@import '../node_modules/@syncfusion/ej2-lists/styles/material.css';
@import '../node_modules/@syncfusion/ej2-navigations/styles/material.css';
@import '../node_modules/@syncfusion/ej2-splitbuttons/styles/material.css';
@import '../node_modules/@syncfusion/ej2-dropdowns/styles/material.css';
@import "../node_modules/@syncfusion/ej2-vue-documenteditor/styles/material.css";
</style>
`

```

Document editor has different themes, please refer to [supported themes](#) section.

### Adding Component

Starting from v19.3.0.x, we have optimized the accuracy of text size measurements such as to match Microsoft Word pagination for most Word documents. This improvement is included as default behavior along with an optional API [to disable it and retain the document pagination behavior of older versions](#).

#### Adding DocumentEditor component in the application

Document Editor Component is used to create, view and edit word documents. In this, you can customize the UI options based on your requirements to modify the document.

You have completed all the necessary configurations needed for rendering the Syncfusion Vue component. Now, you are going to add the Document Editor component using following steps.

**Step 1:** Import the Document Editor component in the `<script>` section of the `src/App.vue` file.

```

<script>

import { DocumentEditorComponent, Print, SfdtExport, WordExport, TextExport, Selection, Search,
Editor, ImageResizer, EditorHistory, ContextMenu, OptionsPane, HyperlinkDialog, TableDialog,
BookmarkDialog, TableOfContentsDialog, PageSetupDialog, StyleDialog, ListDialog, ParagraphDialog,
BulletsAndNumberingDialog, FontDialog, TablePropertiesDialog, BordersAndShadingDialog,
TableOptionsDialog, CellOptionsDialog, StylesDialog } from '@syncfusion/ej2-vue-documenteditor';

</script>
`

```

**Step 2:** Register the Document Editor Container component in your application.

```

`js
import { DocumentEditorComponent } from '@syncfusion/ej2-vue-documenteditor';

//Component registration
export default {
  name: "App",
  components: {
    'ejs-documenteditor' : DocumentEditorComponent

```

```

}
}
,

```

In the above code snippet, you have registered DocumentEditorContainer.

**Step 3:** Add the component definition in template section.

```

,

```

```

<template>

<ejs-documenteditor :serviceUrl='serviceUrl' :isReadOnly='false' :enablePrint='true'
:enableSfdtExport='true' :enableSelection='true' :enableContextMenu='true' :enableSearch='true'
:enableOptionsPane='true' :enableWordExport='true' :enableTextExport='true' :enableEditor='true'
:enableImageResizer='true' :enableEditorHistory='true' :enableHyperlinkDialog='true'
:enableTableDialog='true' :enableBookmarkDialog='true' :enableTableOfContentsDialog='true'
:enablePageSetupDialog='true' :enableStyleDialog='true' :enableListDialog='true'
:enableParagraphDialog='true' :enableFontDialog='true' :enableTablePropertiesDialog='true'
:enableBordersAndShadingDialog='true' :enableTableOptionsDialog='true'> </ejs-documenteditor>

</template>
,

```

**Step 4:** Declare the bound properties `serviceUrl` in the `script` section.

```

`js
data () {
return {
serviceUrl:'https://ej2services.syncfusion.com/production/web-services/api/documenteditor/'
}
},
provide: {
DocumentEditor: [Print, SfdtExport, WordExport, TextExport, Selection, Search, Editor, ImageResizer,
EditorHistory, ContextMenu, OptionsPane, HyperlinkDialog, TableDialog, BookmarkDialog,
TableOfContentsDialog, PageSetupDialog, StyleDialog, ListDialog, ParagraphDialog,
BulletsAndNumberingDialog, FontDialog, TablePropertiesDialog, BordersAndShadingDialog,
TableOptionsDialog, CellOptionsDialog, StylesDialog]
}
,

```

**Step 5:** Summarizing the above steps, update the `src/App.vue` file with following code.

```

,

```

```

<template>

<ejs-documenteditor :serviceUrl='serviceUrl' :isReadOnly='false' :enablePrint='true'
:enableSfdtExport='true' :enableSelection='true' :enableContextMenu='true' :enableSearch='true'

```



```

:enableOptionsPane='true' :enableWordExport='true' :enableTextExport='true' :enableEditor='true'
:enableImageResizer='true' :enableEditorHistory='true' :enableHyperlinkDialog='true'
:enableTableDialog='true' :enableBookmarkDialog='true' :enableTableOfContentsDialog='true'
:enablePageSetupDialog='true' :enableStyleDialog='true' :enableListDialog='true'
:enableParagraphDialog='true' :enableFontDialog='true' :enableTablePropertiesDialog='true'
:enableBordersAndShadingDialog='true' :enableTableOptionsDialog='true'> </ejs-documenteditor>
</template>

<script>

import { DocumentEditorComponent, Print, SfdtExport, WordExport, TextExport, Selection, Search,
Editor, ImageResizer, EditorHistory, ContextMenu, OptionsPane, HyperlinkDialog, TableDialog,
BookmarkDialog, TableOfContentsDialog, PageSetupDialog, StyleDialog, ListDialog, ParagraphDialog,
BulletsAndNumberingDialog, FontDialog, TablePropertiesDialog, BordersAndShadingDialog,
TableOptionsDialog, CellOptionsDialog, StylesDialog } from '@syncfusion/ej2-vue-documenteditor';

//Component registration
export default {
name: 'App',
components: {
// Declaring component
'ejs-documenteditor' : DocumentEditorComponent
},
data () {
return {
serviceUrl:'https://ej2services.syncfusion.com/production/web-services/api/documenteditor/'
};
},
provide: {
DocumentEditor: [Print, SfdtExport, WordExport, TextExport, Selection, Search, Editor, ImageResizer,
EditorHistory, ContextMenu, OptionsPane, HyperlinkDialog, TableDialog, BookmarkDialog,
TableOfContentsDialog, PageSetupDialog, StyleDialog, ListDialog, ParagraphDialog,
BulletsAndNumberingDialog, FontDialog, TablePropertiesDialog, BordersAndShadingDialog,
TableOptionsDialog, CellOptionsDialog, StylesDialog]
}
}
</script>

<style>

@import '../node_modules/@syncfusion/ej2-base/styles/material.css';
@import '../node_modules/@syncfusion/ej2-buttons/styles/material.css';

```

```
@import '../node_modules/@syncfusion/ej2-inputs/styles/material.css';
@import '../node_modules/@syncfusion/ej2-popups/styles/material.css';
@import '../node_modules/@syncfusion/ej2-lists/styles/material.css';
@import '../node_modules/@syncfusion/ej2-navigations/styles/material.css';
@import '../node_modules/@syncfusion/ej2-splitbuttons/styles/material.css';
@import '../node_modules/@syncfusion/ej2-dropdowns/styles/material.css';
@import "../node_modules/@syncfusion/ej2-vue-documenteditor/styles/material.css";
</style>
`
```

#### Running the DocumentEditor application

Run the application using the following command.

```
`bash
npm run serve
`
```

Web server will be initiated, Open the quick start app in the browser at port `localhost:8080`.



#### Adding DocumentEditorContainer component in the application

Document Editor Container Component is also used to create, view and edit word document. But here, you can use predefined toolbar and properties pane to view and modify word document.

You have completed all the necessary configurations needed for rendering the Syncfusion Vue component. Now, you are going to add the DocumentEditorContainer component using following steps.

**Step 1:** Import the DocumentEditorContainer component in the `<script>` section of the `src/App.vue` file.

```
<script>
import { DocumentEditorContainerComponent, Toolbar } from '@syncfusion/ej2-vue-documenteditor';
```

```
</script>
```

**Step 2:** Register the DocumentEditorContainer component in your application.

```
`js
import { DocumentEditorContainerComponent, Toolbar } from '@syncfusion/ej2-vue-documenteditor';
//Component registration
export default {
  name: "App",
  components: {
    'ejs-documenteditorcontainer' : DocumentEditorContainerComponent
  }
}
```

In the above code snippet, you have registered DocumentEditorContainer.

**Step 3:** Add the component definition in template section.

```
<template>
<ejs-documenteditorcontainer :serviceUrl='serviceUrl' :enableToolbar='true'> </ejs-
documenteditorcontainer>
</template>
```

**Step 4:** Declare the bound properties `serviceUrl` in the `script` section.

```
`js
data () {
  return {
    serviceUrl:'https://ej2services.syncfusion.com/production/web-services/api/documenteditor/'
  };
},
provide: {
  DocumentEditorContainer: [Toolbar]
}
```

**Step 5:** Summarizing the above steps, update the `src/App.vue` file with following code.

```

,

<template>
<ejs-documenteditorcontainer :serviceUrl='serviceUrl' :enableToolbar='true'> </ejs-
documenteditorcontainer>
</template>
<script>
import { DocumentEditorContainerComponent, Toolbar } from '@syncfusion/ej2-vue-documenteditor';
//Component registration
export default {
name: 'App',
components: {
// Declaring component
'ejs-documenteditorcontainer' : DocumentEditorContainerComponent
},
data () {
return {
serviceUrl:'https://ej2services.syncfusion.com/production/web-services/api/documenteditor/'
};
},
provide: {
DocumentEditorContainer: [Toolbar]
}
}
</script>
<style>
@import '../node_modules/@syncfusion/ej2-base/styles/material.css';
@import '../node_modules/@syncfusion/ej2-buttons/styles/material.css';
@import '../node_modules/@syncfusion/ej2-inputs/styles/material.css';
@import '../node_modules/@syncfusion/ej2-popups/styles/material.css';
@import '../node_modules/@syncfusion/ej2-lists/styles/material.css';
@import '../node_modules/@syncfusion/ej2-navigations/styles/material.css';
@import '../node_modules/@syncfusion/ej2-splitbuttons/styles/material.css';
@import '../node_modules/@syncfusion/ej2-dropdowns/styles/material.css';

```

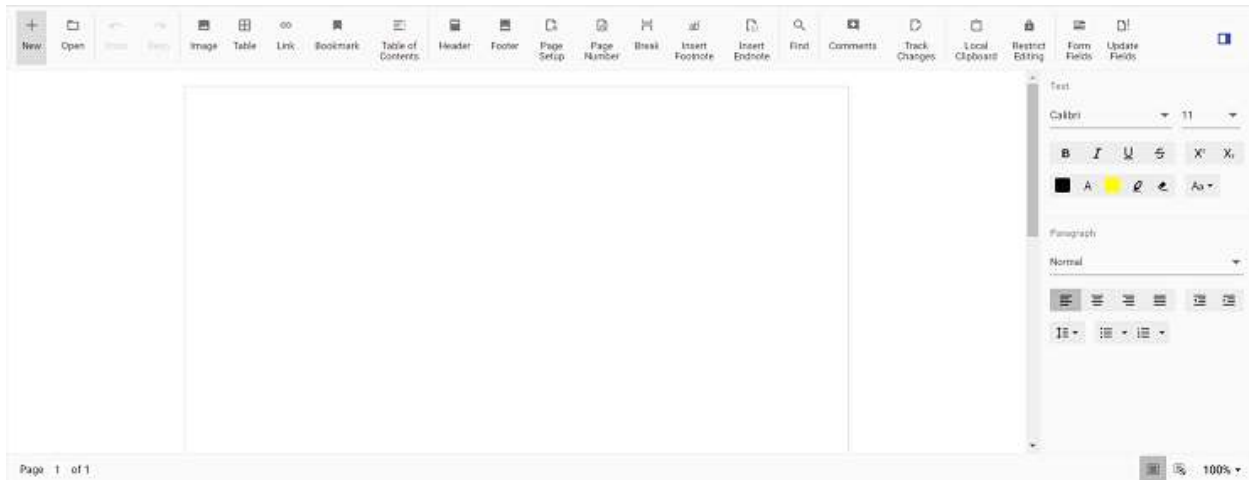
```
@import "../node_modules/@syncfusion/ej2-vue-documenteditor/styles/material.css";
</style>
,
```

### Running the DocumentEditorContainer application

Run the application using the following command.

```
`bash
npm run serve
,
```

Web server will be initiated, Open the quick start app in the browser at port [localhost:8080](http://localhost:8080).



### Feature module in Vue Document editor component

Document Editor features are segregated into individual feature-wise modules to enable selective referencing. By default, the Document Editor displays the document in read-only mode. The required modules should be injected to extend its functionality. The following are the selective modules of Document Editor that can be included as required:

- **Print** - Prints the document.
- **SfdtExport** - Exports the document as Syncfusion Document Text (.SFDT) file.
- **Selection** - Selects a portion of the document and copy it to the clipboard.
- **Search** - Searches specific text and navigate between the results.
- **WordExport** - Exports the document as Word Document (.DOCX) file.
- **TextExport** - Exports the document as Text Document (.TXT) file.
- **Editor** - Performs all kind of editing operations.
- **EditorHistory** - Maintains the history of editing operations so that you can perform undo and redo at any time.
- User interface options such as context menu, options pane, image resizer, and dialog are available as individual modules.

In addition to injecting the required modules in your application, enable corresponding properties to extend the functionality for a Document Editor instance.

Refer to the following table.

| Module | Dependent modules to be injected for extending the functionality of Document Editor in your application | Property to enable the functionality for a Document Editor instance |

|---|---|---|

|Print|Print|<ejs-documenteditor :enablePrint='true'></ejs-documenteditor>|

|SfdtExport|SfdtExport|<ejs-documenteditor :enableSfdtExport='true'></ejs-documenteditor>|

|Selection|Selection|<ejs-documenteditor :enableSelection='true'></ejs-documenteditor>|

|Search|Selection, Search|<ejs-documenteditor :enableSearch='true'></ejs-documenteditor>|

|WordExport|SfdtExport, WordExport|<ejs-documenteditor :enableWordExport='true'></ejs-documenteditor>|

|TextExport|SfdtExport, TextExport|<ejs-documenteditor :enableTextExport='true'></ejs-documenteditor>|

|Editor|Selection, Editor|<ejs-documenteditor :isReadOnly='false' :enableEditor='true'></ejs-documenteditor>|

|EditorHistory|Selection, Editor, EditorHistory|<ejs-documenteditor :isReadOnly='false' :enableEditor='true' :enableEditorHistory='true'></ejs-documenteditor>|

|OptionsPane(Find)|Selection, Search, OptionsPane|<ejs-documenteditor :enableSearch='true' :enableOptionsPane='true'></ejs-documenteditor>|

|OptionsPane(Find and Replace)|Selection, Search, Editor, OptionsPane|<ejs-documenteditor :isReadOnly='false' :enableEditor='true' :enableSearch='true' :enableOptionsPane='true'></ejs-documenteditor>|

|ContextMenu|Selection, ContextMenu|<ejs-documenteditor :enableSelection='true' :enableContextMenu='true'></ejs-documenteditor>|

|ImageResizer|Selection, Editor, ImageResizer|<ejs-documenteditor :isReadOnly='false' :enableEditor='true' :enableImageResizer='true'></ejs-documenteditor>|

|HyperlinkDialog|Selection, Editor, HyperlinkDialog|<ejs-documenteditor :isReadOnly='false' :enableEditor='true' :enableHyperlinkDialog='true'></ejs-documenteditor>|

|TableDialog|Selection, Editor, TableDialog|<ejs-documenteditor :isReadOnly='false' :enableEditor='true' :enableTableDialog='true'></ejs-documenteditor>|

|FontDialog|Selection, Editor, FontDialog|<ejs-documenteditor :isReadOnly='false' :enableEditor='true' :enableFontDialog='true'></ejs-documenteditor>|

|ParagraphDialog|Selection, Editor, ParagraphDialog|<ejs-documenteditor :isReadOnly='false' :enableEditor='true' :enableParagraphDialog='true'></ejs-documenteditor>|

|BookmarkDialog|Selection, Editor, BookmarkDialog|<ejs-documenteditor :isReadOnly='false' :enableEditor='true' :enableBookmarkDialog='true'></ejs-documenteditor>|

|PageSetupDialog|Selection, Editor, PageSetupDialog|<ejs-documenteditor :isReadOnly='false' :enableEditor='true' :enablePageSetupDialog='true'></ejs-documenteditor>|

```
|TableOfContentsDialog|Selection, Editor, TableOfContentsDialog|<ejs-documenteditor
:isReadOnly='false' :enableEditor='true' :enableTableOfContentsDialog='true'></ejs-
documenteditor>|
```

```
|ListDialog|Selection, Editor, ListDialog|<ejs-documenteditor :isReadOnly='false'
:enableEditor='true' :enableListDialog='true'></ejs-documenteditor>|
```

```
|TablePropertiesDialog|Selection, Editor, TablePropertiesDialog|<ejs-documenteditor
:isReadOnly='false' :enableEditor='true' :enableTablePropertiesDialog='true'></ejs-
documenteditor>|
```

```
|CellOptionsDialog|Selection, Editor, CellOptionsDialog|<ejs-documenteditor :isReadOnly='false'
:enableEditor='true' :enableTablePropertiesDialog='true'></ejs-documenteditor>|
```

```
|BordersAndShadingDialog|Selection, Editor, BordersAndShadingDialog|<ejs-documenteditor
:isReadOnly='false' :enableEditor='true' :enableBordersAndShadingDialog='true'></ejs-
documenteditor>|
```

```
|TableOptionsDialog|Selection, Editor, TableOptionsDialog|<ejs-documenteditor
:isReadOnly='false' :enableEditor='true' :enableTableOptionsDialog='true'></ejs-
documenteditor>|
```

```
|StylesDialog|Selection, Editor, StylesDialog,StyleDialog|<ejs-documenteditor
:isReadOnly='false' :enableEditor='true' :enableStyleDialog='true'
:enableStylesDialog='true'></ejs-documenteditor>|
```

```
|StyleDialog|Selection, Editor, StyleDialog|<ejs-documenteditor :isReadOnly='false'
:enableEditor='true' :enableStyleDialog='true'></ejs-documenteditor>|
```

```
|BulletsAndNumberingDialog|Selection, Editor, BulletsAndNumberingDialog|<ejs-documenteditor
:isReadOnly='false' :enableEditor='true' :enableStyleDialog='true'></ejs-documenteditor>|
```

These modules should be injected into the `provide` section and use `DocumentEditor` as a key of the object.

### Import in Vue Document editor component

In Document Editor, the documents are stored in its own format called **Syncfusion Document Text (SFDT)**.

The following example shows how to open SFDT data in Document Editor.

#### APP.VUE

```
<template>
  <div id="app">
    <ejs-documenteditor ref="documenteditor" id="container_1"
height="370px" style="width: 100%;display:block" ></ejs-documenteditor>
  </div>
</template>
<script>
  import Vue from 'vue'
  import { DocumentEditorPlugin } from '@syncfusion/ej2-vue-
documenteditor';
  Vue.use(DocumentEditorPlugin);
```

```

export default {
  data: function() {
    return {
    };
  },
  mounted: function() {
    let sfdt: string = `{
      "sections": [
        {
          "blocks": [
            {
              "inlines": [
                {
                  "characterFormat": {
                    "bold": true,
                    "italic": true
                  },
                  "text": "Hello World"
                }
              ]
            }
          ]
        },
        "headersFooters": {
        }
      ]
    }`;
    //Open the default document in Document Editor.
    this.$refs.documenteditor.open(sfdt);
  }
}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-vue-
documenteditor/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/document-editor/import-cs1" %}

### Import document from local machine

The following example shows how to import document from local machine.

#### APP.VUE

```

<template>
  <div id="app">
    <input type="file" ref="fileUpload" v-on:change="onFileUpload"
    accept=".sfdt" style="position:fixed; left:-100em" />
    <div>
      <button v-on:click='openFileButtonClickHandler'
      >Import</button>
    </div>
    <ejs-documenteditor ref="documenteditor" height="370px"
    style="width: 100%;display:block" ></ejs-documenteditor>
  </div>
</template>

```



```
<script>
  import Vue from 'vue'
  import { DocumentEditorPlugin } from '@syncfusion/ej2-vue-
documenteditor';
  Vue.use(DocumentEditorPlugin);
  export default {
    data: function() {
      return {
      };
    },
    methods: {
      openFileButtonClickHandler: function() {
        this.$refs.fileUpload.click();
      },
      onFileUpload: function(e) {
        if (e.target.files[0]) {
          let file = e.target.files[0];
          if (file.name.substr(file.name.lastIndexOf('.')) ===
'.sfdt') {
            let fileReader: FileReader = new FileReader();
            fileReader.onload = (e: any) => {
              let contents: string = e.target.result;
              //Open the default document in Document
Editor.

              this.$refs.documenteditor.open(contents);
            };
            fileReader.readAsText(file);
          }
        }
      }
    }
  }
</script>
<style>
  @import "../node_modules/@syncfusion/ej2-vue-
documenteditor/styles/material.css";
</style>
```

{% previewsample "page.domainurl/code-snippet/document-editor/import-sfdt-cs1" %}

### Convert word documents into SFDT

You can convert word documents into SFDT format using the .NET Standard library

[Syncfusion.EJ2.WordEditor.AspNet.Core](#) by the web API service implementation. This library helps you to convert word documents (.dotx,.docx,.docm,.dot,.doc), rich text format documents (.rtf), and text documents (.txt) into SFDT format.

Note: The Syncfusion Document Editor component's document pagination (page-by-page display) can't be guaranteed for all the Word documents to match the pagination of Microsoft Word application. For more information about [why the document pagination \(page-by-page display\) differs from Microsoft Word](#)

Please refer the following example for converting word documents into SFDT.

,

<template>

```

<input type="file" ref="fileUpload" v-on:change="onFileUpload"
accept=".dotx,.docx,.docm,.dot,.doc,.rtf,.txt,.xml,.sfdt" style="position:fixed; left:-100em" />
<div>
<div>
<button v-on:click='openFileButtonClickHandler'>Import</button>
</div>
<ejs-documenteditor ref="documenteditor" height="370px" style="width: 100%;display:block"></ejs-
documenteditor>
</div>
</template>
<script>
import Vue from 'vue'
import { DocumentEditorPlugin } from '@syncfusion/ej2-vue-documenteditor';
Vue.use(DocumentEditorPlugin);
export default {
data: function() {
return {
};
},
methods: {
loadFile: function(file: File): void {
let ajax: XMLHttpRequest = new XMLHttpRequest();
ajax.open('POST', 'https://localhost:4000/api/documenteditor/Import', true);
ajax.onreadystatechange = () => {
if (ajax.readyState === 4) {
if (ajax.status === 200 || ajax.status === 304) {
// open SFDT text in document editor
this.$refs.documenteditor.open(ajax.responseText);
}
}
}
let formData: FormData = new FormData();
formData.append('files', file);
ajax.send(formData);

```

```

},
openFileButtonClickHandler: function() {
this.$refs.fileUpload.click();
},
onFileUpload: function(e) {
if (e.target.files[0]) {
let file = e.target.files[0];
if (file.name.substr(file.name.lastIndexOf('.')) === '.sfdt') {
let fileReader: FileReader = new FileReader();
fileReader.onload = (e: any) => {
let contents: string = e.target.result;
this.$refs.documenteditor.open(contents);
};
fileReader.readAsText(file);
}
}
}
}
}
</script>
<style>
@import "../../node_modules/@syncfusion/ej2-vue-documenteditor/styles/material.css";
</style>
`

```

Here's how to handle the server-side action for converting word document in to SFDT.

```

`c#
[AcceptVerbs("Post")]
public string Import(Microsoft.AspNetCore.Http.IFormCollection data)
{
if (data.Files.Count == 0)
return null;

System.IO.Stream stream = new System.IO.MemoryStream();
Microsoft.AspNetCore.Http.IFormFile file = data.Files[0];

```

```

int index = file.FileName.LastIndexOf('.');
string type = index > -1 && index < file.FileName.Length - 1 ?
file.FileName.Substring(index) : ".docx";
file.CopyTo(stream);
stream.Position = 0;
Syncfusion.EJ2.DocumentEditor.WordDocument document =
Syncfusion.EJ2.DocumentEditor.WordDocument.Load(stream, GetFormatType(type.ToLower()));
string sfdt = Newtonsoft.Json.JsonConvert.SerializeObject(document);
document.Dispose();
return sfdt;
}

internal static Syncfusion.EJ2.DocumentEditor.FormatType GetFormatType(string format)
{
if (string.IsNullOrEmpty(format))
throw new System.NotSupportedException("EJ2 DocumentEditor does not support this file format.");
switch (format.ToLower()) {
case ".dotx":
case ".docx":
case ".docm":
case ".dotm":
return Syncfusion.EJ2.DocumentEditor.FormatType.Docx;
case ".dot":
case ".doc":
return Syncfusion.EJ2.DocumentEditor.FormatType.Doc;
case ".rtf":
return Syncfusion.EJ2.DocumentEditor.FormatType.Rtf;
case ".txt":
return Syncfusion.EJ2.DocumentEditor.FormatType.Txt;
case ".xml":
return Syncfusion.EJ2.DocumentEditor.FormatType.WordML;
default:
throw new System.NotSupportedException("EJ2 DocumentEditor does not support this file format.");
}
}

```

```
}
,
```

To know about server-side action, please refer this [page](#).

### Compatibility with Microsoft Word

Syncfusion Document Editor is a minimal viable Word document viewer/editor product for web applications. As most compatible Word editor, the product vision is adding valuable feature sets of Microsoft Word, and not to cover 100% feature sets of Microsoft Word desktop application. You can even see the feature sets difference between Microsoft Word desktop and their Word online application. So kindly don't misunderstand this component as a complete replacement for Microsoft Word desktop application and expect 100% feature sets of it.

### How Syncfusion accepts the feature request for Document Editor

Syncfusion accepts new feature request as valid based on feature value and technological feasibility, then plan to implement unsupported features incrementally in future releases in a phase-by-phase manner.

### How to report the problems in Document Editor

You can report the problems with displaying, or editing Word documents in Document Editor component through [support forum](#), [Direct-Trac](#), or [feedback portal](#). Kindly share the Word document for replicating the problem easily in minimal time. If you have confidential data, you can replace it and attach the document.

### Why the document pagination differs from Microsoft Word

For your understanding about the Word document structure and the workflow of Word viewer/editor components, the Word document is a flow document in which content will not be preserved page by page; instead, the content will be preserved sequentially like a HTML file. Only the Word viewer/editor paginates the content of the Word document page by page dynamically, when opened for viewing or editing and this page-wise position information will not be preserved in the document level (it is Word file format specification standard). Syncfusion Document Editor component also does the same.

At present there is a known technical limitation related to slight difference in text size calculated using HTML element based text measuring approach. Even though the text size is calculated with correct font and font size values, the difference lies; it is as low as 0.00XX to 0. XXXX values compared to that of Microsoft Word application's display. Hence the document pagination (page-by-page display) can't be guaranteed for all the Word documents to match the pagination of Microsoft Word application.

### How Syncfusion address the document pagination difference compared to Microsoft Word

The following table illustrates the reasons for pagination (page-by-page display) difference compared to Microsoft Word in your documents and how Syncfusion address it.

Root causes	How is it solved?
Any mistake (wrong behavior handled) in lay outing the supported elements and formatting	Customer can report to Syncfusion support and track the status through bug report link. Syncfusion fixes the bugs in next possible weekly patch release and service pack or main releases.
Font missing in deployment environment	Customer can either report to Syncfusion support and get suggestion or solve it on their own by installing the missing fonts in their deployment environment.

|Any unsupported elements or formatting present in your document |Customer can report to Syncfusion support and track the status through feature request link. Syncfusion implements unsupported features incrementally in future releases based on feature importance, customer interest, efforts involved, and technological feasibility. Also, suggests alternate approach for possible cases. |

|Technical limitation related to framework For example, there is a known case with slight fractional difference in text size measured using HTML and Microsoft Word's display. |Customer can report to Syncfusion support and track the status through feature request link. Syncfusion does research about alternate approaches to overcome the technical limitation/behaviors and process it same as a feature. >Note: Here the challenge is, time schedule for implementation varies based on the alternate solution and its reliability. |

See Also

- [Feature modules](#)

## Server side export in Vue Document editor component

### SFDT to DOCX export

Document Editor supports server-side export of **Syncfusion Document Text (.sfdt)** to Doc, DOCX, RTF, Txt, WordML, HTML formats using server-side helper **Syncfusion.EJ2.DocumentEditor** available in ASP.NET Core & ASP.NET MVC platform in the below NuGet's.

- [Syncfusion.EJ2.WordEditor.AspNet.Core](#)
- [Syncfusion.EJ2.WordEditor.AspNet.Mvc5](#)
- [Syncfusion.EJ2.WordEditor.AspNet.Mvc4](#)

Please refer the following code example.

```
`c#
//API controller for the conversion.
[HttpPost]
public void ExportSFDT([FromBody]SaveParameter data)
{
    Stream document = WordDocument.Save(data.content, FormatType.Docx);
    FileStream file = new FileStream("sample.docx", FileMode.OpenOrCreate, FileAccess.ReadWrite);
    document.CopyTo(file);
    file.Close();
    document.Close();
}

public class SaveParameter
{
    public string content { get; set; }
```

```
}
,
```

Please refer the client side example to serialize the sfdt and send to the server.

```
,
```

```
<template>
<div id="app">
<div>
<button v-on:click='exportBlob' >Save</button>
</div>
<ejs-documenteditor ref="documenteditor" :enableSfdtExport='true' :enableWordExport='true'
:enableSelection='true' :enableEditor='true' :isReadOnly='false' height="370px" style="width:
100%;"></ejs-documenteditor>
</div>
</template>
<script>
import Vue from 'vue'

import { DocumentEditorPlugin, Selection, Editor, SfdtExport, WordExport } from '@syncfusion/ej2-vue-
documenteditor';

Vue.use(DocumentEditorPlugin);

export default {
data: function() {
return {
};
},
provide: {
//Inject require modules.
DocumentEditor : [SfdtExport, WordExport, Selection, Editor]
}
methods: {
exportBlob: function() {
let http: XMLHttpRequest = new XMLHttpRequest();
http.open('POST', 'http://localhost:5000/api/documenteditor/ExportSFDT');
http.setRequestHeader('Content-Type', 'application/json;charset=UTF-8');
http.responseType = 'json';
```

```
//Serialize document content as SFDT.
let sfdt: any = { content: this.$refs.documenteditor.serialize() };
//Send the sfdt content to server side.
http.send(JSON.stringify(sfdt));
}
}
}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-vue-documenteditor/styles/material.css";
</style>
`
```

DocumentEditor object is available in DocumentEditorContainer component(DocumentEditor packaged with toolbar, status bar & properties pane) as [documentEditor](#)

### Export in Vue Document editor component

Document Editor exports the document into various known file formats in client-side such as Microsoft Word document (.docx), text document (.txt), and its own format called **Syncfusion Document Text (.sfdt)**.

We are providing two types of save APIs as mentioned below.

API name	Purpose
-----	-----
save(filename,FormatType):void	 FormatType: Sfdt or Docx or Txt   Creates the document with specified file name and format type. Then, the created file is downloaded in the client browser by default.
saveAsBlob(FormatType):Blob	Creates the document in specified format type and returns the created document as Blob. This blob can be uploaded to your required server, database, or file path.

### SFDT export

The following example shows how to export documents in Document Editor as Syncfusion document text (.sfdt).

### APP.VUE

```
<template>
  <div id="app">
    <div>
      <button v-on:click='saveAsSfdt' >Save</button>
    </div>
    <ejs-documenteditor ref="documenteditor"
:enableSfdtExport='true' :enableSelection='true' :enableEditor='true'
:isReadOnly='false' height="370px" style="width: 100%;"></ejs-
documenteditor>
  </div>
```



```

</template>
<script>
  import Vue from 'vue'
  import { DocumentEditorPlugin, Selection, Editor, SfdtExport } from
  '@syncfusion/ej2-vue-documenteditor';
  Vue.use(DocumentEditorPlugin);
  export default {
    data: function() {
      return {
      };
    },
    provide: {
      //Inject require modules.
      DocumentEditor : [SfdtExport, Selection, Editor]
    }
    methods: {
      saveAsSfdt: function() {
        //Download the current document in sfdt format.
        this.$refs.documenteditor.save('sample', 'Sfdt');
      }
    }
  }
</script>
<style>
  @import "../node_modules/@syncfusion/ej2-vue-
  documenteditor/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/document-editor/export-cs1" %}

### Word export

The following example shows how to export the document as Word document (.docx).

Note: The Syncfusion Document Editor component's document pagination (page-by-page display) can't be guaranteed for all the Word documents to match the pagination of Microsoft Word application. For more information about [why the document pagination \(page-by-page display\) differs from Microsoft Word](#)

### APP.VUE

```

<template>
  <div id="app">
    <div>
      <button v-on:click='saveAsDocx' >Save</button>
    </div>
    <ejs-documenteditor ref="documenteditor"
    :enableSfdtExport='true' :enableWordExport='true' :enableSelection='true'
    :enableEditor='true' :isReadOnly='false' height="370px" style="width:
    100%;"></ejs-documenteditor>
  </div>
</template>
<script>
  import Vue from 'vue'
  import { DocumentEditorPlugin, Selection, Editor, SfdtExport,
  WordExport } from '@syncfusion/ej2-vue-documenteditor';
  Vue.use(DocumentEditorPlugin);

```

```

    export default {
      data: function() {
        return {
        };
      },
      provide: {
        //Inject require modules.
        DocumentEditor : [SfdtExport, WordExport, Selection, Editor]
      },
      methods: {
        saveAsDocx: function() {
          //Download the document in docx format.
          this.$refs.documenteditor.save('sample', 'Docx');
        }
      }
    }
  </script>
  <style>
    @import "../node_modules/@syncfusion/ej2-vue-
    documenteditor/styles/material.css";
  </style>

```

{% previewsample "page.domainurl/code-snippet/document-editor/export-cs2" %}

### Text export

The following example shows how to export document as text document (.txt).

### APP.VUE

```

<template>
  <div id="app">
    <div>
      <button v-on:click='saveAsTextDocument' >Save</button>
    </div>
    <ejs-documenteditor ref="documenteditor"
    :enableSfdtExport='true' :enableTextExport='true' :enableSelection='true'
    :enableEditor='true' :isReadOnly='false' height="370px" style="width:
    100%;"></ejs-documenteditor>
    </div>
  </template>
  <script>
    import Vue from 'vue'
    import { DocumentEditorPlugin, Selection, Editor, SfdtExport,
    TextExport } from '@syncfusion/ej2-vue-documenteditor';
    Vue.use(DocumentEditorPlugin);
    export default {
      data: function() {
        return {
        };
      },
      provide: {
        //Inject require modules.
        DocumentEditor : [SfdtExport, TextExport, Selection, Editor]
      },
      methods: {
        saveAsTextDocument: function() {

```

```

        //Download the current document as txt file.
        this.$refs.documenteditor.save('sample', 'Txt');
    }
}
}
</script>
<style>
    @import "../node_modules/@syncfusion/ej2-vue-
documenteditor/styles/material.css";
</style>

```

```
{% previewsample "page.domainurl/code-snippet/document-editor/export-cs3" %}
```

### Export as blob

Document Editor also supports API to store the document into a blob. Refer to the following sample to export document into blob in client-side.

```

<template>
<div id="app">
<div>
<button v-on:click='exportBlob' >Save</button>
</div>
<ejs-documenteditor ref="documenteditor" :enableSfdtExport='true' :enableWordExport='true'
:enableSelection='true' :enableEditor='true' :isReadOnly='false' height="370px" style="width:
100%;"></ejs-documenteditor>
</div>
</template>
<script>
import Vue from 'vue'
import { DocumentEditorPlugin, Selection, Editor, SfdtExport, WordExport } from '@syncfusion/ej2-vue-
documenteditor';
Vue.use(DocumentEditorPlugin);
export default {
data: function() {
return {
};
},
provide: {
//Inject require modules.
DocumentEditor : [SfdtExport, WordExport, Selection, Editor]

```

```

}
methods: {
  exportBlob: function() {
    //Export the document as Blob object.
    this.$refs.documenteditor.saveAsBlob('Docx').then((exportedDocument: Blob) => {
      // The blob can be processed further
    });
  }
}
}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-vue-documenteditor/styles/material.css";
</style>
`

```

For instance, to export the document as Rich Text Format file, implement an ASP.NET MVC web API controller using DocIO library by passing the DOCX blob. Refer to the following code example.

```

`c#
//API controller for the conversion.
[HttpPost]
public HttpResponseMessage ExportAsRtf()
{
  System.Web.HttpPostedFile data = HttpContext.Current.Request.Files[0];
  //Opens document stream
  WordDocument wordDocument = new WordDocument(data.InputStream);
  MemoryStream stream = new MemoryStream();
  //Converts document stream as RTF
  wordDocument.Save(stream, FormatType.Rtf);
  wordDocument.Close();
  stream.Position = 0;
  return new HttpResponseMessage() { Content = new StreamContent(stream) };
}
`

```

In client-side, you can consume this web service and save the document as Rich Text Format (.rtf) file. Refer to the following example.

```
`ts
function exportBlob() {
  documenteditor.saveAsBlob('Docx').then((exportedDocument: Blob) => {
    // The blob can be processed further
    let formData: FormData = new FormData();
    formData.append('fileName', 'sample.docx');
    formData.append('data', exportedDocument);
    //Send blob object to server.
    saveAsRtf(formData);
  });
}

function saveAsRtf(formData: FormData): void {
  let httpRequest: XMLHttpRequest = new XMLHttpRequest();
  httpRequest.open('POST', '/api/DocumentEditor/ExportAsRtf', true);
  httpRequest.onreadystatechange = () => {
    if (httpRequest.readyState === 4) {
      if (httpRequest.status === 200 || httpRequest.status === 304) {
        if (!navigator.msSaveBlob) {
          navigator.msSaveBlob(httpRequest.response, 'sample.rtf');
        } else {
          let downloadLink: HTMLAnchorElement =
            document.createElementNS('http://www.w3.org/1999/xhtml', 'a') as HTMLAnchorElement;
          download('sample.rtf', 'rtf', httpRequest.response, downloadLink, 'download' in downloadLink);
        }
      } else {
        console.error(httpRequest.response);
      }
    }
  }
  httpRequest.responseType = 'blob';
  httpRequest.send(formData);
}
```

```
//Download the rtf document.
```

```
function download(fileName: string, extension: string, buffer: Blob, downloadLink:
HTMLAnchorElement, hasDownloadAttribute: Boolean): void {
  if (hasDownloadAttribute) {
    downloadLink.download = fileName;
    let dataUrl: string = window.URL.createObjectURL(buffer);
    downloadLink.href = dataUrl;
    let event: MouseEvent = document.createEvent('MouseEvent');
    event.initEvent('click', true, true);
    downloadLink.dispatchEvent(event);
    setTimeout(() : void => {
      window.URL.revokeObjectURL(dataUrl);
      dataUrl = undefined;
    });
  } else {
    if (extension !== 'docx' && extension !== 'xlsx') {
      let url: string = window.URL.createObjectURL(buffer);
      let isPopupBlocked: Window = window.open(url, '_blank');
      if (!isPopupBlocked) {
        window.location.href = url;
      }
    }
  }
}
```

See Also

- [Feature modules](#)
- [How to export the document as pdf.](#)

### Web services in Vue Document editor component

You can deploy web APIs for server-side dependencies of Document Editor component in the following platforms.

- [ASP.NET Core](#)
- [ASP.NET MVC](#)
- [Java](#)

### Which operations require server-side interaction

|Operations|When client-server communication will be triggered?|What type of data will be transferred between client and server?|

|-----|-----|-----|

|[Open file formats other than SFDT](#)|When opening the document other than SFDT (Syncfusion Document Editor's native file format), the server-side web API is invoked from client-side script. |**Client:** Sends the input file.<br>**Server:** Receives the input file and sends the converted SFDT back to the client.<br><br>The server-side web API internally extracts the content from the document (DOCX, DOC, WordML, RTF, HTML) using Syncfusion Word library (DocIO) and converts it into SFDT for opening the document in Document Editor. |

|[Paste with formatting](#)|When pasting the formatted content (HTML/RTF) received from system clipboard. For converting HTML/RTF to SFDT format.<br><br> **Note:** Whereas plain text received from system clipboard will be pasted directly in the client-side. |**Client:** Sends the input Html or Rtf string. <br>**Server:** Receives the input Html or Rtf string and sends the converted SFDT back to the client. |

|[Restrict editing](#)|When protecting the document, for generating hash. |**Client:** Sends the input data for hashing algorithm.<br> **Server:** Receives the input data for hashing algorithm and sends the result hash information back to the client. |

|[Spellcheck](#)(default)|When the spellchecker is enabled on client-side Document Editor, and it performs the spell check validation for words in the document. |**Client:** Sends the words (string) with their language for spelling validation.<br> **Server:** Receives the words (string) with their language for spelling validation and sends the validation result as JSON back to the client. |

|[SpellCheckByPage](#)|Document editor provides options to spellcheck page by page when loading the documents. By [enabling optimized spell check](#) in client-side, you can perform spellcheck page by page when loading the documents. |**Client:** Sends the words (string) with their language for spelling validation.<br> **Server:** Receives the words (string) with their language for spelling validation and sends the validation result as JSON back to the client. |

|[Save as file formats other than SFDT and DOCX](#) (optional API)|You can configure this API, if you want to save the document in file format other than DOCX and SFDT.<br><br> For saving the files as WordML, DOC, RTF, HTML, ODT, Text using Syncfusion Word library (DocIO) and PDF using Syncfusion Word (DocIO) and PDF libraries. |You can transfer document from client to server either as SFDT or DOCX format.<br><br>First option (SFDT):<br>**Client:** Sends the SFDT.<br>**Server:** Receives the SFDT and saves the converted document as any file format supported by [Syncfusion Word library \(DocIO\)](#) in server or sends the saved file to the client browser.<br><br>Second option (DOCX):<br>**Client:** Sends the DOCX file.<br>**Server:** Receives the DOCX file and saves the converted document as any file format supported by [Syncfusion Word library \(DocIO\)](#) in server or sends the saved file to the client browser. |

Note: If you don't require the above functionalities then you can deploy as pure client-side component without any server-side interactions.

Please refer the [example from GitHub](#) to configure the web service and set the [serviceUrl](#).

If your running web service Url is `http://localhost:62869/`, set the serviceUrl like below:

```
`ts
```

```
this.$refs.container.serviceUrl = "http://localhost:62869/api/documenteditor/";
```

```
,
```

### Required Web API structure

Please check below table for expected web API structure.

Expected method name	Parameters	Return type
----- ---- ----		
Import	Files(IFormCollection)  json(sfdd format)	
SystemClipboard	CustomerParameter: content(type string either rtf or html) and type(either .rtf or .html)  json(sfdd format)	
RestrictEditing	Parameter of type CustomRestrictParameter public class CustomRestrictParameter {     public string passwordBase64 { get; set; }     public string saltBase64 { get; set; }     public int spinCount { get; set; } }  result hash information	
SpellCheck(default)	Parameter: SpellCheckJsonData public class SpellCheckJsonData  {     public int LanguageID { get; set; }     public string TexttoCheck { get; set; }     public bool CheckSpelling { get; set; }     public bool CheckSuggestion { get; set; }     public bool AddWord { get; set; } }  Json type of Spellcheck containing details of spell checked word	
SpellCheckByPage	Parameter: SpellCheckJsonData public class SpellCheckJsonData  {     public int LanguageID { get; set; }     public string TexttoCheck { get; set; }     public bool CheckSpelling { get; set; }     public bool CheckSuggestion { get; set; }     public bool AddWord { get; set; } }  Json type of Spellcheck containing details of spell checked word   <b>Note:</b> Document editor provides options to spellcheck page by page when loading the documents. By <a href="#">enabling optimized spell check</a> in client-side, you can perform spellcheck page by page when loading the documents.	
Save(optional API)	parameter: SaveParameter  public class SaveParameter {     public string Content { get; set; }     public string FileName { get; set; } }  void(Save the file as file stream)	
ExportSFDT(optional API)	parameter: SaveParameter  public class SaveParameter {     public string Content { get; set; }     public string FileName { get; set; } }  FileStreamResult (to save the document in client-side)	
Export(optional API)	Files(IFormCollection)  FileStreamResult (to save the document in client-side)	

### Customize the expected method name

Document editor component provides an option to customize the expected method name for Import, SystemClipboard, RestrictEditing and SpellCheck using [serverActionSettings](#).

The following example code illustrates how to customize the method name using serverActionSettings.

,

```
<template>
<div id="app">
<ejs-documenteditorcontainer ref='documenteditor' :serviceUrl='serviceUrl'
:serverActionSettings='settings' height="590px" id='container' :enableToolbar='true'></ejs-
documenteditorcontainer>
</div>
</template>
```



```
<script>
import Vue from 'vue';

import { DocumentEditorContainerPlugin, DocumentEditorContainerComponent, Toolbar } from
 '@syncfusion/ej2-vue-documenteditor';

Vue.use(DocumentEditorContainerPlugin);

export default {
  data() {
    return {
      serviceUrl: 'https://ej2services.syncfusion.com/production/web-services/api/documenteditor/',
      // Customize the API name
      settings: {
        import: 'Import1',
        systemClipboard: 'SystemClipboard1',
        spellCheck: 'SpellCheck1',
        restrictEditing: 'RestrictEditing1'
      },
      provide: {
        // Inject required modules.
        DocumentEditorContainer: [Toolbar]
      }
    };
  }
};
</script>
```

#### Modify the XMLHttpRequest before request send

Document editor component provides an option to modify the XMLHttpRequest object (setting additional headers, if needed) using [beforeXmlHttpRequestSend](#) event and it gets triggered before a server request.

You can customize the required [XMLHttpRequest](#) properties.

The following example code illustrates how to modify the XMLHttpRequest using [beforeXmlHttpRequestSend](#).

```
<template>
<div id="app">
  <ejs-documenteditorcontainer ref='container' :serviceUrl='serviceUrl' height="590px" id='container'
    :enableToolbar='true'></ejs-documenteditorcontainer>
</div>
</template>
<script>
import Vue from 'vue';
```

```
import { DocumentEditorContainerPlugin, DocumentEditorContainerComponent, Toolbar } from
 '@syncfusion/ej2-vue-documenteditor';
Vue.use(DocumentEditorContainerPlugin);
export default {
  data() {
    return { serviceUrl: 'https://ej2services.syncfusion.com/production/web-services/api/documenteditor/' };
  },
  provide: {
    //Inject require modules.
    DocumentEditorContainer: [Toolbar]
  },
  mounted() {
    //Here, modifying the request headers
    this.$refs.container.headers = [{ syncfusion: 'true' }];
    // Below action, cancel all server-side interactions expect spell check
    this.$refs.container.beforeXmlHttpRequestSend = (
      args ) => {
      args.headers = this.$refs.container.headers;
      args.withCredentials = true;
      switch (args.serverActionType) {
        case 'Import':
        case 'RestrictEditing':
        case 'SystemClipboard':
          args.cancel = true;
          break;
      }
    };
  }
}
</script>
```

Note: Find the customizable serverActionType values are 'Import' | 'RestrictEditing' | 'SpellCheck' | 'SystemClipboard'.

## Server Deployment

### Word processor server docker image overview in Vue Document editor component

The Syncfusion **Word Processor (also known as Document Editor)** is a component with editing capabilities like Microsoft Word. It is used to create, edit, view, and print Word documents. It provides all the common word processing abilities, including editing text; formatting contents; resizing images and tables; finding and replacing text; importing, exporting, and printing Word documents; and using bookmarks and tables of contents.

This Docker image is the predefined Docker container of Syncfusion's Word Processor backend. You can deploy it quickly to your infrastructure.

Word Processor is a commercial product, and it requires a valid license to use it in a production environment ([request license or trial key](#)).

The Word Processor is supported in the JavaScript, Angular, React, Vue, ASP.NET Core, ASP.NET MVC, and Blazor platforms.

### Prerequisites

Have [Docker](#) installed in your environment:

- On Windows, install [Docker for Windows](#).
- On macOS, install [Docker for Mac](#).

### How to deploy Word Processor Docker image

**Step 1:** Pull the word-processor-server image from Docker Hub.

`console

```
docker pull syncfusion/word-processor-server
```

`

**Step 2:** Create the docker-compose.yml file with the following code in your file system.

`yaml

```
version: '3.4'
```

```
services:
```

```
word-processor-server:
```

```
image: syncfusion/word-processor-server:latest
```

```
environment:
```

[Provide your license key for activation](#)

```
SYNCFUSIONLICENSEKEY: YOURLICENSEKEY
```

```
ports:
```

- "6002:80"

`

**Step 3:** In a terminal tab, navigate to the directory where you've placed the docker-compose.yml file and execute the following.

```
`console
docker-compose up
`
```

Now the Word Processor server Docker instance runs in the localhost with the provided port number `http://localhost:6002`. Open this link in a browser and navigate to the Word Processor Web API control `http://localhost:6002/api/documenteditor`. It returns the default get method response.

**Step 4:** Append the Docker instance running the URL (`http://localhost:6002/api/documenteditor`) to the service URL in the client-side Word Processor control. For more information about how to get started with the Word Processor control, refer to this [getting started page](#).

*[How to configure spell checker dictionaries path in Docker compose file](#)*

**Step 1:** In the Docker compose file, mount the local directory as a container volume using the following code.

```
`yaml
version: '3.4'

services:
  word-processor-server:
    image: syncfusion/word-processor-server:latest
    environment:
      Provide your license key for activation
      SYNCFUSIONLICENSEKEY: YOURLICENSEKEY
    volumes:
```

- `./data:/app/data`

```
ports:
```

- `"6002:80"`
- ```
`
```

This YAML definition binds the data folder that is available in the Docker compose file directory.

**Step 2:** In the data folder, include the dictionary files (`.dic`, `.aff`) and JSON file. The JSON file should contain the language based dictionary file configuration in the following format.

```
`yaml
[
{
  "LanguageID": 1036,
```

```

"DictionaryPath": "fr_FR.dic",
"AffixPath": "fr_FR.aff",
"PersonalDictPath": "customDict.dic"
},
{
  "LanguageID": 1033,
  "DictionaryPath": "en_US.dic",
  "AffixPath": "en_US.aff",
  "PersonalDictPath": "customDict.dic"
}
]
`

```

Note: By default, the json file name should be "spellcheck.json". You can also use different file name by mounting the file name to 'SPELLCHECK/JSONFILENAME' attribute in Docker compose file as below,

`yaml

version: '3.4'

services:

word-processor-server:

image: syncfusion/word-processor-server:latest

environment:

[Provide your license key for activation](#)

SYNCFUSIONLICENSEKEY: YOURLICENSEKEY

SPELLCHECKDICTIONARYPATH: data

SPELLCHECKJSONFILENAME: spellcheck1.json

volumes:

- ./data:/app/data

ports:

- "6002:80"

`

**Step 3:** For handling the personal dictionary, place an empty .dic file (e.g., customDict.dic file) in the data folder.

**Step 4:** Provide the configured volume path to the environment variable like in the following in the Docker compose file.

```
`yaml
version: '3.4'
services:
  word-processor-server:
    image: syncfusion/word-processo -server:latest
    environment:
      Provide your license key for activation
      SYNCFUSION/LICENSEKEY: YOURLICENSEKEY
      SPELLCHECKDICTIONARYPATH: data
    volumes:
      • ./data:/app/data
    ports:
      • "6002:80"
```

#### *How to copy template Word documents to Docker image*

You can copy the required template Word documents into docker container while deploying the docker image to server. You can open these Word documents present in the server by passing the document path (name with relative path) to LoadDocument() web API.

Note: Place the word files in the data folder mentioned in the volumes section(i.e., C:/Docker/Data) of the docker-compose.yml file. All the files present in the folder path (C:/Docker/Data) mentioned in the volumes section of 'docker-compose.yml' file will be copied to the respective folder (/app/Data) of docker container. The Word documents copied to docker container can be processed using the 'LoadDocument' web API.

The following code example shows how to use LoadDocument() API in document editor.

```
<template>
<div id="app">
  <ejs-documenteditor ref="documenteditor" id="container_1" height='600px;'
  :enableSpellCheck='true'></ejs-documenteditor>
</div>
</template>
<script>
import Vue from 'vue'
import { DocumentEditorPlugin } from '@syncfusion/ej2-vue-documenteditor';
```

```

Vue.use(DocumentEditorPlugin);
export default {
  data: function () {
    return {
    };
  },
  mounted: function () {
    var dataContext = this;
    var uploadDocument = new FormData();
    uploadDocument.append('DocumentName', 'Getting Started.docx');
    var baseUrl = 'http://localhost:6002/api/documenteditor/LoadDocument';
    var httpRequest = new XMLHttpRequest();
    httpRequest.open('POST', baseUrl, true);
    httpRequest.onreadystatechange = function () {
      if (httpRequest.readyState === 4) {
        if (httpRequest.status === 200 || httpRequest.status === 304) {
          //Open the document in DocumentEditor
          this.$refs.documenteditor.ej2Instances.open(httpRequest.responseText);
        }
      }
    };
    httpRequest.send(uploadDocument);
  }
}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-vue-documenteditor/styles/material.css";
</style>
`

```

Refer to these getting started pages to create a Word Processor in [Typescript](#), [React](#), [Vue](#), [ASP.NET MVC](#), [ASP.NET Core](#), and [Blazor](#).

## How to deploy word processor server docker container in azure app service in Vue Document editor component

### Prerequisites

- Have [Azure account](#) and [Azure CLI](#) setup in your environment.
- Run the following command to open the Azure login page. Sign into your [Microsoft Azure account](#).

,

az login

,

#### Step 1: Create a resource group.

Create a resource group using the [az group create](#) command.

The following example creates a resource group named documenteditorresourcegroup in the eastus location.

,

```
az group create --name documenteditorresourcegroup --location "East US"
```

,

#### Step 2: Create an Azure App Service plan.

Create an App Service plan in the resource group with the [az appservice plan create](#) command.

The following example creates an App Service plan named documenteditorappservice in the Standard pricing tier (--sku S1) and in a Linux container (--is-linux).

,

```
az appservice plan create --name documenteditorappservice --resource-group documenteditorresourcegroup --sku S1 --is-linux
```

,

#### Step 3: Create a Docker Compose app.

Create a multi-container [web app](#) in the documenteditorappservice App Service plan with the [az webapp create](#) command. The following command creates the web app using the provided Docker compose file. Please look into the section for getting started with Docker compose to create the Docker compose file for the document editor server and use the created Docker compose file here.

,

```
az webapp create --resource-group documenteditorresourcegroup --plan documenteditorappservice --name documenteditor-server --multicontainer-config-type compose --multicontainer-config-file documenteditor-server-compose.yml
```

,

#### Step 4: Browse to the app.



Browse to the deployed app at `http://<app_name>.azurewebsites.net`, i.e. `http://documenteditor-server.azurewebsites.net`. Browse this link and navigate to the Document Editor Web API control `http://documenteditor-server.azurewebsites.net/api/documenteditor`. It returns the default get method response.

Append the app service running the URL `http://documenteditor-server.azurewebsites.net/api/documenteditor/` to the service URL in the client-side Document Editor control. For more information about the Document Editor control, refer to this [getting started page](#).

For more information about the app container service, please look deeper into the [Microsoft Azure Container Service](#) for a production-ready setup.

How to deploy word processor server docker container in azure kubernetes service in Vue Document editor component

#### *Prerequisites*

- Have [Azure account](#) and [Azure CLI](#) setup in your environment.
- Run the following command to open the Azure login page. Sign into your [Microsoft Azure account](#).

az login

#### **Step 1:** Create a resource group.

Create a resource group using the [az group create](#) command.

The following example creates a resource group named `documenteditorresourcegroup` in the `eastus` location.

```
az group create --name documenteditorresourcegroup --location "East US"
```

#### **Step 2:** Create AKS cluster.

Use the [az aks create](#) command to create an AKS cluster. The following example creates a cluster named `documenteditorcluster` with one node.

```
az aks create --resource-group documenteditorresourcegroup --name documenteditorcluster --node-count 1
```

#### **Step 3:** Connect to the cluster.

Install the [kubect](#) into the workspace using the following command.

```
az aks install-cli
```

```
,
```

To configure kubectl to connect to your Kubernetes cluster, use the [az aks get-credentials](#) command. This command downloads credentials and configures the Kubernetes CLI to use them.

```
,
```

```
az aks get-credentials --resource-group documenteditorresourcegroup --name documenteditorcluster
```

```
,
```

#### **Step 4:** Create Kubernetes Services and Deployments

[Kubernetes Services](#) and [Deployments](#) can be configured in a file. To run the Document Editor server, you must define a Service and a Deployment `documenteditorserver`. To do this, create the `documenteditor-server.yml` file in the current directory using the following code.

```
`yaml
```

```
apiVersion: apps/v1
```

```
kind: Deployment
```

```
metadata:
```

```
labels:
```

```
app: documenteditorserver
```

```
name: documenteditorserver
```

```
spec:
```

```
replicas: 1
```

```
selector:
```

```
matchLabels:
```

```
app: documenteditorserver
```

```
strategy: {}
```

```
template:
```

```
metadata:
```

```
labels:
```

```
app: documenteditorserver
```

```
spec:
```

```
containers:
```

- `image: syncfusion/word-processor-server:latest`

```
name: documenteditorserver
```

```
ports:
```

- containerPort: 80

env:

- name: SYNCFUSIONLICENSEKEY

value: "YOURLICENSEKEY"

apiVersion: v1

kind: Service

metadata:

labels:

app: documenteditorserver

name: documenteditorserver

spec:

ports:

- port: 80

targetPort: 80

selector:

app: documenteditorserver

type: LoadBalancer

,

**Step 5:** To create all Services and Deployments needed to run the Document Editor server, execute the following.

```
`console
```

```
kubectl create -f ./documenteditor-server.yml
```

,

Run the following command to get the Kubernetes cluster deployed service details and copy the external IP address of the Document Editor service.

```
`console
```

```
kubectl get all
```

,

Browse the copied external IP address and navigate to the Document Editor Web API control `http://<external-ip>/api/documenteditor`. It returns the default get method response.

**Step 6:** Append the Kubernetes service running the URL `http://<external-ip>/api/documenteditor/` to the service URL in the client-side Document Editor control. For more information about the Document Editor control, refer to this [getting started page](#).

For more details about the Azure Kubernetes service, please look deeper into [Microsoft Azure Kubernetes Service](#) for a production-ready setup.

How to publish documenteditor web api application in azure app service from visual studio in Vue Document editor component

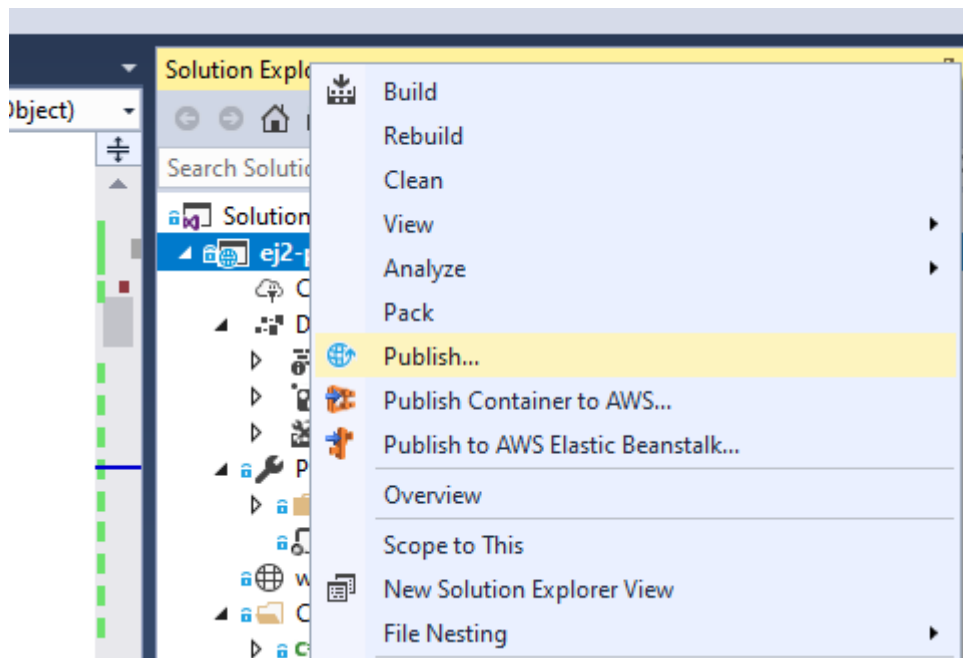
#### *Prerequisites*

- Visual Studio 2017 or 2019.
- An [Azure subscription](#).
- The Document Editor Web API controller application from [here](#).

Make sure you build the project using the Build > Build Solution menu command before following the deployment steps.

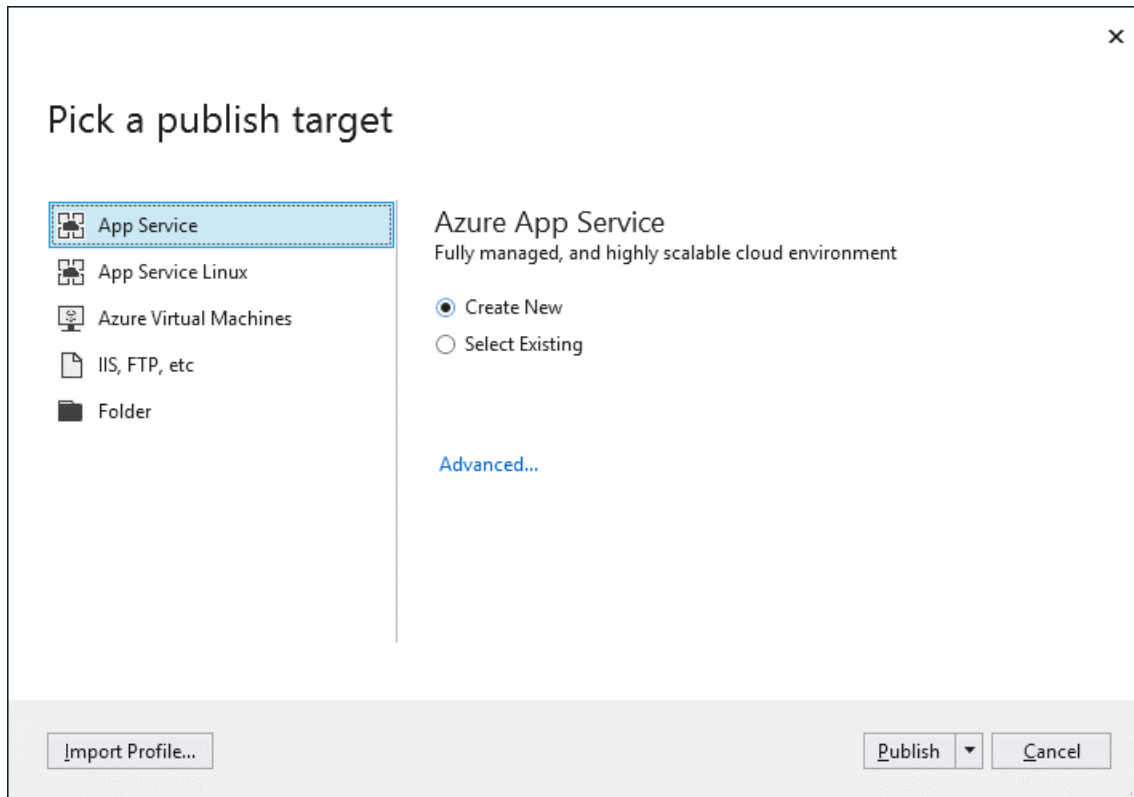
#### *Publish to Azure App Service*

**Step 1:** In Solution Explorer, right-click the project and click Publish (or use the Build > Publish menu item).



**Step 2:** If you have previously configured any publishing profiles, the Publish pane appears, in which case select Create new profile.

**Step 3:** In the Pick a publish target dialog box, select App Service.



**Step 4:** Select Publish. The Create App Service dialog box appears. Sign in with your Azure account, if necessary, and then the default app service settings populate the fields.

App Name  
ej2-documenteditor-server20200514102909

Subscription  
Microsoft Azure Enterprise

Resource Group  
cloud-shell-storage-centralindia (centralindia) [New...](#)

Hosting Plan  
ej2-documenteditor-server20200514102909P\* (South Centra [New...](#)

Application Insights  
None

Explore additional Azure services

- [Create a SQL Database](#)
- [Create a storage account](#)

Clicking the Create button will create the following Azure resources

- Hosting Plan - ej2-documenteditor-server202005141...
- App Service - ej2-documenteditor-server20200514102909

Export... [Create](#) [Cancel](#)

**Step 5:** Select Create. Visual Studio deploys the app to your Azure App Service, and the web app loads in your browser with the app name at `http://<app_name>.azurewebsites.net` (i.e. `http://ej2-documenteditor-server20200514102909.azurewebsites.net`).

**Step 6:** Navigate to Document Editor Web API control `http://ej2-documenteditor-server20200514102909.azurewebsites.net/api/documenteditor`. It returns the default get method response.

Append the app service running the URL `http://ej2-documenteditor-server20200514102909.azurewebsites.net/api/documenteditor` to the service URL in the client-side Document Editor control. For more information about how to get started with the Document Editor control, refer to this [getting started page](#).

For more information about the app container service, please look deeper into the [Microsoft Azure App Service](#) for a production-ready setup.

How to deploy documenteditor java web api in azure in Vue Document editor component

#### *Prerequisites*

Have [Azure account](#) and [Azure CLI](#) setup in your environment.

You can get the example [web service project from GitHub](#) and then perform the following steps to create the packages and host in azure app service.

**Step 1** Clean the package using following command.

```
`console
mvn clean package
`
```

**Step 2** Run the application locally using following command.

```
`console
mvn spring-boot:run
`
```

**Step 3** Build the package using following command.

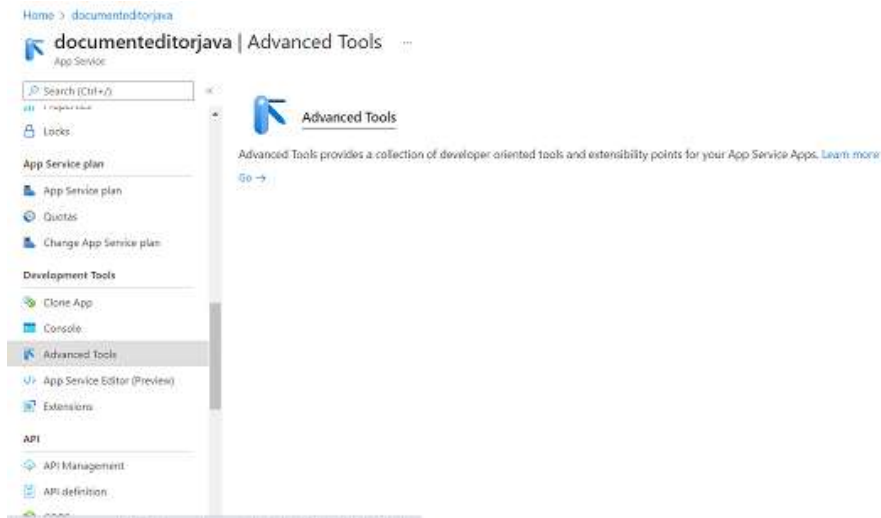
```
`console
mvn package
`
```

Above package generation command creates the `tomcat-0.0.1-SNAPSHOT.war` in the below location in the sample folder.

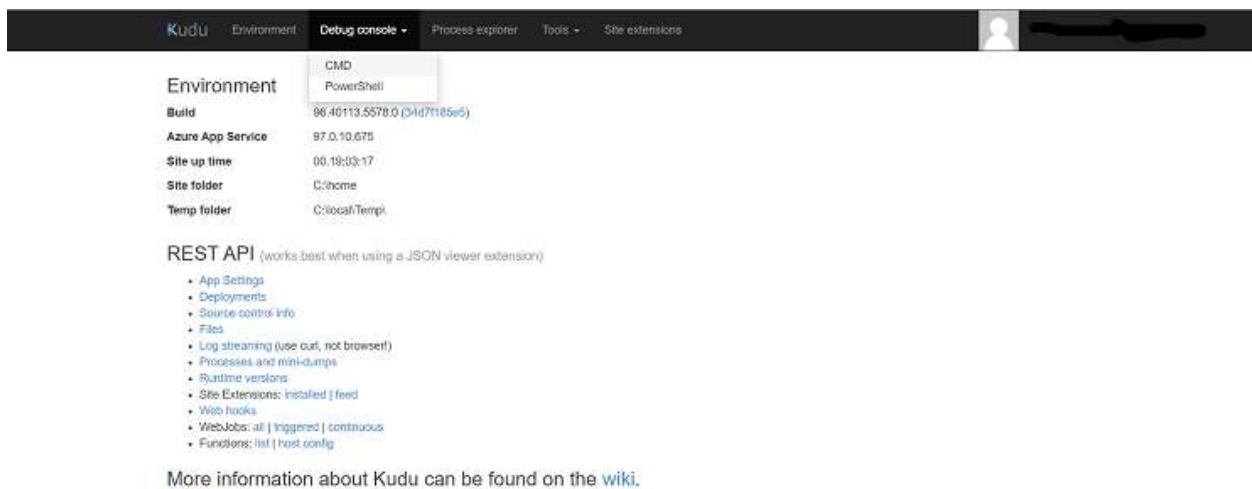
`target/tomcat-0.0.1-SNAPSHOT.war`

**Step 4:** Create a Azure app service with Java & Tomcat. For example, create the app services name as `documenteditorjava`.

**Step 5:** After creating app service, navigate to `Advanced Tools` options under `Development Tools`.



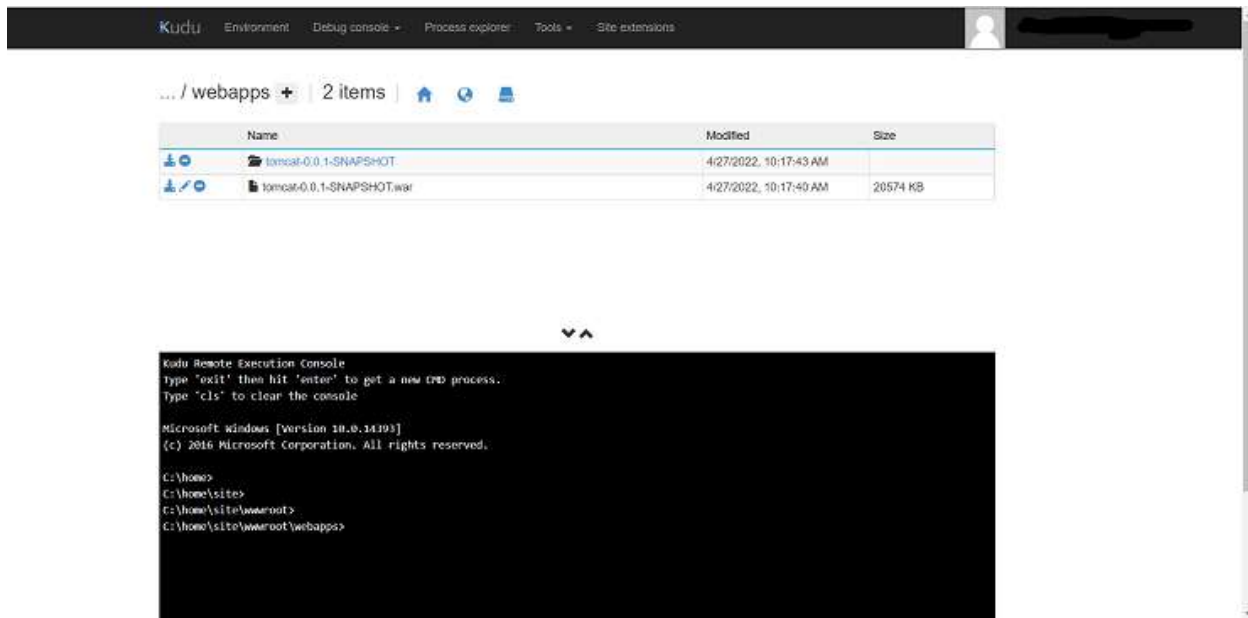
Then, click **Go** and select the **CMD** options under **Debug console**.



**Step 6:** Once the file manager is opened, please navigate to

site -> wwwroot -> webapps

**Step 7:** Now, upload the generated war file **tomcat-0.0.1-SNAPSHOT.war**. Uploaded war file gets extracted automatically, it will be uploaded like below:



### Step 8: Browse to the app.

Browse to the deployed app at [http://<app\\_name>.azurewebsites.net](http://<app_name>.azurewebsites.net), i.e. <http://documenteditorjava.azurewebsites.net>. Browse this link and it navigate to the Document Editor Web API control <http://documenteditorjava.azurewebsites.net/tomcat-0.0.1-SNAPSHOT>. It returns the default get method response.

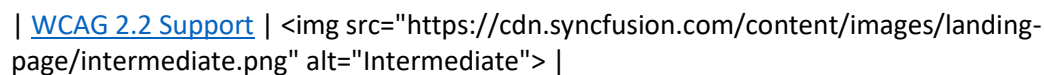
Append the app service running the URL <http://documenteditorjava.azurewebsites.net/tomcat-0.0.1-SNAPSHOT> to the service URL in the client-side Document Editor control. For more information about the Document Editor control, refer to this [getting started page](#).

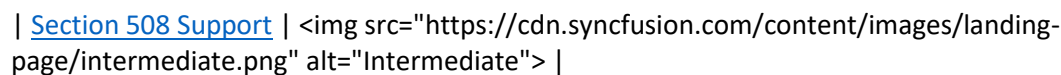
### Accessibility in Vue Document editor component

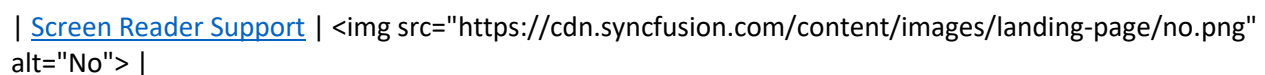
The accessibility compliance for the Document editor component is outlined below.

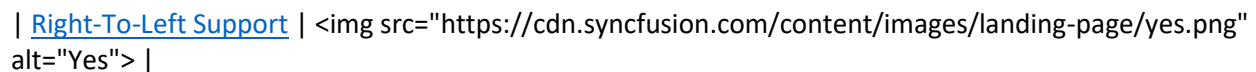
| Accessibility Criteria | Compatibility |

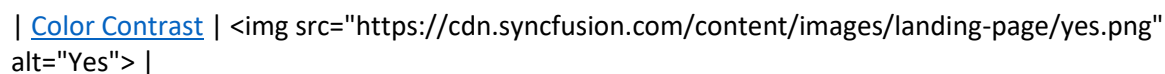
| -- | -- |

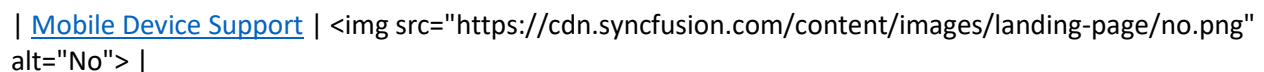
| [WCAG 2.2 Support](#) |  |

| [Section 508 Support](#) |  |

| [Screen Reader Support](#) |  |

| [Right-To-Left Support](#) |  |

| [Color Contrast](#) |  |

| [Mobile Device Support](#) |  |



```
| Keyboard Navigation Support |  |
| Accessibility Checker Validation |  |
| Axe-core Accessibility Validation |  |
<style>
.post .post-content img {
display: inline-block;
margin: 0.5em 0;
}
</style>
<div> - All
features of the component meet the requirement.</div>
<div> - Some features of the component do not meet the requirement.</div>
<div> - The
component does not meet the requirement.</div>
```

### Keyboard interaction

Document editor supports [keyboard shortcuts](#).

### Ensuring accessibility

The Document editor component's accessibility levels are ensured through an [accessibility-checker](#) and [axe-core](#) software tools during automated testing.

The accessibility compliance of the Document editor component is shown in the following sample. Open the [sample](#) in a new window to evaluate the accessibility of the Document editor component with accessibility tools.

```
{% previewsample "https://ej2.syncfusion.com/accessibility/wordprocessor.html" %}
```

See also

- [Accessibility in Syncfusion Vue components](#)

### Collaborative Editing (preview)

Allows multiple users to work on the same document simultaneously. This can be done in real-time, so that collaborators can see the changes as they are made. Collaborative editing can be a great way to improve efficiency, as it allows team members to work together on a document without having to wait for others to finish their changes.

Note: Collaborative editing support is currently in preview mode only and is not yet ready for production environments.

### Prerequisites

The following are needed to enable collaborative editing in Document Editor.

- SignalR
- Microsoft SQL Server

### How to enable collaborative editing in client side

#### *Step 1: Enable collaborative editing in Document Editor*

To enable collaborative editing, inject `CollaborativeEditingHandler` and set the property `enableCollaborativeEditing` to true in the Document Editor, like in the code snippet below.

```
`javascript
<template>
  <div id="spinner">
    <ejs-documenteditorcontainer ref="doceditcontainer" :contentChange="onContentChange"
    :serviceUrl='serviceUrl'
    :enableToolbar='true' v-bind:created="onCreated">
  </ejs-documenteditorcontainer>
</div>
</template>
<script>
import { DocumentEditorContainerComponent, Toolbar, DocumentEditor } from '@syncfusion/ej2-vue-documenteditor';
import { CollaborativeEditingHandler } from '@syncfusion/ej2-documenteditor';
import { HubConnectionBuilder, HttpTransportType, HubConnectionState } from '@microsoft/signalr';
import { hideSpinner, showSpinner } from '@syncfusion/ej2-popups';
export default {
  name: 'App',
  components: {
    'ejs-documenteditorcontainer': DocumentEditorContainerComponent
  },
  data() {
    return {
      serviceUrl: 'http://localhost:5212/api/documenteditor/',
      collaborativeEditingServiceUrl: "http://localhost:5212/",
      collaborativeEditingHandler: null,
      connection: null,
```

```
};
},
provide: {
  DocumentEditorContainer: [Toolbar]
},
methods: {
  onCreated() {
    DocumentEditor.Inject(CollaborativeEditingHandler);
    // Enable collaborative editing in Document Editor.
    this.$refs.doceditcontainer.ej2Instances.documentEditor.enableCollaborativeEditing = true;
    this.initializeSignalR();
    this.loadDocumentFromServer();
  }
}
},
`
```

#### *Step 2: Configure SignalR to send and receive changes*

To broadcast the changes made and receive changes from remote users, configure SignalR like below.

```
`javascript
methods: {
  initializeSignalR() {
    // SignalR connection
    this.connection = new HubConnectionBuilder().withUrl(this.collaborativeEditingServiceUrl +
    'documenteditorhub', {
      skipNegotiation: true,
      transport: HttpTransportType.WebSockets
    }).withAutomaticReconnect().build();
    // Event handler for signalR connection
    this.connection.on('dataReceived', this.onDataRecived.bind(this));
    this.connection.onclose(async () => {
      if (this.connection && this.connection.state === HubConnectionState.Disconnected) {
        alert('Connection lost. Please relod the browser to continue.');
```

```
,
onDataRecived(action, data) {
  if (this.collaborativeEditingHandler) {
    if (action == 'connectionId') {
      // Update the current connection id to track other users
      this.connectionId = data;
    }
    // Apply the remote action in DocumentEditor
    this.collaborativeEditingHandler.applyRemoteAction(action, data);
  }
},
connectToRoom(data) {
  try {
    if (this.connection) {
      // start the connection.
      this.connection.start().then(() => {
        // Join the room.
        if (this.connection) {
          this.connection.send('JoinGroup', { roomName: data.roomName, currentUser: data.currentUser });
        }
        console.log('server connected!!!');
      });
    }
  } catch (err) {
    console.log(err);
    // Attempting to reconnect in 5 seconds
    setTimeout(this.connectToRoom, 5000);
  }
}
},
```

### *Step 3: Join SignalR room while opening the document*

When opening a document, we need to generate a unique ID for each document. These unique IDs are then used to create rooms using SignalR, which facilitates sending and receiving data from the server.

```
`javascript
methods: {
  openDocument(responseText, roomName) {
    showSpinner(document.getElementById('spinner'));
    let data = JSON.parse(responseText);
    if (this.$refs.doceditcontainer) {
      this.collaborativeEditingHandler =
        this.$refs.doceditcontainer.ej2Instances.documentEditor.collaborativeEditingHandlerModule;
      // Update the room and version information to collaborative editing handler.
      this.collaborativeEditingHandler.updateRoomInfo(roomName, data.version,
        this.collaborativeEditingServiceUrl + 'api/CollaborativeEditing/');
      // Open the document
      this.$refs.doceditcontainer.ej2Instances.documentEditor.open(data.sfdt);
      setTimeout(() => {
        if (this.$refs.doceditcontainer) {
          // connect to server using signalR
          this.connectToRoom({ action: 'connect', roomName: roomName, currentUser:
            this.$refs.doceditcontainer.currentUser });
        }
      });
    }
    hideSpinner(document.getElementById('spinner'));
  }
},
`
```

#### *Step 4: Broadcast current editing changes to remote users*

Changes made on the client-side need to be sent to the server-side to broadcast them to other connected users. To send the changes made to the server, use the method shown below from the document editor using the `contentChange` event.

```
`javascript
methods: {
  onContentChange(args) {
    if (this.collaborativeEditingHandler) {
      // Send the editing action to server
      this.collaborativeEditingHandler.sendActionToServer(args.operations)
    }
  }
}
```

```

}
},
}
`

```

How to enable collaborative editing in ASP.NET Core

#### *Step 1: Configure SignalR in ASP.NET Core*

We are using Microsoft SignalR to broadcast the changes. Please add the following configuration to your application's Program.cs file.

```

`csharp
using Microsoft.Azure.SignalR;

.....

builder.Services.AddSignalR();

.....

.....

.....

app.MapHub<DocumentEditorHub>("/documenteditorhub");

.....

.....
`

```

#### *Step 2: Configure SignalR hub to create room for collaborative editing session*

To manage groups for each document, create a folder named "Hub" and add a file named "DocumentEditorHub.cs" inside it. Add the following code to the file to manage SignalR groups using room names.

Join the group by using unique id of the document by using `JoinGroup` method.

```

`csharp
static Dictionary<string, ActionInfo> userManager = new Dictionary<string, ActionInfo>();
internal static Dictionary<string, List<ActionInfo>> groupManager = new Dictionary<string,
List<ActionInfo>>>();

// Join to the specified room name
public async Task JoinGroup(ActionInfo info)
{
    if (!userManager.ContainsKey(Context.ConnectionId))
    {
        userManager.Add(Context.ConnectionId, info);
    }
}

```

```
info.ConnectionId = Context.ConnectionId;
//Add the current connected use to the specified group
await Groups.AddToGroupAsync(Context.ConnectionId, info.RoomName);
if (groupManager.ContainsKey(info.RoomName))
{
    await Clients.Caller.SendAsync("dataReceived", "addUser", groupManager[info.RoomName]);
}
lock (groupManager)
{
    if (groupManager.ContainsKey(info.RoomName))
    {
        groupManager[info.RoomName].Add(info);
    }
    else
    {
        List<ActionInfo> actions = new List<ActionInfo>
        {
            info
        };
        groupManager.Add(info.RoomName, actions);
    }
}
// Notify other users in the group about new user joined the collaborative editing session.
Clients.GroupExcept(info.RoomName, Context.ConnectionId).SendAsync("dataReceived", "addUser",
info);
`

Handle user disconnection using SignalR.
`csharp
//Handle disconnection from group.
public override Task OnDisconnectedAsync(Exception? e)
{
    string roomName = userManager[Context.ConnectionId].RoomName;
```

```

if (groupManager.ContainsKey(roomName))
{
    groupManager[roomName].Remove(userManager[Context.ConnectionId]);
    if (groupManager[roomName].Count == 0)
    {
        groupManager.Remove(roomName);
        //If all user disconnected from current room. Auto save the change to source document.
        CollaborativeEditingController.UpdateOperationsToSourceDocument(roomName,
        "<<documentpath>>", false);
    }
}
if (userManager.ContainsKey(Context.ConnectionId))
{
    //Notify other user in the group about user exit the collaborative editing session
    Clients.OthersInGroup(roomName).SendAsync("dataReceived", "removeUser", Context.ConnectionId);
    Groups.RemoveFromGroupAsync(Context.ConnectionId, roomName);
    userManager.Remove(Context.ConnectionId);
}
return base.OnDisconnectedAsync(e);
}
,

```

*Step 3: Configure Microsoft SQL database connection string in application level*

Configure the SQL database that stores temporary data for the collaborative editing session. Provide the SQL database connection string in `appsettings.json` file.

```

`json
.....
"ConnectionStrings": {
  "DocumentEditorDatabase": "<SQL server connection string>"
}
.....
,

```



*Step 4: Configure Web API actions for collaborative editing**Import File*

1. When opening a document, create a database table to store temporary data for the collaborative editing session. 2. If the table already exists, retrieve the records from the table and apply them to the WordDocument instance using the `UpdateActions` method before converting it to the SFDT format.

```
`csharp
public string ImportFile([FromBody] FileInfo param)
{
    .....
    .....
    DocumentContent content = new DocumentContent();
    .....
    //Get source document from database/file system/blob storage
    WordDocument document = GetDocumentFromDatabase(param.fileName, param.documentOwner);
    .....
    //Get temporary records from database
    List<ActionInfo> actions = CreatedTable(param.fileName);
    if(actions!=null)
    {
        //Apply temporary data to the document.
        document.UpdateActions(actions);
    }
    string json = Newtonsoft.Json.JsonConvert.SerializeObject(document);
    content.version = 0;
    content.sfdt = json;
    return Newtonsoft.Json.JsonConvert.SerializeObject(content);
}
`
```

*Update editing records to database.*

Each edit operation made by the user is sent to the server and is pushed to the database. Each operation receives a version number after being inserted into the database.

After inserting the records to the server, the position of the current editing operation must be transformed against any previous editing operations not yet synced with the client using the `TransformOperation` method.

After performing the transformation, the current operation is broadcast to all connected users within the group.

```
`csharp
public async Task<ActionInfo> UpdateAction([FromBody] ActionInfo param)
{
    try
    {
        ActionInfo modifiedAction = AddOperationsToTable(param);
        //After transformation broadcast changes to all users in the group
        await _hubContext.Clients.Group(param.RoomName).SendAsync("dataReceived", "action",
            modifiedAction);
        return modifiedAction;
    }
    catch
    {
        return null;
    }
}

private ActionInfo AddOperationsToTable(ActionInfo action)
{
    int clientVersion = action.Version;
    string tableName = action.RoomName;
    .....
    .....
    .....
    .....

    List<ActionInfo> actions = GetOperationsQueue(tableName);
    foreach (ActionInfo info in actions)
    {
        if (!info.IsTransformed)
        {
            CollaborativeEditingHandler.TransformOperation(info, actions);
        }
    }
    action = actions[actions.Count - 1];
}
```

```

action.Version = updateVersion;
//Return the transformed operation to broadcast it to other clients.
return action;
}
`

```

#### Add Web API to get previous operation as a backup to get lost operations

On the client side, messages broadcasted using SignalR may be received in a different order, or some operations may be missed due to network issues. In these cases, we need a backup method to retrieve missing records from the database.

Using the following method, we can retrieve all operations after the last successful client-synced version and return all missing operations to the requesting client.

```

`csharp
public async Task<ActionInfo> UpdateAction([FromBody] ActionInfo param)
{
    try
    {
        ActionInfo modifiedAction = AddOperationsToTable(param);
        //After transformation broadcast changes to all users in the group
        await _hubContext.Clients.Group(param.RoomName).SendAsync("dataReceived", "action",
            modifiedAction);
        return modifiedAction;
    }
    catch
    {
        return null;
    }
}

private ActionInfo AddOperationsToTable(ActionInfo action)
{
    int clientVersion = action.Version;
    string tableName = action.RoomName;
    .....
    .....
    .....
    .....
}

```

```
List<ActionInfo> actions = GetOperationsQueue(table);
foreach (ActionInfo info in actions)
{
    if (!info.IsTransformed)
    {
        CollaborativeEditingHandler.TransformOperation(info, actions);
    }
}
action = actions[actions.Count - 1];
action.Version = updateVersion;
//Return the transformed operation to broadcast it to other clients.
return action;
}
```

Full version of the code discussed about can be found in below GitHub location.

Github Example: [Collaborative editing examples](#)

### Image in Vue Document editor component

Document Editor supports common raster format images like PNG, BMP, JPEG, SVG and GIF. You can insert an image file or online image in the document using the [insertImage\(\)](#) method. Refer to the following sample code.

The following example shows how to open bookmark dialog in Document Editor.

#### **APP.VUE**

```
<template>
  <div id="app">
    <input id="insertImageButton" ref="insertImageButton"
    style="position:fixed; left:-100em" type="file" v-on:change="onInsertImage"
    accept=".jpeg,.jpg,.png,.gif,.bmp">
    <div>
      <button v-on:click='insertImageButtonClick' >Insert
      Image</button>
    </div>
    <ejs-documenteditor ref="documenteditor"
    :enableSfdtExport='true' :enableWordExport='true' :enableSelection='true'
    :enableEditor='true' :isReadOnly='false' height="370px" style="width:
    100%;display:block" ></ejs-documenteditor>
  </div>
</template>
<script>
  import Vue from 'vue'
  import { DocumentEditorPlugin, Selection, Editor, SfdtExport,
  WordExport } from '@syncfusion/ej2-vue-documenteditor';
  Vue.use(DocumentEditorPlugin);
```

```

export default {
  data: function() {
    return {
    };
  },
  provide: {
    //Inject require modules.
    DocumentEditor : [SfdtExport, WordExport, Selection, Editor]
  },
  methods: {
    onInsertImage: function(args: any): void {
      if (navigator.userAgent.match('Chrome') ||
navigator.userAgent.match('Firefox') || navigator.userAgent.match('Edge') ||
navigator.userAgent.match('MSIE') || navigator.userAgent.match('.NET')) {
        let documenteditor =this.$refs.documenteditor;
        if (args.target.files[0]) {
          let path = args.target.files[0];
          let reader = new FileReader();
          reader.onload = function (frEvent: any) {
            let base64String = frEvent.target.result;
            let image = document.createElement('img');
            image.addEventListener('load', function () {
              //Insert image in Document Editor.

documenteditor.ej2Instances.editor.insertImage(base64String, this.width,
this.height);

            })
            image.src = base64String;
          };
          reader.readAsDataURL(path);
        }
        //Safari does not Support FileReader Class
      } else {
        let image = document.createElement('img');
        image.addEventListener('load', function () {
          //Insert image in Document Editor.

documenteditor.ej2Instances.editor.insertImage(args.target.value);
        })
        image.src = args.target.value;
      }
    },
    insertImageButtonClick: function() {
      this.$refs.insertImageButton.value = '';
      this.$refs.insertImageButton.click();
    }
  }
}
</script>
<style>
  @import "../node_modules/@syncfusion/ej2-vue-
documenteditor/styles/material.css";
</style>

```

```
{% previewsample "page.domainurl/code-snippet/document-editor/bookmark-cs2" %}
```

Image files will be internally converted to base64 string. Whereas, online images are preserved as URL.

Note: EMF and WMF images can't be inserted, but these types of images will be preserved in Document Editor when using ASP.NET MVC Web API.

#### Image resizing

Document Editor provides built-in image resizer that can be injected into your application based on the requirements. This allows you to resize the image by dragging the resizing points using mouse or touch interactions. This resizer appears as follows.



#### Changing size

Document Editor expose API to get or set the size of the selected image. Refer to the following sample code.

```
`ts  
this.$refs.documenteditor.ej2Instances.selection.imageFormat.width = 800;  
this.$refs.documenteditor.ej2Instances.selection.imageFormat.height = 800;  
`
```

Note: Images are stored and processed(read/write) as base64 string in Document Editor. The online image URL is preserved as a URL in Document Editor upon saving.

#### Text wrapping style

Text wrapping refers to how images fit with surrounding text in a document. Please [refer to this page](#) for more information about text wrapping styles available in Word documents.

#### Positioning the image

Document Editor preserves the position properties of the image and displays the image based on position properties. It does not support modifying the position properties. Whereas the image will be automatically moved along with text edited if it is positioned relative to the line or paragraph.

#### See Also

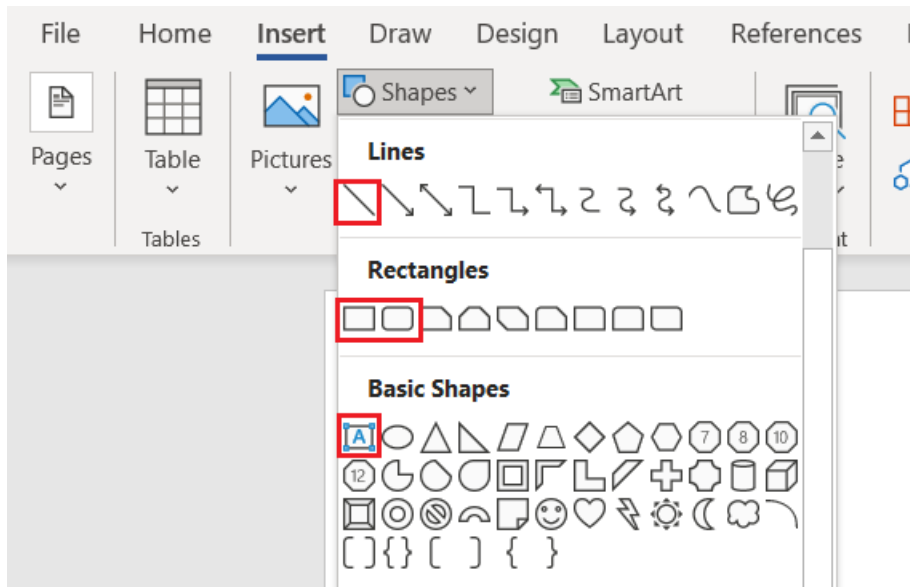
- [Feature modules](#)

### Shapes in Vue Document editor component

Shapes are drawing objects that include a text box, rectangles, lines, curves, circles, etc. It can be preset or custom geometry. At present, Document Editor does not have support to insert shapes. However, if the document contains a shape while importing, it will be preserved properly.

#### Supported shapes

The Document Editor has preservation support for Text box, Rectangle, Rounded Rectangle and Line shapes.



Note: When using ASP.NET MVC service, the unsupported shapes will be converted as image and preserved as image.

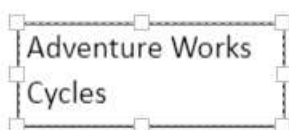
#### Text box Shape

A text box is a rectangular area on the document where you can enter text. When you click in a text box, a flashing cursor will display indicating that you can begin typing. It allows you to enter multiple lines of text with all text formatting.

Adventure Works Cycles, the fictitious company on which the Adventure Works sample databases are based, is a large, multinational manufacturing company. The company manufactures and sells metal **Adventure Works Cycles** and composite bicycles to North American, European and Asian commercial markets. While its base operation is located in Bothell, Washington with 290 employees, several regional sales teams are located throughout their market base.

#### Shape Resizer

The Document Editor also supports a built-in shape resizer to resize the shapes present in the document. The shape resizer accepts both touch and mouse interactions.



### Text wrapping style

Text wrapping refers to how shapes fit with surrounding text in a document. Please [refer to this page](#) for more information about text wrapping styles available in Word documents.

### Positioning the shape

Document Editor preserves the position properties of the shape and displays the shape based on position properties. It does not support modifying the position properties. Whereas the shape will be automatically moved along with text edited if it is positioned relative to the line or paragraph.

### Text wrapping style in Vue Document editor component

Text wrapping refers to how images and shapes are fit with surrounding text in a document. Currently, Document Editor has only preservation support for image and textbox shape with below wrapping styles.

#### In-Line with Text

In this option, the image or shape is placed on the same line surrounding with text like any other word or letter. This image or shape will be automatically moved along with the text while editing, whereas the other options denote that the image or shape stays in a fixed position while the text shifts and wraps around it.

Adventure Works Cycles, the fictitious company on which the AdventureWorks sample databases are based, is a large, multinational manufacturing company. The company



manufactures and sells metal and composite bicycles to North American, European and Asian commercial markets. While its base operation is located in

#### In Front of Text

In this option, the image or shape is placed in front of the text. This can be used to place an image around some text or to add shape to highlight the part in a paragraph.

Adventure Works Cycles, the fictitious company on which the AdventureWorks sample databases are based, is a large manufacturing company. The company manufactures and sells metal and composite bicycles to North American, European and Asian commercial markets. While its base operation is located in Bothell, Washington with 290 employees, several regional sales representatives are located and shipped in throughout their market base.



Note: Starting from v18.2.0.x, the in front of wrapping styles are supported.

#### Top and Bottom

In this option, Text wraps above and below the image or shape. No text is to the left or right of the image or shape. This can be used for larger images or shapes that occupy most of the width in a document.

Note: Starting from v19.1.0.x, the top and bottom wrapping style is supported.



Adventure Works Cycles, the fictitious company on which the AdventureWorks sample databases are based, is a large, multinational manufacturing company. The company



manufactures and sells metal and composite bicycles to North American, European and Asian commercial markets. While its base operation is located in Bothell, Washington with 290

#### Behind

In this option, the image or shape is placed behind the text. This can be used when you need to add a watermark or background image to a document.

Adventure Works Cycles, the fictitious company on which the AdventureWorks sample databases are based, is a large, multinational manufacturing company. The company manufactures and sells metal and composite bicycles to North American, European and Asian commercial markets. While its base operation is located in Bothell, Washington with 290 employees, several regional sales teams and manufacturer and located and shipped in throughout their market base.

Note: Starting from v19.2.0.x, behind text wrapping styles are supported.

#### Square

In this option, Text wraps around the image or text box in a square shape.

Note: Tight and Through styles will be preserved as square wrapping style in Document Editor which is supported from v19.2.0.x.

Adventure Works Cycles, the fictitious company on which the Adventure Works sample databases are based, is a large, multinational manufacturing company. The company manufactures and sells metal and composite bicycles to North American, European and Asian commercial markets. While its base operation is located in Bothell, Washington with 290 employees, several regional sales teams are located throughout their market base.

#### Bookmark in Vue Document editor component

Bookmark is a powerful tool that helps you to mark a place in the document to find again easily. You can enter many bookmarks in the document and give each one a unique name to identify easily.

Document Editor provides built-in dialog to add, delete, and navigate bookmarks within the document. To add a bookmark, select a portion of text in the document. After that, jump to the location or add links to it within the document using built-in hyperlink dialog. You can also delete bookmarks from a document.

Bookmark names need to begin with a letter. They can include both numbers and letters, but not spaces. To separate the words, use an underscore.

Bookmark names starting with an underscore are called hidden bookmarks. For example, bookmarks generated for table of contents.

#### Add bookmark

Using [insertBookmark](#) method, Bookmark can be added to the selected text.

```
`c#
this.$refs.container.ej2Instances.documentEditor.editor.insertBookmark("Bookmark1");
`
```

#### Select Bookmark

You can select the bookmark in the document using [selectBookmark](#) method by providing Bookmark name to select as shown in the following code snippet.

```
`c#
this.$refs.container.ej2Instances.documentEditor.selection.selectBookmark("Bookmark1", true)
`
```

Note: Second parameter is optional parameter and it denotes is exclude bookmark start and end from selection. If true, excludes bookmark start and end from selection.

#### Delete Bookmark

You can delete bookmark in the document using [deleteBookmark](#) method as shown in the following code snippet.

```
`c#
this.$refs.container.ej2Instances.documentEditor.editor.deleteBookmark("Bookmark1");
`
```

#### Get Bookmark from document

You can get all the bookmarks in the document using [getBookmarks](#) method as shown in the following code snippet.

```
`c#
this.$refs.container.ej2Instances.documentEditor.getBookmarks(false);
`
```

Note: Parameter denotes is include hidden bookmarks. If false, ignore hidden bookmark.

#### Get Bookmark from selection

You can get bookmarks in current selection in the document using [getBookmarks](#) method as shown in the following code snippet.

```
`csharp
this.$refs.container.ej2Instances.documentEditor.selection.getBookmarks(false);
`
```

#### Replace bookmark content

You can replace bookmark content without removing the bookmark start and end for backtracking the bookmark content.

```
`csharp
```

```
this.$refs.container.ej2Instances.documentEditor.selection.selectBookmark("Bookmark1", true);
this.$refs.container.ej2Instances.documentEditor.editor.insertText('Hello World')
`
```

You can replace content by removing the bookmark start and end, thus the bookmark content can't be tracked in future.

```
`csharp
```

```
this.$refs.container.ej2Instances.documentEditor.selection.selectBookmark("Bookmark1");
this.$refs.container.ej2Instances.documentEditor.editor.insertText('Hello World')
`
```

### Show or Hide bookmark

You can show or hide the show square brackets around bookmarked items in Document editor component.

The following example code illustrates how to show or hide square brackets around bookmarked items.

```
`typescript
```

```
this.$refs.container.ej2Instances.documentEditorSettings.showBookmarks = true;
`
```

### Bookmark Dialog

The following example shows how to open bookmark dialog in Document Editor.

#### APP.VUE

```
<template>
  <div id="app">
    <div>
      <button v-on:click='showBookmarkDialog'>Open dialog</button>
    </div>
    <ejs-documenteditor ref="documenteditor" :enableSelection='true'
:isReadOnly='false' :enableEditor='true' :enableEditorHistory='true'
:enableBookmarkDialog='true' height="370px" style="width: 100%;"></ejs-
documenteditor>
  </div>
</template>
<script>
  import Vue from 'vue'
  import { DocumentEditorPlugin, Selection, Editor, SfdtExport,
EditorHistory, BookmarkDialog } from '@syncfusion/ej2-vue-documenteditor';
  Vue.use(DocumentEditorPlugin);
  export default {
    data: function() {
      return {
      };
    },
    provide: {
      //Inject require modules.
```

```

        DocumentEditor : [Selection, Editor, BookmarkDialog ,
EditorHistory, SfdtExport]
    }
    methods: {
        showBookmarkDialog: function() {
            //Open bookmark dialog.
            this.$refs.documenteditor.showDialog('Bookmark');
        }
    }
}
</script>
<style>
    @import "../node_modules/@syncfusion/ej2-vue-
documenteditor/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/document-editor/bookmark-cs1" %}

See Also

- [Feature modules](#)
- [Bookmark dialog](#)

### Link in Vue Document editor component

Document Editor supports hyperlink field. You can link a part of the document content to Internet or file location, mail address, or any text within the document.

#### Navigate a hyperlink

Document Editor triggers 'requestNavigate' event whenever user clicks Ctrl key or tap a hyperlink within the document. This event provides necessary details about link type, navigation URL, and local URL (if any) as arguments, and allows you to easily customize the hyperlink navigation functionality.

#### Add the requestNavigate event for DocumentEditor

The following example illustrates how to add requestNavigate event for DocumentEditor.

#### APP.VUE

```

<template>
    <div id="app">
        <ejs-documenteditor ref="documenteditor" :enableSelection='true'
:isReadOnly='false' v-bind:requestNavigate="onRequestNavigate"
height="370px" style="width: 100%;"></ejs-documenteditor>
    </div>
</template>
<script>
    import Vue from 'vue'
    import { DocumentEditorPlugin, Selection, RequestNavigateEventArgs }
from '@syncfusion/ej2-vue-documenteditor';
    Vue.use(DocumentEditorPlugin);
    export default {
        data: function() {
            return {
            };
        },
        provide: {

```

```

        DocumentEditor : [Selection]
    }
    methods: {
        onRequestNavigate: function(args: RequestNavigateEventArgs) {
            if (args.linkType !== 'Bookmark') {
                let link: string = args.navigationLink;
                if (args.localReference.length > 0) {
                    link += '#' + args.localReference;
                }
                window.open(link);
                args.isHandled = true;
            }
        }
    }
}
</script>
<style>
    @import "../node_modules/@syncfusion/ej2-vue-
documenteditor/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/document-editor/link-cs1" %}

[Add the requestNavigate event for DocumentEditorContainer component](#)

The following example illustrates how to add requestNavigate event for DocumentEditorContainer component.

,

```

<template>
<div id="app">
<ejs-documenteditorcontainer ref='container' :serviceUrl='serviceUrl' v-on:created="onCreated"
height="590px" id='container' :enableToolbar='true'></ejs-documenteditorcontainer>
</div>
</template>
<script>
import Vue from 'vue'

import { DocumentEditorContainerPlugin, Selection, Editor, RequestNavigateEventArgs, Toolbar } from
'@syncfusion/ej2-vue-documenteditor';

Vue.use(DocumentEditorContainerPlugin);

export default {
data: function() {
return { serviceUrl:'https://ej2services.syncfusion.com/production/web-services/api/documenteditor/'
};
},
provide: {

```

```

DocumentEditorContainer: [Toolbar],
},
methods: {
  onCreated() {
    // Add event listener for requestNavigate event to customize hyperlink navigation functionality
    this.$refs.container.ej2Instances.documentEditor.requestNavigate = function (args) {
      if (args.linkType !== 'Bookmark') {
        let link: string = args.navigationLink;
        if (args.localReference.length > 0) {
          link += '#' + args.localReference;
        }
        window.open(link);
        args.isHandled = true;
      }
    }
  }
}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-vue-documenteditor/styles/material.css";
</style>
`

```

If the selection is in hyperlink, trigger this event by calling `navigateHyperlink` method of `Selection` instance. Refer to the following example.

```
documenteditor.selection.navigateHyperlink();
```

#### Copy link

Document Editor copies link text of a hyperlink field to the clipboard if the selection is in hyperlink. Refer to the following example.

```
documenteditor .selection.copyHyperlink();
```

#### Add hyperlink

To create a basic hyperlink in the document, press `ENTER` / `SPACEBAR` / `SHIFT + ENTER` / `TAB` key after typing the address, for instance <http://www.google.com>. Document Editor automatically converts this address to a hyperlink field. The text can be considered as a valid URL if it starts with any of the following.

``http://`<br>```https://`<br>``file:///<br>``www.<br>``mailto:<br>`

Refer to the following example.

#### **APP.VUE**

```

<template>
  <div id="app">
    <ejs-documenteditor ref="documenteditor" :enableSelection='true'
    :isReadOnly='false' :enableEditor='true' v-
    bind:requestNavigate="onRequestNavigate" height="370px" style="width:
    100%;"></ejs-documenteditor>
  </div>
</template>
<script>
  import Vue from 'vue'
  import { DocumentEditorPlugin, Selection, Editor,
  RequestNavigateEventArgs } from '@syncfusion/ej2-vue-documenteditor';
  Vue.use(DocumentEditorPlugin);
  export default {
    data: function() {
      return {
      };
    },
    provide: {
      DocumentEditor : [Selection, Editor]
    },
    methods: {
      onRequestNavigate: function(args: RequestNavigateEventArgs) {
        if (args.linkType !== 'Bookmark') {
          let link: string = args.navigationLink;
          if (args.localReference.length > 0) {
            link += '#' + args.localReference;
          }
          window.open(link);
          args.isHandled = true;
        }
      }
    }
  }
</script>
<style>
  @import "../node_modules/@syncfusion/ej2-vue-
  documenteditor/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/document-editor/link-cs2" %}

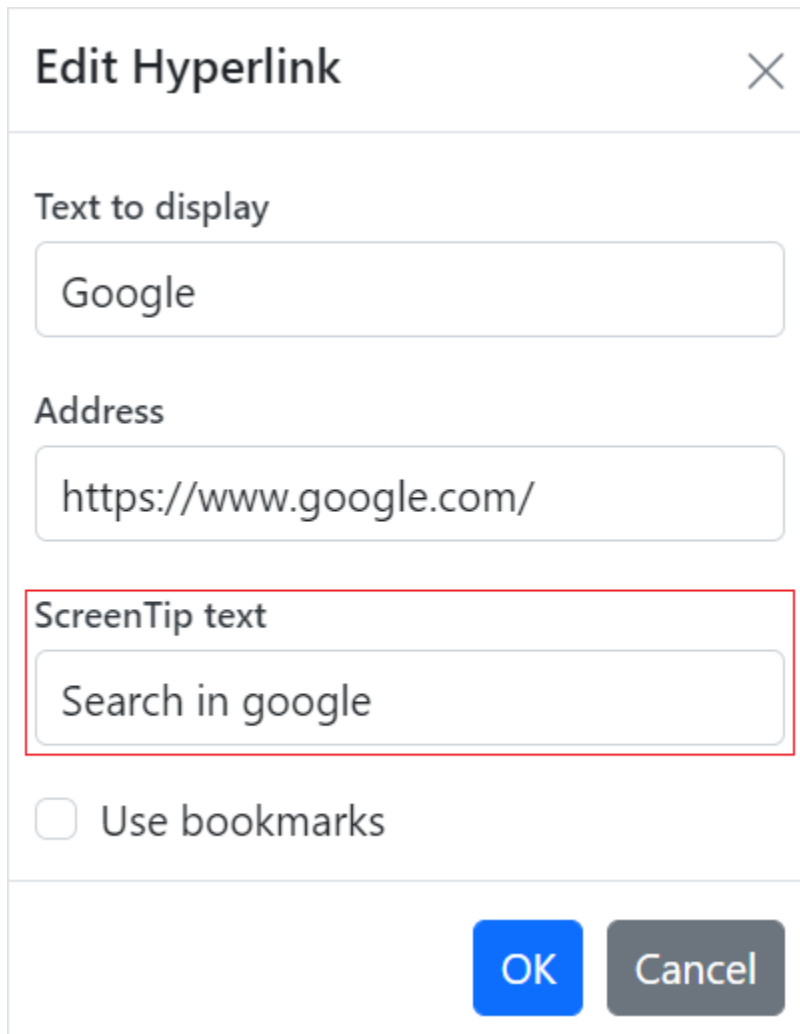
### Customize screen tip

You can customize the screen tip text for the hyperlink by using below sample code.

```
`ts
```

```
this.$refs.documenteditor.ej2Instances.insertHyperlink('https://www.google.com', 'Google', '<<Screen  
tip text>>');
```

Screen tip text can be modified through UI by using the [Hyperlink dialog](#)



### Remove hyperlink

To remove link from hyperlink in the document, press Backspace key at the end of a hyperlink. By removing the link, it will be converted as plain text. You can use `removeHyperlink` method of `Editor` instance if the selection is in hyperlink. Refer to the following example.

```
this.$refs.documenteditor.ej2Instances.editor.removeHyperlink();
```

### Hyperlink dialog

Document Editor provides dialog support to insert or edit a hyperlink. Refer to the following example.



**APP.VUE**

```

<template>
  <div id="app" style="height:330px">
    <div>
      <button v-on:click='showHyperlinkDialog' >Open
dialog</button>
    </div>
    <ejs-documenteditor ref="documenteditor" :enableSelection='true'
:isReadOnly='false' :enableEditor='true' :enableEditorHistory='true'
:enableHyperlinkDialog='true' :enableSfdtExport='true' height="370px"
style="width: 100%;"></ejs-documenteditor>
  </div>
</template>
<script>
  import Vue from 'vue'
  import { DocumentEditorPlugin, Selection, Editor, EditorHistory,
HyperlinkDialog, SfdtExport, RequestNavigateEventArgs } from
'@syncfusion/ej2-vue-documenteditor';
  Vue.use(DocumentEditorPlugin);
  export default {
    data: function() {
      return {
      };
    },
    provide: {
      DocumentEditor : [Selection, Editor, EditorHistory,
HyperlinkDialog, SfdtExport]
    },
    methods: {
      showHyperlinkDialog: function() {
        this.$refs.documenteditor.showDialog('Hyperlink');
      }
    }
  }
</script>
<style>
  @import "../node_modules/@syncfusion/ej2-vue-
documenteditor/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/document-editor/link-cs3" %}

You can use the following keyboard shortcut to open the hyperlink dialog if the selection is in hyperlink.

Key Combination	Description
-----------------	-------------

----- -----	
-------------	--

Ctrl + K	Open hyperlink dialog that allows you to create or edit hyperlink
----------	-------------------------------------------------------------------

See Also

- [Feature modules](#)
- [Hyperlink dialog](#)

## Table in Vue Document editor component

Tables are an efficient way to present information. Document Editor can display and edit the tables. You can select and edit tables through keyboard, mouse, or touch interactions. Document Editor exposes a rich set of APIs to perform these operations programmatically.

### Create a table

You can create and insert a table at cursor position by specifying the required number of rows and columns.

Refer to the following sample code.

```
`ts
this.$refs.documenteditor.ej2Instances.editor.insertTable(3,3);
`
```

The maximum size of row and column is limited to 32767 and 63 respectively.

### Insert rows

You can add a row (or several rows) above or below the row at cursor position by using the [insertRow](#) method. This method accepts the following parameters:

Parameter | Type | Description

left(optional) | boolean | This is optional and if omitted, it takes the value as false and inserts to the right of column at cursor position.

count(optional) | number | This is optional and if omitted, it takes the value as 1.

Refer to the following sample code.

```
`ts
//Insert a column to the right of the column at cursor position.
this.$refs.documenteditor.ej2Instances.editor.insertColumn();
//Insert a column to the left of the column at cursor position.
this.$refs.documenteditor.ej2Instances.editor.insertColumn(false);
//Insert two columns to the left of the column at cursor position.
this.$refs.documenteditor.ej2Instances.editor.insertColumn(false, 2);
`
```

### Select an entire table

If the cursor position is inside a table, you can select the entire table by using the following sample code.

```
`ts
this.$refs.documenteditor.ej2Instances.selection.selectTable();
`
```

### Select row

You can select the entire row at cursor position by using the following sample code.

```
`ts
```

```
this.$refs.documenteditor.ej2Instances.selection.selectRow();  
`
```

If current selection spans across cells of different rows, all these rows will be selected.

#### Select column

You can select the entire column at cursor position by using the following sample code.

```
`ts  
this.$refs.documenteditor.ej2Instances.selection.selectColumn();  
`
```

If current selection spans across cells of different columns, all these columns will be selected.

#### Select cell

You can select the cell at cursor position by using the following sample code.

```
`ts  
this.$refs.documenteditor.ej2Instances.selection.selectCell();  
`
```

#### Delete table

Document Editor allows you to delete the entire table. You can use the [deleteTable\(\)](#) method of editor instance, if selection is in table. Refer to the following sample code.

```
`ts  
this.$refs.documenteditor.ej2Instances.editor.deleteTable();  
`
```

#### Delete row

Document Editor allows you to delete the selected number of rows. You can use the [deleteRow\(\)](#) method of editor instance to delete the selected number of rows, if selection is in table. Refer to the following sample code.

```
`ts  
this.$refs.documenteditor.ej2Instances.editor.deleteRow();  
`
```

#### Delete column

Document Editor allows you to delete the selected number of columns. You can use the [deleteColumn\(\)](#) method of editor instance to delete the selected number of columns, if selection is in table. Refer to the following sample code.

```
`ts  
this.$refs.documenteditor.ej2Instances.editor.deleteColumn();  
`
```

### Merge cells

You can merge cells vertically, horizontally, or combination of both to a single cell. To vertically merge the cells, the columns within selection should be even in left and right directions. To horizontally merge the cells, the rows within selection should be even in top and bottom direction.

Refer to the following sample code.

```
`ts
this.$refs.documenteditor.ej2Instances.editor.mergeCells()
`
```

### Positioning the table

Document Editor preserves the position properties of the table and displays the table based on position properties. It does not support modifying the position properties. Whereas the table will be automatically moved along with text edited if it is positioned relative to the paragraph.

### How to work with tables

The following sample demonstrates how to delete the table row or columns, merge cells and how to bind the API with button.

```
<template>
<div id="app">
<div>
<ejs-toolbar v-bind:clicked='toolbarClickHandler'>
<e-items>
<e-item prefixIcon="e-de-icon-Table" tooltipText="Insert Table" id="table"></e-item>
<e-item type="Separator"></e-item>
<e-item prefixIcon="e-de-icon-InsertAbove" tooltipText="Insert new row above" id="insert_above"></e-item>
<e-item prefixIcon="e-de-icon-InsertBelow" tooltipText="Insert new row below" id="insert_below"></e-item>
<e-item type="Separator"></e-item>
<e-item prefixIcon="e-de-icon-InsertLeft" tooltipText="Insert new column to the left" id="insert_left"></e-item>
<e-item prefixIcon="e-de-icon-InsertRight" tooltipText="Insert new column to the right" id="insert_right"></e-item>
<e-item type="Separator"></e-item>
<e-item prefixIcon="e-de-icon-DeleteTable" tooltipText="Delete Entire table" id="delete_table"></e-item>
<e-item prefixIcon="e-de-icon-DeleteRows" tooltipText="Delete the selected row" id="delete_row"></e-item>
```

```

<e-item prefixIcon="e-de-icon-DeleteColumns" tooltipText="Delete the selected column"
id="delete_column"></e-item>
<e-item type="Separator"></e-item>
<e-item prefixIcon="e-de-icon-Cell" tooltipText="Merge the selected cells" id="merge_cell"></e-item>
<e-item type="Separator"></e-item>
<e-item text="Dialog" tooltipText="Open insert table dialog" id="table_dialog"></e-item>
</e-items>
</ejs-toolbar>
</div>

<ejs-documenteditor ref="documenteditor" :isReadOnly='false' :enableEditor='true'
:enableEditorHistory='true' :enableTableDialog='true' :enableSfdtExport='true'
:enableContextMenu='true' height="370px" style="width: 100%;"></ejs-documenteditor>
</div>
</template>
<script>
import Vue from 'vue'

import { DocumentEditorPlugin, Editor, Selection, EditorHistory, TableDialog, ContextMenu, SfdtExport }
from '@syncfusion/ej2-vue-documenteditor';

import { ToolbarPlugin } from "@syncfusion/ej2-vue-navigations";

Vue.use(DocumentEditorPlugin);
Vue.use(ToolbarPlugin);

export default {
  data: function () {
    return {
    };
  },
  provide: {
    DocumentEditor: [Editor, Selection, EditorHistory, TableDialog, ContextMenu, SfdtExport]
  },
  methods: {
    toolbarClickHandler :function(args){
      switch (arg.item.id) {
        case 'table':
          //Insert table API to add table

```

```

this.$refs.documenteditor.ej2Instances.editor.insertTable(3, 2);
break;
case 'insert_above':
//Insert the specified number of rows to the table above to the row at cursor position
this.$refs.documenteditor.ej2Instances.editor.insertRow(true, 2);
break;
case 'insert_below':
//Insert the specified number of rows to the table below to the row at cursor position
this.$refs.documenteditor.ej2Instances.editor.insertRow();
break;
case 'insert_left':
//Insert the specified number of columns to the table left to the column at cursor position
this.$refs.documenteditor.ej2Instances.editor.insertColumn(true, 2);
break;
case 'insert_right':
//Insert the specified number of columns to the table right to the column at cursor position
this.$refs.documenteditor.ej2Instances.editor.insertColumn();
break;
case 'delete_table':
//Delete the entire table
this.$refs.documenteditor.ej2Instances.editor.deleteTable();
break;
case 'delete_row':
//Delete the selected number of rows
this.$refs.documenteditor.ej2Instances.editor.deleteRow();
break;
case 'delete_column':
//Delete the selected number of columns
this.$refs.documenteditor.ej2Instances.editor.deleteColumn();
break;
case 'merge_cell':
//Merge the selected cells into one (both vertically and horizontally)
this.$refs.documenteditor.ej2Instances.editor.mergeCells();

```

```
break;
case 'table_dialog':
  //Opens insert table dialog
  this.$refs.documenteditor.showDialog('Table');
break;
}
}
},
mounted() {
  this.$refs.documenteditor.ej2Instances.editor.insertTable(2, 2);
}
}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-vue-documenteditor/styles/material.css";
</style>
`
```

See Also

- [Feature modules](#)
- [Insert table dialog](#)

## Table of contents in Vue Document editor component

The table of contents in a document is same as the list of chapters at the beginning of a book. It lists each heading in the document and the page number, where that heading starts with various options to customize the appearance.

### Inserting table of contents

Document Editor exposes an API to insert table of contents at cursor position programmatically. You can specify the settings for table of contents explicitly. Otherwise, the default settings will be applied.

[TableOfContentsSettings](#) contain the following properties:

- **startLevel:** Specifies the start level for constructing table of contents.
- **endLevel:** Specifies the end level for constructing table of contents.
- **includeHyperlink:** Specifies whether the link for headings is included.
- **includePageNumber:** Specified whether the page number of the headings is included.
- **rightAlign:** Specifies whether the page number is right aligned.
- **tabLeader:** Specifies the tab leader styles such as none, dot, hyphen, and underscore.
- **includeOutlineLevels:** Specifies whether the outline levels are included.

The following code illustrates how to insert table of content in Document Editor.

```
`ts
```

```
let tocSettings: TableOfContentsSettings =
```

```
{
```

```
startLevel: 1, endLevel: 3, includeHyperlink: true, includePageNumber: true, rightAlign: true
```

```
};
```

```
this.$refs.documenteditor.ej2Instances.editor.insertTableOfContents(tocSettings);
```

```
,
```

## APP.VUE

```
<template>
  <div style="width:100%;height:330px">
    <div>
      <button v-on:click='onInsertToc'>Insert TOC</button>
    </div>
    <ejs-documenteditor ref="documenteditor" :isReadOnly='true' v-
bind:created='onCreated' :enableSelection='true' :enableEditor='true'
height="370px" style="width: 100%;"></ejs-documenteditor>
  </div>
</template>
<script>
  import Vue from 'vue'
  import { DocumentEditorPlugin, Selection, Editor,
TableOfContentsSettings } from '@syncfusion/ej2-vue-documenteditor';
  Vue.use(DocumentEditorPlugin);
  export default {
    data: function() {
      return {
      };
    },
    provide: {
      DocumentEditor : [Selection, Editor]
    },
    methods: {
      onCreated: function(args) {
        let sfdt: string =
'{"sections":[{"blocks":[{"paragraphFormat":{"styleName":"Heading
1"},"inlines":[{"text":"Headin"}, {"name":"_GoBack","bookmarkType":0}, {"name":
"_GoBack","bookmarkType":1}, {"text":"g1"}]}], {"paragraphFormat":{"styleName":
"Heading
2"},"inlines":[{"text":"Heading2"}]}, {"paragraphFormat":{"styleName":"Headin
g
3"},"inlines":[{"text":"Heading3"}]}, {"paragraphFormat":{"styleName":"Headin
g
4"},"inlines":[{"text":"Heading4"}]}, {"paragraphFormat":{"styleName":"Headin
g
5"},"inlines":[{"text":"Heading5"}]}, {"paragraphFormat":{"styleName":"Headin
g
6"},"inlines":[{"text":"Heading6"}]}, {"paragraphFormat":{"styleName":"Normal
"},"inlines":[{"text":"Normal"}]}], "headersFooters": {}, "sectionFormat": {"hea
derDistance":36.0, "footerDistance":36.0, "pageWidth":612.0, "pageHeight":792.0
```



```

, "leftMargin": 72.0, "rightMargin": 72.0, "topMargin": 72.0, "bottomMargin": 72.0, "
differentFirstPage": false, "differentOddAndEvenPages": false } } ], "characterForm
at": { "fontSize": 11.0, "fontFamily": "Calibri", "paragraphFormat": { "afterSpacin
g": 8.0, "lineSpacing": 1.0791666507720947, "lineSpacingType": "Multiple", "backg
round": { "color": "#FFFFFF", "styles": [ { "type": "Paragraph", "name": "Normal", "
next": "Normal", }, { "type": "Paragraph", "name": "Heading
1", "basedOn": "Normal", "next": "Normal", "link": "Heading 1
Char", "characterFormat": { "fontSize": 16.0, "fontFamily": "Calibri
Light", "fontColor": "#2F5496FF", "paragraphFormat": { "beforeSpacing": 12.0, "aft
erSpacing": 0.0, "outlineLevel": "Level1", }, { "type": "Paragraph", "name": "Heading
2", "basedOn": "Normal", "next": "Normal", "link": "Heading 2
Char", "characterFormat": { "fontSize": 13.0, "fontFamily": "Calibri
Light", "fontColor": "#2F5496FF", "paragraphFormat": { "beforeSpacing": 2.0, "afte
rSpacing": 0.0, "outlineLevel": "Level2", }, { "type": "Paragraph", "name": "Heading
3", "basedOn": "Normal", "next": "Normal", "link": "Heading 3
Char", "characterFormat": { "fontSize": 12.0, "fontFamily": "Calibri
Light", "fontColor": "#1F3763FF", "paragraphFormat": { "beforeSpacing": 2.0, "afte
rSpacing": 0.0, "outlineLevel": "Level3", }, { "type": "Paragraph", "name": "Heading
4", "basedOn": "Normal", "next": "Normal", "link": "Heading 4
Char", "characterFormat": { "italic": true, "fontFamily": "Calibri
Light", "fontColor": "#2F5496FF", "paragraphFormat": { "beforeSpacing": 2.0, "afte
rSpacing": 0.0, "outlineLevel": "Level4", }, { "type": "Paragraph", "name": "Heading
5", "basedOn": "Normal", "next": "Normal", "link": "Heading 5
Char", "characterFormat": { "fontFamily": "Calibri
Light", "fontColor": "#2F5496FF", "paragraphFormat": { "beforeSpacing": 2.0, "afte
rSpacing": 0.0, "outlineLevel": "Level5", }, { "type": "Paragraph", "name": "Heading
6", "basedOn": "Normal", "next": "Normal", "link": "Heading 6
Char", "characterFormat": { "fontFamily": "Calibri
Light", "fontColor": "#1F3763FF", "paragraphFormat": { "beforeSpacing": 2.0, "afte
rSpacing": 0.0, "outlineLevel": "Level6", }, { "type": "Character", "name": "Default
Paragraph Font", }, { "type": "Character", "name": "Heading 1
Char", "basedOn": "Default Paragraph
Font", "characterFormat": { "fontSize": 16.0, "fontFamily": "Calibri
Light", "fontColor": "#2F5496FF", }, { "type": "Character", "name": "Heading 2
Char", "basedOn": "Default Paragraph
Font", "characterFormat": { "fontSize": 13.0, "fontFamily": "Calibri
Light", "fontColor": "#2F5496FF", }, { "type": "Character", "name": "Heading 3
Char", "basedOn": "Default Paragraph
Font", "characterFormat": { "fontSize": 12.0, "fontFamily": "Calibri
Light", "fontColor": "#1F3763FF", }, { "type": "Character", "name": "Heading 4
Char", "basedOn": "Default Paragraph
Font", "characterFormat": { "italic": true, "fontFamily": "Calibri
Light", "fontColor": "#2F5496FF", }, { "type": "Character", "name": "Heading 5
Char", "basedOn": "Default Paragraph
Font", "characterFormat": { "fontFamily": "Calibri
Light", "fontColor": "#2F5496FF", }, { "type": "Character", "name": "Heading 6
Char", "basedOn": "Default Paragraph
Font", "characterFormat": { "fontFamily": "Calibri
Light", "fontColor": "#1F3763FF", } } } ] }';

    this.$refs.documenteditor.open(sfdd);
  },
  onInsertToc: function() {
    //Table of contents settings.
    let tocSettings: TableOfContentsSettings =
    {
      startLevel: 1, endLevel: 3, includeHyperlink: true,
      includePageNumber: true, rightAlign: true
    }
  }
}

```

```

    };
    //Insert table of contents.

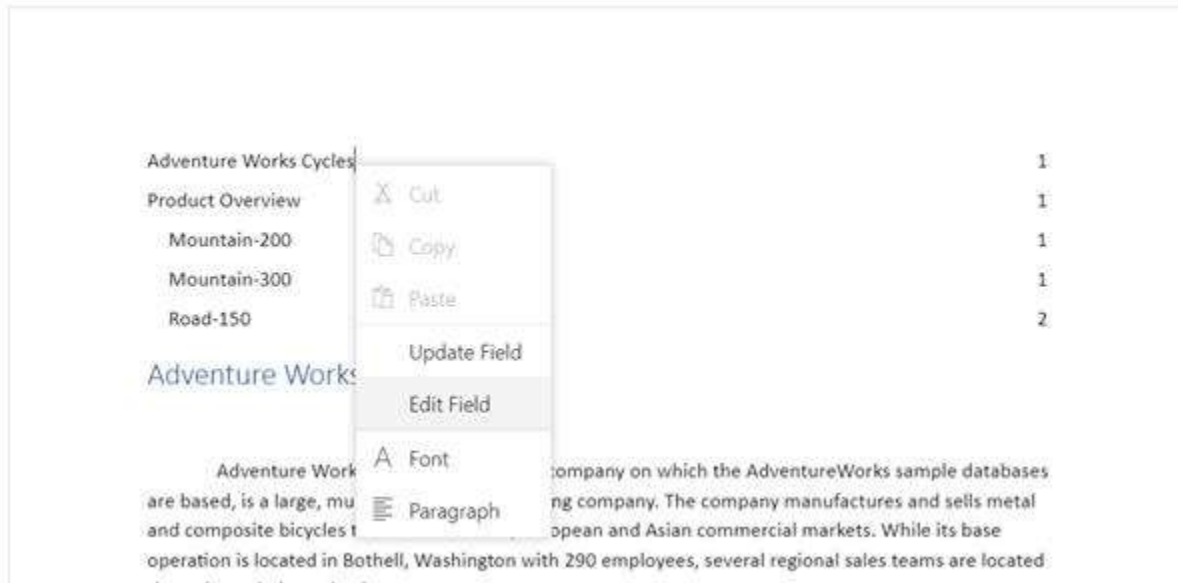
    this.$refs.documenteditor.ej2Instances.editor.insertTableOfContents(tocSettings);
  }
}
</script>
<style>
  @import "../node_modules/@syncfusion/ej2-vue-documenteditor/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/document-editor/export-cs8" %}

### Update or edit table of contents

You can update or edit the table of contents using the built-in context menu shown up by right-clicking it. Refer to the following screenshot.



- **Update Field:** Updates the headings in table of contents with same settings by searching the entire document.
- **Edit Field:** Opens the built-in table of contents dialog and allows you to modify its settings.

You can also do it programmatically by using the exposed API. Refer to the following sample code.

```

`ts
let tocSettings: TableOfContentsSettings =
{
  startLevel: 1, endLevel: 3, includeHyperlink: true, includePageNumber: true, rightAlign: true
};

```

```
this.$refs.documenteditor.ej2Instances.editor.insertTableOfContents(tocSettings);  
`
```

Same method is used for inserting, updating, and editing table of contents. This will work based on the current element at cursor position and the optional settings parameter. If table of contents is present at cursor position, the update operation will be done based on the optional settings parameter. Otherwise, the insert operation will be done.

See Also

- [Table of contents dialog](#)

### Header footer in Vue Document editor component

Document editor supports headers and footers in its document. Each section in the document can have the following types of headers and footers:

- First page: Used only on the first page of the section.
- Even pages: Used on all even numbered pages in the section.
- Default: Used on all pages of the section, where first or even pages are not applicable or not specified.

You can define this by setting format properties of the corresponding section using the following sample code.

```
`javascript  
//Defines whether different header footer is required for first page of the section  
this.$refs.documenteditor.ej2Instances.selection.sectionFormat.differentFirstPage= true;  
//Defines whether different header footer is required for odd and even pages in the section  
this.$refs.documenteditor.ej2Instances.selection.sectionFormat.differentOddAndEvenPages= true;  
`
```

### Go to header footer region

Double click in header or footer region to move the selection into it. You can also do this by using the following code.

```
`javascript  
this.$refs.documenteditor.ej2Instances.selection.goToHeader();  
`  
  
`javascript  
this.$refs.documenteditor.ej2Instances.selection.goToFooter();  
`
```

### Link to previous

Link to previous is enabled by default when document has more than one section. If you're using different headers and footers such as different first page or different odd and even pages, they can't be linked together because they're all separate.

Before setting or getting the link to previous value, use the '[goToHeader](#)' or '[goToFooter](#)' API to move the current selection to the header or footer region.

You can get or set the default header footer link to previous value of a section at cursor position by using the following sample code.

```
`ts
this.$refs.container.ej2Instances.documentEditor.selection.sectionFormat.oddPageHeader.linkToPrevious = false;

this.$refs.container.ej2Instances.documentEditor.selection.sectionFormat.oddPageFooter.linkToPrevious = false;
`
```

In case the document has different header and footer types, such as different first page, odd, and even pages.

```
`ts
// Different first page
this.$refs.container.ej2Instances.documentEditor.selection.sectionFormat.firstPageHeader.linkToPrevious = false;

this.$refs.container.ej2Instances.documentEditor.selection.sectionFormat.firstPageFooter.linkToPrevious = false;

//Even page
this.$refs.container.ej2Instances.documentEditor.selection.sectionFormat.firstPageHeader.linkToPrevious = false;

this.$refs.container.ej2Instances.documentEditor.selection.sectionFormat.firstPageFooter.linkToPrevious = false;
`
```

Note: When there is more than one section in the document, the Link to Previous option becomes available. By default, this feature is disabled state in UI and set to return false for the first section.

### Header and footer distance

You can define the distance of header region content from the top of the page. Refer to the following sample code.

```
`javascript
this.$refs.documenteditor.ej2Instances.selection.sectionFormat.headerDistance= 36;
`
```

Same way, you can define the distance of footer region content from the bottom of the page. Refer to the following sample code.

```
`javascript
```

```
this.$refs.documenteditor.ej2Instances.selection.sectionFormat.footerDistace= 36;
```

```
,
```

### Close header footer region

Move the selection to the document body from header or footer region by double clicking or tapping the document area. You can also perform this by using the following sample code.

```
`javascript
```

```
this.$refs.documenteditor.ej2Instances.selection.closeHeaderFooter()
```

```
,
```

### See Also

- [Working with Section Formatting](#)

### Text format in Vue Document editor component

Document Editor supports several formatting options for text like bold, italic, font color, highlight color, and more. This section describes how to modify the formatting for selected text in detail.

#### Bold

The bold formatting for selected text can be get or set by using the following sample code.

```
`ts
```

```
//Gets the value for bold formatting of selected text.
```

```
let bold : boolean = this.$refs.documenteditor.ej2Instances.selection.characterFormat.bold;
```

```
//Sets bold formatting for selected text.
```

```
this.$refs.documenteditor.ej2Instances.selection.characterFormat.bold = true;
```

```
,
```

You can toggle the bold formatting based on existing value at selection. Refer to the following sample code.

```
`ts
```

```
this.$refs.documenteditor.ej2Instances.editor.toggleBold();
```

```
,
```

#### Italic

The Italic formatting for selected text can be get or set by using the following sample code.

```
`ts
```

```
//Gets the value for italic formatting of selected text.
```

```
let italic : boolean = this.$refs.documenteditor.ej2Instances.selection.characterFormat.italic;
```

```
//Sets italic formatting for selected text.
```

```
this.$refs.documenteditor.ej2Instances.selection.characterFormat.italic= true|false;
```

`

You can toggle the Italic formatting based on existing value at selection. Refer to the following sample code.

```
`ts
this.$refs.documenteditor.ej2Instances.editor.toggleItalic();
```

`

### Underline property

The underline style for selected text can be get or set by using the following sample code.

```
`ts
//Gets the value for underline formatting of selected text.
let underline : Underline = this.$refs.documenteditor.ej2Instances.selection.characterFormat.underline;
//Sets underline formatting for selected text.
this.$refs.documenteditor.ej2Instances.selection.characterFormat.underline='Single' | 'None';
```

`

You can toggle the underline style of selected text based on existing value at selection by specifying a value. Refer to the following sample code.

```
`ts
this.$refs.documenteditor.ej2Instances.editor.toggleUnderline('Single');
```

`

### Strikethrough property

The strikethrough style for selected text can be get or set by using the following sample code.

```
`ts
//Gets the value for strikethrough formatting of selected text.
let strikethrough : Strikethrough =
this.$refs.documenteditor.ej2Instances.selection.characterFormat.strikethrough;
//Sets strikethrough formatting for selected text.
this.$refs.documenteditor.ej2Instances.selection.characterFormat.strikethrough='Single' | 'Normal';
```

`

You can toggle the strikethrough style of selected text based on existing value at selection by specifying a value. Refer to the following sample code.

```
`ts
this.$refs.documenteditor.ej2Instances.editor.toggleStrikethrough();
```

`

### Superscript property

The selected text can be made superscript by using the following sample code.

```
`ts
//Gets the value for baselineAlignment formatting of selected text.
let baselineAlignment : BaselineAlignment =
this.$refs.documenteditor.ej2Instances.selection.characterFormat.baselineAlignment;
//Sets baselineAlignment formatting for selected text.
this.$refs.documenteditor.ej2Instances.selection.characterFormat.baselineAlignment='Superscript';
`
```

Toggle the selected text as superscript or normal using the following sample code.

```
`ts
this.$refs.documenteditor.ej2Instances.editor.toggleSuperscript();
`
```

### Subscript property

The selected text can be made subscript by using the following sample code.

```
`ts
//Gets the value for baselineAlignment formatting of selected text.
let baselineAlignment : BaselineAlignment =
this.$refs.documenteditor.ej2Instances.selection.characterFormat.baselineAlignment;
//Sets baselineAlignment formatting for selected text.
this.$refs.documenteditor.ej2Instances.selection.characterFormat.baselineAlignment='Subscript';
`
```

Toggle the selected text as subscript or normal using the following sample code.

```
`ts
this.$refs.documenteditor.ej2Instances.editor.toggleSubscript();
`
```

You can make a subscript or superscript text as normal using the following code.

```
`ts
this.$refs.documenteditor.ej2Instances.selection.characterFormat.baselineAlignment='Normal';
`
```

### Size

The size of selected text can be get or set using the following code.

```
`ts
//Gets the value for fontSize formatting of selected text.
let fontSize : number = this.$refs.documenteditor.ej2Instances.selection.characterFormat.fontSize;
//Sets fontSize formatting for selected text.
```

```
this.$refs.documenteditor.ej2Instances.selection.characterFormat.fontSize= 32;
、
```

### Color

The color of selected text can be get or set using the following code.

```
`ts
//Gets the value for fontColor formatting of selected text.
let fontColor : string = this.$refs.documenteditor.ej2Instances.selection.characterFormat.fontColor;
//Sets fontColor formatting for selected text.
this.$refs.documenteditor.ej2Instances.selection.characterFormat.fontColor= 'Pink';
this.$refs.documenteditor.ej2Instances.selection.characterFormat.fontColor= '#FFC0CB';
、
```

### Font

The font style of selected text can be get or set using the following sample code.

```
`ts
//Gets the value for fontFamily formatting of selected text.
let baselineAlignement : string =
this.$refs.documenteditor.ej2Instances.selection.characterFormat.fontFamily;
//Sets fontFamily formatting for selected text.
this.$refs.documenteditor.ej2Instances.selection.characterFormat.fontFamily= 'Arial';
、
```

### Highlight color

The highlight color of the selected text can be get or set using the following sample code.

```
`ts
//Gets the value for highlightColor formatting of selected text.
let highlightColor : HighlightColor =
this.$refs.documenteditor.ej2Instances.selection.characterFormat.highlightColor;
//Sets highlightColor formatting for selected text.
this.$refs.documenteditor.ej2Instances.selection.characterFormat.highlightColor= 'Pink';
、
```

### Toolbar with options for text formatting

Refer to the following example.

```
、
<template>
<div id="app">
<div>
```



```

<ejs-toolbar v-bind:clicked='toolbarClickHandler'>
<e-items>
<e-item prefixIcon="e-de-icon-Bold" tooltipText="Bold" id="bold"></e-item>
<e-item prefixIcon="e-de-icon-Italic" tooltipText="Italic" id="italic"></e-item>
<e-item prefixIcon="e-de-icon-Underline" tooltipText="Underline" id="underline"></e-item>
<e-item prefixIcon="e-de-icon-Strikethrough" tooltipText="Strikethrough" id="striketrough"></e-item>
<e-item prefixIcon="e-de-icon-Subscript" tooltipText="Subscript" id="subscript"></e-item>
<e-item prefixIcon="e-de-icon-Superscript" tooltipText="Superscript" id="superscript"></e-item>
<e-item type="Seperator"></e-item>
<e-item type="Input" :template="<ejs-colorpicker :value='#000000' :showButtons='true'
v:bind:change='onFontColorChange' ></ejs-colorpicker>"></e-item>
<e-item type="Seperator"></e-item>
<e-item type="Input" :template="<ejs-combobox :dataSource='fontStyle' :width='120' :index='2'
:allowCustom='true' v:bind:change='onFontFamilyChange' :showClearButton='false'></ejs-
combobox>"></e-item>
<e-item type="Input" :template="<ejs-combobox :dataSource='fontSize' :width='80' :index='2'
:allowCustom='true' v:bind:change='onFontSizeChange' :showClearButton='false'></ejs-
combobox>"></e-item>
</e-items>
</ejs-toolbar>
</div>

<ejs-documenteditor ref="documenteditor" v-bind:selectionChange='onSelectionChange'
:isReadOnly='false' :enableEditor='true' :enableEditorHistory='true' :enableSfdtExport='true'
height="370px" style="width: 100%;"></ejs-documenteditor>
</div>
</template>
<script>
import Vue from 'vue'

import { DocumentEditorPlugin, Editor, Selection, EditorHistory , SfdtExport } from '@syncfusion/ej2-
vue-documenteditor';

import { ToolbarPlugin } from "@syncfusion/ej2-vue-navigations";
import { ColorPickerPlugin } from '@syncfusion/ej2-vue-inputs';
import { ComboBoxPlugin } from "@syncfusion/ej2-vue-dropdowns";

Vue.use(DocumentEditorPlugin);
Vue.use(ToolbarPlugin);
Vue.use(ColorPickerPlugin);

```

```
Vue.use(ComboBoxPlugin);
export default {
  data: function () {
    return {
      fontStyle:['Algerian', 'Arial', 'Calibri', 'Cambria', 'Cambria Math', 'Candara', 'Courier New', 'Georgia',
        'Impact', 'Segoe Print', 'Segoe Script', 'Segoe UI', 'Symbol', 'Times New Roman', 'Verdana', 'Windings'],
      fontSize:['8', '9', '10', '11', '12', '14', '16', '18','20', '22', '24', '26', '28', '36', '48', '72', '96']
    };
  },
  provide: {
    DocumentEditor: [Editor, Selection, EditorHistory, SfdtExport]
  },
  methods: {
    toolbarButtonClick: function(arg) {
      switch (arg.item.id) {
        case 'bold':
          //Toggles the bold of selected content
          this.$refs.documenteditor.ej2Instances.editor.toggleBold();
          break;
        case 'italic':
          //Toggles the Italic of selected content
          this.$refs.documenteditor.ej2Instances.editor.toggleItalic();
          break;
        case 'underline':
          //Toggles the underline of selected content
          this.$refs.documenteditor.ej2Instances.editor.toggleUnderline('Single');
          break;
        case 'strikethrough':
          //Toggles the strikethrough of selected content
          this.$refs.documenteditor.ej2Instances.editor.toggleStrikethrough();
          break;
        case 'subscript':
          //Toggles the subscript of selected content
```

```

this.$refs.documenteditor.ej2Instances.editor.toggleSubscript();
break;
case 'superscript':
//Toggles the superscript of selected content
this.$refs.documenteditor.ej2Instances.editor.toggleSuperscript();
break;
}
},
onFontFamilyChange: function(args) {
this.$refs.documenteditor.ej2Instances.selection.characterFormat.fontFamily = args.value;
this.$refs.documenteditor.focusIn();
},
onFontSizeChange: function(args) {
this.$refs.documenteditor.ej2Instances.selection.characterFormat.fontSize = args.value;
this.$refs.documenteditor.focusIn();
},
onFontColorChange: function(args) {
this.$refs.documenteditor.ej2Instances.selection.characterFormat.fontColor = args.currentValue.hex;
this.$refs.documenteditor.focusIn();
},
onSelectionChange: function() {
var characterformat = this.$refs.documenteditor.ej2Instances.selection.characterFormat;
var properties = [characterformat.bold, characterformat.italic, characterformat.underline,
characterformat.strikeThrough];
var toggleBtnId = ["bold", "italic", "underline", "strikethrough"];
for (var i = 0; i < properties.length; i++) {
let toggleBtn: HTMLElement = document.getElementById(toggleBtnId[i]);
if ((typeof (properties[i]) == 'boolean' && properties[i] == true) || (typeof (properties[i]) == 'string' &&
properties[i] != 'None'))
toggleBtn.classList.add("e-btn-toggle");
else {
if (toggleBtn.classList.contains("e-btn-toggle"))
toggleBtn.classList.remove("e-btn-toggle");
}
}
}

```

```

}
}
},
mounted() {
this.$refs.documenteditor.ej2Instances.editor.insertTable(2, 2);
}
}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-vue-documenteditor/styles/material.css";
</style>
`

```

See Also

- [Feature modules](#)
- [Font dialog](#)
- [Keyboard shortcuts](#)

## Paragraph format in Vue Document editor component

Document Editor supports various paragraph formatting options such as text alignment, indentation, paragraph spacing, and more.

### Indentation

You can modify the left or right indentation of selected paragraphs using the following sample code.

```

`javascript
this.$refs.documenteditor.ej2Instances.selection.paragraphFormat.leftIndent= 24;
this.$refs.documenteditor.ej2Instances.selection.paragraphFormat.rightIndent= 24;
`

```

### Special indentation

You can define special indent for first line of the paragraph using the following sample code.

```

`javascript
this.$refs.documenteditor.ej2Instances.selection.paragraphFormat.firstLineIndent= 24;
`

```

### Increase indent

You can increase the left indent of selected paragraphs by a factor of 36 points using the following sample code.

```

`javascript

```

```
this.$refs.documenteditor.ej2Instances.editor.increaseIndent()
```

```
,
```

### Decrease indent

You can decrease the left indent of selected paragraphs by a factor of 36 points using the following sample code.

```
`javascript
```

```
this.$refs.documenteditor.ej2Instances.editor.decreaseIndent()
```

```
,
```

### Text alignment

You can get or set the text alignment of selected paragraphs using the following sample code.

```
`javascript
```

```
this.$refs.documenteditor.ej2Instances.selection.paragraphFormat.textAlignment= 'Center' | 'Left' |  
'Right' | 'Justify';
```

```
,
```

You can toggle the text alignment of selected paragraphs by specifying a value using the following sample code.

```
`javascript
```

```
this.$refs.documenteditor.ej2Instances.editor.toggleTextAlignment('Center' | 'Left' | 'Right' | 'Justify');
```

```
,
```

### Line spacing and its type

You can define the line spacing and its type for selected paragraphs using the following sample code.

```
`javascript
```

```
//Set line spacing type.
```

```
this.$refs.documenteditor.ej2Instances.selection.paragraphFormat.lineSpacingType='AtLeast';
```

```
//Set line spacing.
```

```
this.$refs.documenteditor.ej2Instances.selection.paragraphFormat.lineSpacing= 6;
```

```
,
```

### Paragraph spacing

You can define the spacing before or after the paragraph by using the following sample code.

```
`javascript
```

```
//Set before spacing.
```

```
this.$refs.documenteditor.ej2Instances.selection.paragraphFormat.beforeSpacing= 24;
```

```
//Set after spacing.
```

```
this.$refs.documenteditor.ej2Instances.selection.paragraphFormat.afterSpacing= 24;
```

```
,
```

You can also set automatic spacing before and after the paragraph by using the following sample code.

```
`ts
this.$refs.documenteditor.ej2Instances.selection.paragraphFormat.spaceBeforeAuto = true;
this.$refs.documenteditor.ej2Instances.selection.paragraphFormat.spaceAfterAuto = true;
`
```

Note: If auto spacing property is enabled, then value defined in the `beforeSpacing` and `afterSpacing` property will not be considered.

### Pagination properties

You can enable or disable the following pagination properties for the paragraphs in a Word document.

- Widow/Orphan control - whether the first and last lines of the paragraph are to remain on the same page as the rest of the paragraph when paginating the document.
- Keep with next - whether the specified paragraph remains on the same page as the paragraph that follows it while paginating the document.
- Keep lines together - whether all lines in the specified paragraphs remain on the same page while paginating the document.

The following example code illustrates how to enable or disable these pagination properties for the selected paragraphs.

```
`javascript
this.$refs.documenteditor.selection.paragraphFormat.widowControl = false;
this.$refs.documenteditor.selection.paragraphFormat.keepWithNext = true;
this.$refs.documenteditor.selection.paragraphFormat.keepLinesTogether = true;
`
```

### Paragraph Border

You can apply borders to the paragraphs in a Word document. Using borders, decorate the paragraphs to set them apart from other paragraphs in the document.

The following example code illustrates how to apply box border for the selected paragraphs.

```
`ts
// left
this.$refs.documenteditor.selection.paragraphFormat.borders.left.lineStyle = 'Single';
this.$refs.documenteditor.selection.paragraphFormat.borders.left.lineWidth = 3;
this.$refs.documenteditor.selection.paragraphFormat.borders.left.color = "#000000";
//right
this.$refs.documenteditor.selection.paragraphFormat.borders.right.lineStyle = 'Single';
this.$refs.documenteditor.selection.paragraphFormat.borders.right.lineWidth = 3;
this.$refs.documenteditor.selection.paragraphFormat.borders.right.color = "#000000";
```

```
//top
this.$refs.documenteditor.selection.paragraphFormat.borders.top.lineStyle = 'Single';
this.$refs.documenteditor.selection.paragraphFormat.borders.top.lineWidth = 3;
this.$refs.documenteditor.selection.paragraphFormat.borders.top.color = "#000000";
//bottom
this.$refs.documenteditor.selection.paragraphFormat.borders.bottom.lineStyle = 'Single';
this.$refs.documenteditor.selection.paragraphFormat.borders.bottom.lineWidth = 3;
this.$refs.documenteditor.selection.paragraphFormat.borders.bottom.color = "#000000";
`
```

Note: At present, the Document editor component displays all the border styles as single line. But you can apply any border style and get the proper display in Microsoft Word app when opening the exported Word document.

### Show or Hide Paragraph marks

You can show or hide the hidden formatting symbols like spaces, tab, paragraph marks, and breaks in Document editor component. These marks help identify the start and end of a paragraph and all the hidden formatting symbols in a Word document.

The following example code illustrates how to show or hide paragraph marks.

```
`ts
this.$refs.documenteditor.documentEditorSettings.showHiddenMarks = true;
`
```

### Toolbar with paragraph formatting options

The following sample demonstrates the paragraph formatting options using a toolbar.

```
<template>
<div id="app" style="height:330px">
<div>
<ejs-toolbar v-bind:clicked='toolbarButtonClick'>
<e-items>
<e-item prefixIcon='e-de-icon-AlignLeft' id='AlignLeft' tooltipText='Align Left'></e-item>
<e-item prefixIcon='e-de-icon-AlignCenter' id='Align Center' tooltipText='AlignCenter'></e-item>
<e-item prefixIcon='e-de-icon-AlignRight' id='Align Right' tooltipText='AlignRight'></e-item>
<e-item prefixIcon='e-de-icon-Justify' id='Justify' tooltipText='Justify'></e-item>
<e-item prefixIcon='e-de-icon-IncreaseIndent' id='IncreaseIndent'
tooltipText='Increase Indent'></e-item>
<e-item prefixIcon='e-de-icon-DecreaseIndent' id='DecreaseIndent'
```

```

tooltipText='Decrease Indent'></e-item>
<e-item type='Separator'></e-item>
<e-item prefixIcon='e-de-icon-ClearAll' id='ClearFormat' tooltipText='ClearFormatting'></e-item>
<e-item type='Separator'></e-item>
<e-item prefixIcon='e-de-e-paragraph-mark e-icons' id='ShowParagraphMark'
tooltipText='Show the hidden characters like spaces, tab, paragraph marks, and breaks.(Ctrl + *)'></e-
item>
</e-items>
</ejs-toolbar>
</div>

<ejs-documenteditor ref="documenteditor" v-bind:selectionChange='onSelectionChange'
:enableSelection='true'
:isReadOnly='false' :enableEditor='true' :enableEditorHistory='true' :enableSfdtExport='true'
:enableContextMenu='true' height="370px" style="width: 100%;"></ejs-documenteditor>
</div>
</template>
<script>
import Vue from 'vue'

import { DocumentEditorPlugin, Editor, Selection, EditorHistory, SfdtExport, ContextMenu } from
'@syncfusion/ej2-vue-documenteditor';

import { ToolbarPlugin } from "@syncfusion/ej2-vue-navigations";

Vue.use(DocumentEditorPlugin);
Vue.use(ToolbarPlugin);

export default {
data: function () {
return {
};
},
provide: {
DocumentEditor: [Editor, Selection, EditorHistory, SfdtExport, ContextMenu]
},
methods: {
toolbarButtonClick: function (arg) {
switch (arg.item.id) {

```



```

case 'AlignLeft':
//Toggle the Left alignment for selected or current paragraph
this.$refs.documenteditor.ej2Instances.editor.toggleTextAlignment('Left');
break;
case 'AlignRight':
//Toggle the Right alignment for selected or current paragraph
this.$refs.documenteditor.ej2Instances.editor.toggleTextAlignment('Right');
break;
case 'AlignCenter':
//Toggle the Center alignment for selected or current paragraph
this.$refs.documenteditor.ej2Instances.editor.toggleTextAlignment('Center');
break;
case 'Justify':
//Toggle the Justify alignment for selected or current paragraph
this.$refs.documenteditor.ej2Instances.editor.toggleTextAlignment('Justify');
break;
case 'IncreaseIndent':
//Increase the left indent of selected or current paragraph
this.$refs.documenteditor.ej2Instances.editor.increaseIndent();
break;
case 'DecreaseIndent':
//Decrease the left indent of selected or current paragraph
this.$refs.documenteditor.ej2Instances.editor.decreaseIndent();
break;
case 'ClearFormat':
this.$refs.documenteditor.ej2Instances.editor.clearFormatting();
break;
case 'ShowParagraphMark':
//Show or hide the hidden characters like spaces, tab, paragraph marks, and breaks.
this.$refs.documenteditor.ej2Instances.documentEditorSettings.showHiddenMarks =
!this.$refs.documenteditor.ej2Instances.documentEditorSettings.showHiddenMarks;
break;
}

```

```

},
onSelectionChange: function () {
  if (this.$refs.documenteditor.ej2Instances.selection) {
    var paragraphFormat = this.$refs.documenteditor.ej2Instances.selection.paragraphFormat;
    var toggleBtnId = ['AlignLeft', 'AlignCenter', 'AlignRight', 'Justify', 'ShowParagraphMark'];
    for (var i = 0; i < toggleBtnId.length; i++) {
      let toggleBtn: HTMLElement = document.getElementById(toggleBtnId[i]);
      //Remove toggle state.
      toggleBtn.classList.remove('e-btn-toggle');
    }
    //Add toggle state based on selection paragraph format.
    if (paragraphFormat.textAlignment === 'Left') {
      document.getElementById('AlignLeft').classList.add('e-btn-toggle');
    } else if (paragraphFormat.textAlignment === 'Right') {
      document.getElementById('AlignRight').classList.add('e-btn-toggle');
    } else if (paragraphFormat.textAlignment === 'Center') {
      document.getElementById('AlignCenter').classList.add('e-btn-toggle');
    } else {
      document.getElementById('Justify').classList.add('e-btn-toggle');
    }
    if (this.$refs.documenteditor.ej2Instances.documentEditorSettings.showHiddenMarks) {
      document.getElementById('ShowParagraphMark').classList.add('e-btn-toggle');
    }
  }
}
}
}
}
}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-vue-documenteditor/styles/material.css";
</style>

```

## See Also

- [Feature modules](#)
- [Paragraph dialog](#)
- [Keyboard shortcuts](#)

## Styles in Vue Document editor component

Styles are useful for applying a set of formatting consistently throughout the document. In document editor, styles are created and added to a document programmatically or via the built-in Styles dialog.

### Styles definition overview

A Style in document editor should have the following properties:

- **name**: Name of the style. All styles in a document have a unique name, which is used as an identifier when applying the style.
- **type**: Specifies the document elements that the style will target. For example, paragraph or character.
- **next**: Specifies that the current style inherits the style set to this property. This is how hierarchical styles are defined.
- **link**: Provides a relation between the paragraph and character style.
- **characterFormat**: Specifies the properties of paragraph and character style.
- **paragraphFormat**: Specifies the properties of paragraph style.
- **basedOn**: Specifies that the current style inherits the style set to this property. This is how hierarchical styles are defined. It can be optional.

The style type should match the inherited style type. For example, it is not possible to have a character style inherit a paragraph style.

### Default style

The default style for span and paragraph properties is normal. It internally inherits the default style of the document loaded or document editor component.

### Style hierarchy

Each style initially checks its local value for the property that is being evaluated and turns to the style it is based on. If no local value is found, it turns to its default style.

Style inheritance of different styles are listed as follows:

#### Character style

Character styles are based only on other character styles.

The inheritance is: Character properties are inherited from the base character style.

#### Paragraph style

Paragraph styles are based on other paragraph styles or on linked styles.

When a paragraph style is based on another paragraph style, the inheritance of the properties is as follows:

- Paragraph properties are inherited from the base paragraph style.
- Span properties are inherited from the base paragraph style.

When a paragraph style is based on a linked style, the inheritance of the properties is as follows:

- Paragraph properties are inherited from the paragraph style part in its base linked style.
- Span properties are inherited from the span style part in its base linked style.

#### *Linked style*

Linked styles are composite styles and their components are paragraph and character styles with link between them. To apply paragraph properties, take the properties from the linked paragraph style. Similarly, to apply character properties, take the properties from linked character style.

Linked styles are based on other linked styles or on paragraph styles.

When a linked style is based on a paragraph style, the hierarchy of the properties is as follows:

- Paragraph properties are inherited from the 'basedOn' paragraph style.
- Character properties are inherited from the 'basedOn' paragraph style.

When a linked style is based on another linked style, the hierarchy of the properties is as follows:

- Paragraph properties are inherited from the paragraph style part in its base linked style.
- Span properties are inherited from the span style part in its base linked style.

#### *Defining new styles*

New Styles are defined and added to the style collection of the document. In this way, they will be discovered by the default UI and applied to the parts of a document.

#### *Defining a character style*

The following example shows how to programmatically create a character style.

```
`ts
let styleJson: any = {
  "type": "Character",
  "name": "New CharacterStyle",
  "basedOn": "Default Paragraph Font",
  "characterFormat": {
    "fontSize": 16.0,
    "fontFamily": "Calibri Light",
    "fontColor": "#2F5496",
    "bold": true,
    "italic": true,
    "underline": "Single"
  }
};
```

```
this.$refs.documenteditor.ej2Instances.editor.createStyle(JSON.stringify(styleJson));  
、
```

### Defining a paragraph style

The following example shows how to programmatically create a paragraph style.

```
`ts  
  
let styleJson: any = {  
  "type": "Paragraph",  
  "name": "New ParagraphStyle",  
  "basedOn": "Normal",  
  "characterFormat": {  
    "fontSize": 16.0,  
    "fontFamily": "Calibri Light",  
    "fontColor": "#2F5496",  
    "bold": true,  
    "italic": true,  
    "underline": "Single"  
  },  
  "paragraphFormat": {  
    "leftIndent": 0.0,  
    "rightIndent": 0.0,  
    "firstLineIndent": 0.0,  
    "beforeSpacing": 12.0,  
    "afterSpacing": 0.0,  
    "lineSpacing": 1.0791666507720947,  
    "lineSpacingType": "Multiple",  
    "textAlignment": "Left",  
    "outlineLevel": "Level1"  
  }  
};  
  
this.$refs.documenteditor.ej2Instances.editor.createStyle(JSON.stringify(styleJson));  
、
```

### Defining a linked style

The following example shows how to programmatically create linked style.

```
`ts
```

```

let styleJson: any = {
  "type": "Paragraph",
  "name": "New Linked",
  "basedOn": "Normal",
  "next": "Normal",
  "link": "New Linked Char",
  "characterFormat": {
    "fontSize": 16.0,
    "fontFamily": "Calibri Light",
    "fontColor": "#2F5496"
  },
  "paragraphFormat": {
    "leftIndent": 0.0,
    "rightIndent": 0.0,
    "firstLineIndent": 0.0,
    "beforeSpacing": 12.0,
    "afterSpacing": 0.0,
    "lineSpacing": 1.0791666507720947,
    "lineSpacingType": "Multiple",
    "textAlignment": "Left",
    "outlineLevel": "Level1"
  }
};

this.$refs.documenteditor.ej2Instances.editor.createStyle(JSON.stringify(styleJson));

```

### Applying a style

The styles are applied using the **applyStyle** method of **Editor**, the parameter should be passed is the **Name** of the Style.

The styles of the **Character** type is applied to the currently selected part of the document. If there is no selection, the values that will be applied to the word at caret position. The styles of **Paragraph** type follow the same logic and are applied to all paragraphs in the selection or the current paragraph.

When there is no selection, styles of **Linked** type will change the values of the paragraph, and apply both the Paragraph and Character properties. When there is selection, Linked Style changes only the character properties of the selected text.

For example, the following line will apply the "New Linked" to the current paragraph.

```
`ts
this.$refs.documenteditor.ej2Instances.editor.applyStyle('New Linked');
//Clear direct formatting and apply the specified style
this.$refs.documenteditor.ej2Instances.editor.applyStyle('New Linked', true);
`
```

### Get Styles

You can get the styles in the document using the below code snippet.

```
`typescript
//Get paragraph styles
let paragraphStyles = this.$refs.documenteditor.ej2Instances.documentEditor.getStyles('Paragraph');
//Get character styles
let paragraphStyles = this.$refs.documenteditor.ej2Instances.documentEditor.getStyles('Character');
`
```

### Modify an existing style

You can modify a existing style with the specified style properties using [createStyle](#) method. If modifyExistingStyle parameter is set to `true` the style properties is updated to the existing style.

The following illustrate to modify an existing style.

```
`typescript
let styleJson: any = {
  "type": "Paragraph",
  "name": "Heading 1",
  "characterFormat": {
    "fontSize": 32,
    "fontFamily": "Calibri"
  }
};
this.$refs.documenteditor.ej2Instances.documentEditor.editor.createStyle(styleName, true);
`
```

If modifyExistingStyle parameter is set to true and a style already exists with same name, it modifies the specified properties in the existing style.

If modifyExistingStyle parameter is set to false and a style already exists with same name, it creates a new style with unique name by appending '\_1'. Hence, the newly style will not have the specified name.

If no style exists with same name, it creates a new style.

Footer

## List format in Vue Document editor component

Document Editor supports both the single-level and multilevel lists. Lists are used to organize data as step-by-step instructions in documents for easy understanding of key points. You can apply list to the paragraph either using supported APIs.

### Create bullet list

Bullets are usually used for unordered lists. To apply bulleted list for selected paragraphs, use the following method of 'Editor' instance.

```
applyBullet(bullet, fontFamily);
```

Parameter	Type	Description
bullet	string	Bullet character.
fontFamily	string	Bullet font family.

Refer to the following sample code.

```
`ts
```

```
this.$refs.documenteditor.ej2Instances.editor.applyBullet('\uf0b7', 'Symbol');
```

```
,
```

### Create numbered list

Numbered lists are usually used for ordered lists. To apply numbered list for selected paragraphs, use the following method of 'Editor' instance.

```
applyNumbering(numberFormat,listLevelPattern)
```

Parameter	Type	Description
numberFormat	string	“%n” representations in ‘numberFormat’ parameter will be replaced by respective list level’s value. “%1” will be displayed as “1”
listLevelPattern(optional)	string	Default value is 'Arabic'.

Refer to the following example.

```
`ts
```

```
this.$refs.documenteditor.ej2Instances.editor.applyNumbering('%1', 'UpRoman');
```

```
,
```

### Clear list

You can also clear the list formatting applied for selected paragraphs. Refer to the following sample code.

```
`ts
```

```
this.$refs.documenteditor.ej2Instances.editor.clearList();
```

```
,
```



### Working with lists

The following sample demonstrates how to create bullet and numbering lists in Document Editor.

,

```
<template>
<div style="width:100%;height:330px">
<div>
<ejs-toolbar v-bind:clicked='toolbarButtonClick'>
<e-items>
<e-item prefixIcon="e-de-ctnr-bullets e-icons" tooltipText="Bullets" id="Bullets"></e-item>
<e-item prefixIcon="e-de-ctnr-numbering e-icons" tooltipText="Numbering" id="Numbering"></e-item>
<e-item text="Clear" id="clearlist" tooltipText="Clear List"></e-item>
</e-items>
</ejs-toolbar>
</div>
<ejs-documenteditor ref="documenteditor" :enableSelection='true' :isReadOnly='false'
:enableEditor='true' :enableEditorHistory='true' :enableSfdtExport='true' height="370px" style="width:
100%;"></ejs-documenteditor>
</div>
</template>
<script>
import Vue from 'vue'

import { DocumentEditorPlugin, Selection, Editor, SfdtExport, EditorHistory } from '@syncfusion/ej2-
vue-documenteditor';

import { ToolbarPlugin } from "@syncfusion/ej2-vue-navigations";

Vue.use(DocumentEditorPlugin);

Vue.use(ToolbarPlugin);

export default {
data: function() {
return {
};
},
provide: {
DocumentEditor : [SfdtExport, EditorHistory, Selection, Editor]
}
}
```

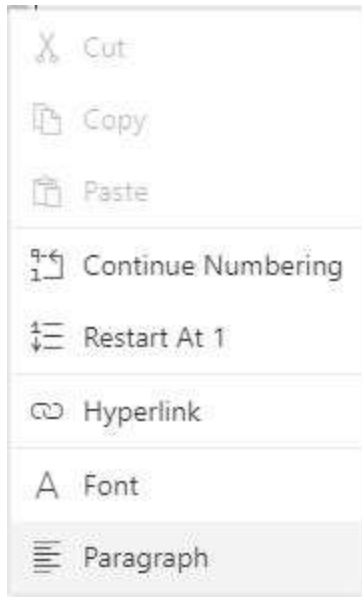
```

methods: {
  toolbarButtonClick: function(args) {
    switch (args.item.id) {
      case 'Bullets':
        //To create bullet list
        this.$refs.documenteditor.ej2Instances.editor.applyBullet('\uf0b7', 'Symbol');
        break;
      case 'Numbering':
        //To create numbering list
        this.$refs.documenteditor.ej2Instances.editor.applyNumbering('%1', 'UpRoman');
        break;
      case 'clearlist':
        //To clear list
        this.$refs.documenteditor.ej2Instances.editor.clearList();
        break;
    }
  }
}
</script>
<style>
@import "../../node_modules/@syncfusion/ej2-vue-documenteditor/styles/material.css";
</style>

```

### Editing numbered list

Document Editor restarts the numbering or continue numbering for a numbered list. These options are found in the built-in context menu, if the list value is selected. Refer to the following screenshot.



See Also

- [List dialog](#)

### Table format in Vue Document editor component

Document editor customizes the formatting of table, or table cells such as table width, cell margins, cell spacing, background color, and table alignment. This section describes how to customize these formatting for selected cells, rows, or table in detail.

#### Cell margins

You can customize the cell margins by using the following sample code.

```
`ts
//To change the left margin
this.$refs.documenteditor.ej2Instances.selection.cellFormat.leftMargin=5.4;
//To change the right margin
this.$refs.documenteditor.ej2Instances.selection.cellFormat.rightMargin=5.4;
//To change the top margin
this.$refs.documenteditor.ej2Instances.selection.cellFormat.topMargin=5.4;
//To change the bottom margin
this.$refs.documenteditor.ej2Instances.selection.cellFormat.bottomMargin=5.4;
`
```

You can also define the default cell margins for a table. If the specific cell margin value is not defined explicitly in the cell formatting, the corresponding value will be retrieved from default cells margin of the table. Refer to the following sample code.

```
`ts
```

```
//To change the left margin
this.$refs.documenteditor.ej2Instances.selection.tableFormat.leftMargin=5.4;
//To change the right margin
this.$refs.documenteditor.ej2Instances.selection.tableFormat.rightMargin=5.4;
//To change the top margin
this.$refs.documenteditor.ej2Instances.selection.tableFormat.topMargin=5.4;
//To change the bottom margin
this.$refs.documenteditor.ej2Instances.selection.tableFormat.bottomMargin=5.4;
`
```

### Background color

You can explicitly set the background color of selected cells using the following sample code.

```
`ts
this.$refs.documenteditor.ej2Instances.selection.cellFormat.background='#E0E0E0';
`
```

Refer to the following sample code to customize the background color of the table.

```
`ts
this.$refs.documenteditor.ej2Instances.selection.tableFormat.background='#E0E0E0';
`
```

### Cell spacing

Refer to the following sample code to customize the spacing between each cell in a table.

```
`ts
this.$refs.documenteditor.ej2Instances.selection.tableFormat.cellSpacing=2;
`
```

### Cell vertical alignment

The content is aligned within a table cell to 'Top', 'Center', or 'Bottom'. You can customize this property of selected cells. Refer to the following sample code.

```
`ts
this.$refs.documenteditor.ej2Instances.selection.cellFormat.verticalAlignment='Bottom';
`
```

### Table alignment

The tables are aligned in document editor to 'Left', 'Right', or 'Center'. Refer to the following sample code.

```
`ts
this.$refs.documenteditor.ej2Instances.selection.tableFormat.tableAlignment='Center';
`
```

### Cell width

Set the desired width of table cells that will be considered when the table is layouted. Refer to the following sample code.

```
`ts
this.$refs.documenteditor.ej2Instances.selection.cellFormat.preferredWidthType='Point';
this.$refs.documenteditor.ej2Instances.selection.cellFormat.preferredWidth=100;
`
```

### Table width

You can set the desired width of a table in **Point** or **Percent** type. Refer to the following sample code.

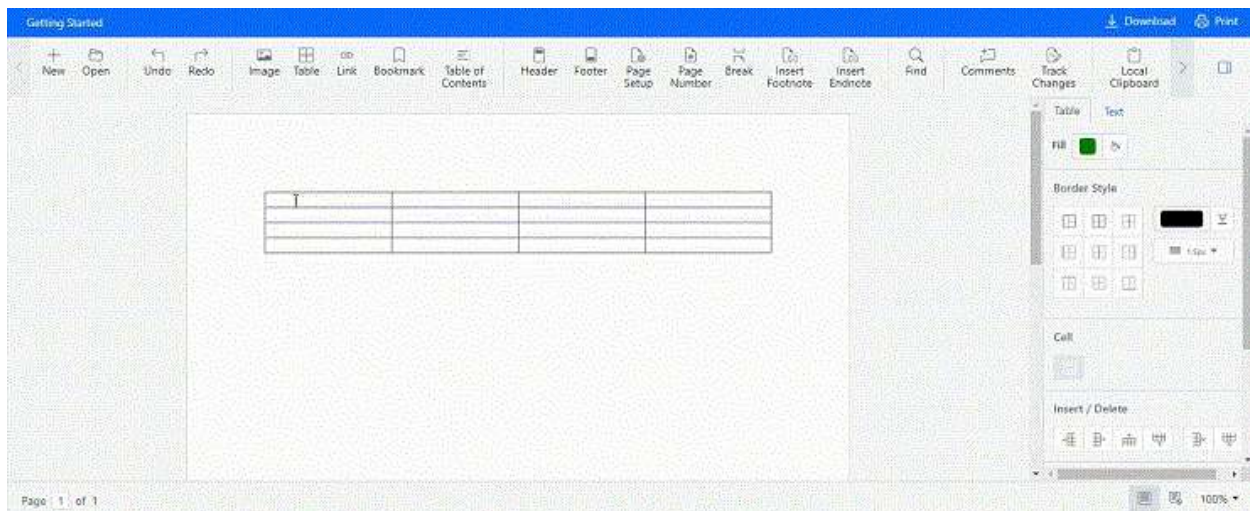
```
`ts
this.$refs.documenteditor.ej2Instances.selection.tableFormat.preferredWidthType='Point';
this.$refs.documenteditor.ej2Instances.selection.tableFormat.preferredWidth=300;
`
```

### Apply borders

Document editor exposes API to customize the borders for table cells by specifying the settings. Refer to the following sample code.

```
`ts
import { BorderSettings } from '@syncfusion/ej2-documenteditor';
//To apply border
let borderSettings: BorderSettings = {
type: 'AllBorders',
lineWidth: 12
};
this.$refs.documenteditor.ej2Instances.editor.applyBorders(borderSettings);
`
```

Please check below gif which illustrates how to apply border for selected cells through properties pane options - border color, line size and no border:



## Working with row formatting

Document editor allows various row formatting such as height and repeat header.

### Row height

You can customize the height of a table row as 'Auto', 'AtLeast', or 'Exactly'. Refer to the following sample code.

```
`ts
this.$refs.documenteditor.ej2Instances.selection.rowFormat.heightType='Exactly';
this.$refs.documenteditor.ej2Instances.selection.rowFormat.height=20;
`
```

### Header row

The header row describes the content of a table. A table can optionally have a header row. Only the first row of a table can be the header row. If the cursor position is at first row of the table, then you can define whether it as header row or not, using the following sample code.

```
`ts
this.$refs.documenteditor.ej2Instances.selection.rowFormat.isHeader=true;
`
```

### Allow row break across pages

This property is valid if a table row does not fit in the current page during table layout. It defines whether a table row can be allowed to break. If the value is false, the entire row will be moved to the start of next page. You can modify this property for selected rows using the following sample code.

```
`ts
this.$refs.documenteditor.ej2Instances.selection.rowFormat.allowRowBreakAcrossPages=false;
`
```

### Title

Document Editor expose API to get or set the table title of the selected table. Refer to the following sample code to set title.

```
`ts
```

```
this.$refs.documenteditor.ej2Instances.selection.tableFormat.title = 'Shipping Details';
```

```
,
```

#### Description

Document Editor expose API to get or set the table description of the selected image. Refer to the following sample code to set description.

```
`ts
```

```
this.$refs.documenteditor.ej2Instances.selection.tableFormat.description = 'Freight cost and shipping details';
```

```
,
```

See Also

- [Table properties dialog](#)

### Section format in Vue Document editor component

Document Editor supports various section formatting such as page size, page margins, and more.

#### Page size

You can get or set the size of a section at cursor position by using the following sample code.

```
`javascript
```

```
this.$refs.documenteditor.ej2Instances.selection.sectionFormat.pageWidth = 500;
```

```
this.$refs.documenteditor.ej2Instances.selection.sectionFormat.pageHeight = 600;
```

```
,
```

You can change the orientation of the page by swapping the values of page width and height respectively.

#### Page margins

Left and right page margin defines the gap between the document content from left and right side of the page respectively. Top and bottom page margins defines the gap between the document content from header and footer of the page respectively.

Refer to the following sample code.

```
`javascript
```

```
this.$refs.documenteditor.ej2Instances.selection.sectionFormat.leftMargin = 10;
```

```
this.$refs.documenteditor.ej2Instances.selection.sectionFormat.rightMargin = 10;
```

```
this.$refs.documenteditor.ej2Instances.selection.sectionFormat.bottomMargin = 10;
```

```
this.$refs.documenteditor.ej2Instances.selection.sectionFormat.topMargin = 10;
```

```
,
```

### Header distance

You can define the distance of header content from the top of the page by using the following sample code.

```
`javascript
this.$refs.documenteditor.ej2Instances.selection.sectionFormat.headerDistance = 72;
`
```

### Footer distance

You can define the distance of footer content from the bottom of the page by using the following sample code.

```
`javascript
this.$refs.documenteditor.ej2Instances.selection.sectionFormat.footerDistance = 72;
`
```

### Columns

You can define the number of columns, column width, and space between columns for the pages in a section.

The following code example illustrates how to define the two columns layout for the pages in a section.

```
`javascript
let column = new SelectionColumnFormat(this.$refs.documenteditor.ej2Instances.selection);
column.width = 216;
column.space = 36;
this.$refs.documenteditor.ej2Instances.selection.sectionFormat.columns = [column, column];
this.$refs.documenteditor.ej2Instances.selection.sectionFormat.lineBetweenColumns = true;
`
```

### Breaks

You can insert Column break. The following code indicate that the text following the column break will begin in the next column.

```
`javascript
this.$refs.documentEditor.ej2Instances.editor.insertColumnBreak();
`
```

You can insert next page section break to start the new section on the next page.

The following code example illustrates how to insert a next page section break.

```
`javascript
this.$refs.documentEditor.ej2Instances.editor.insertSectionBreak(SectionBreakType.NewPage);
`
```

You can insert continuous section break to start the new section on the same page



The following code example illustrates how to insert a continuous section break

```
`javascript
this.$refs.documentEditor.ej2Instances.editor.insertSectionBreak(SectionBreakType.Continuous);
`
```

See Also

[\\*Page setup dialog](#)

### Comments in Vue Document editor component

Document Editor allows you to add comments to documents. You can add, navigate and remove comments in code and from the UI.

#### Add a new comment

Comments can be inserted to the selected text.

```
`ts
this.$refs.documentEditor.ej2Instances.editor.insertComment("Test comment");
`
```

#### Comment navigation

Next and previous comments can be navigated using the below code snippet.

```
`ts
//Navigate to next comment
this.$refs.documentEditor.ej2Instances.selection.navigateNextComment();
//Navigate to previous comment
this.$refs.documentEditor.ej2Instances.selection.navigatePreviousComment();
`
```

#### Delete comment

Current comment can be deleted using the below code snippet.

```
`ts
this.$refs.documentEditor.ej2Instances.editor.deleteComment();
`
```

#### Delete all comment

All the comments in the document can be deleted using the below code snippet.

```
`ts
this.$refs.documentEditor.ej2Instances.editor.deleteAllComments();
`
```

#### Protect the document in comments only mode

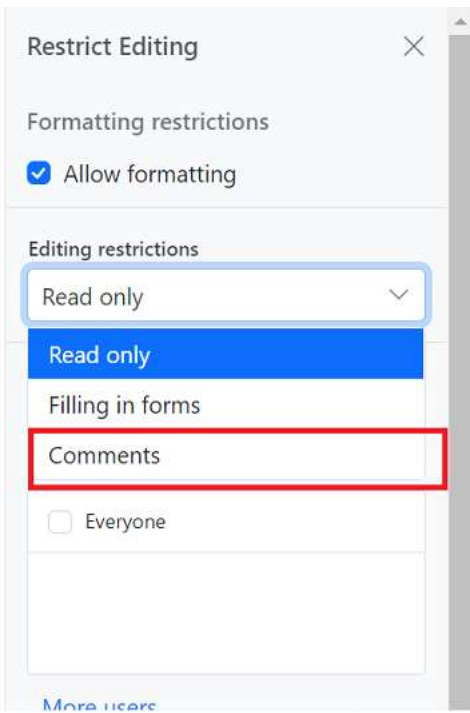
Document Editor provides support for protecting the document with **CommentsOnly** protection. In this protection, user allowed to add or edit comments alone in the document.

Document editor provides an option to protect and unprotect document using [enforceProtection](#) and [stopProtection](#) API.

The following example code illustrates how to enforce and stop protection in Document editor container.

```
`ts
<template>
<div id="app">
<ejs-documenteditorcontainer ref='container' :serviceUrl='serviceUrl' height="590px" id='container'
:enableToolbar='true'></ejs-documenteditorcontainer>
</div>
</template>
<script>
import Vue from 'vue';
import { DocumentEditorContainerPlugin, DocumentEditorContainerComponent,Toolbar} from
'@syncfusion/ej2-vue-documenteditor';
Vue.use(DocumentEditorContainerPlugin);
export default {
data() {
return { serviceUrl:'https://ej2services.syncfusion.com/production/web-services/api/documenteditor/'};
},
provide: {
//Inject require modules.
DocumentEditorContainer: [Toolbar]
},
mounted(){
//enforce protection
this.$refs.container.ej2Instances.documentEditor.editor.enforceProtection('123','CommentsOnly');
//stop the document protection
this.$refs.container.ej2Instances.documentEditor.editor.stopProtection('123');
}
}
</script>
`
```

Comment only protection can be enabled in UI by using [Restrict Editing pane](#)



Note: In enforce Protection method, first parameter denotes password and second parameter denotes protection type. Possible values of protection type are `NoProtection` | `ReadOnly` | `FormFieldsOnly` | `CommentsOnly`. In stop protection method, parameter denotes the password.

### Track changes in Vue Document editor component

Track Changes allows you to keep a record of changes or edits made to a document. You can then choose to accept or reject the modifications. It is a useful tool for managing changes made by several reviewers to the same document. If track changes option is enabled, all editing operations are preserved as revisions in Document Editor.

#### Enable track changes in Document Editor

The following example demonstrates how to enable track changes.

```
`ts
```

```
<ejs-documenteditorcontainer ref='container' :serviceUrl='serviceUrl' height="590px" id='container'
:enableTrackChanges='true'></ejs-documenteditorcontainer>
```

```
,
```

#### Get all tracked revisions

The following example demonstrate how to get all tracked revision from current document.

```
`ts
```

```
<ejs-documenteditorcontainer ref='container' :serviceUrl='serviceUrl' height="590px" id='container'
:enableTrackChanges='true'></ejs-documenteditorcontainer>
```

```
/
```

- Get revisions from the current document

```
*/
```

```
let revisions : RevisionCollection = this.$refs.container.documentEditor.revisions;
```

```
,
```

### Accept or Reject all changes programmatically

The following example demonstrates how to accept/reject all changes.

```
`ts
```

```
<ejs-documenteditorcontainer ref='container' :serviceUrl='serviceUrl' height="590px" id='container'
:enableTrackChanges='true'></ejs-documenteditorcontainer>
```

```
/
```

- Get revisions from the current document

```
*/
```

```
let revisions : RevisionCollection = this.$refs.container.documentEditor.revisions;
```

```
/
```

- Accept all tracked changes

```
*/
```

```
revisions.acceptAll();
```

```
/
```

- Reject all tracked changes

```
*/
```

```
revisions.rejectAll();
```

```
,
```

### Accept or reject a specific revision

The following example demonstrates how to accept/reject specific revision in the Document Editor.

```
`ts
```

```
/
```

- Get revisions from the current document

```
*/
```

```
let revisions : RevisionCollection = this.$refs.container.documentEditor.revisions;
```

```
/
```

- Accept specific changes

```
*/
```

```
revisions.get(0).accept();
```

```
/
```

- Reject specific changes

```
*/
```

```
revisions.get(1).reject();
```

```
,
```

### Navigate between the tracked changes

The following example demonstrates how to navigate tracked revision programmatically.

```
`ts
```

```
<ejs-documenteditorcontainer ref='container' :serviceUrl='serviceUrl' height="590px" id='container'
:enableTrackChanges='true'></ejs-documenteditorcontainer>
```

```
/
```

- Navigate to next tracked change from the current selection.

```
*/
```

```
this.$refs.container.documentEditor.selection.navigateNextRevision();
```

```
/
```

- Navigate to previous tracked change from the current selection.

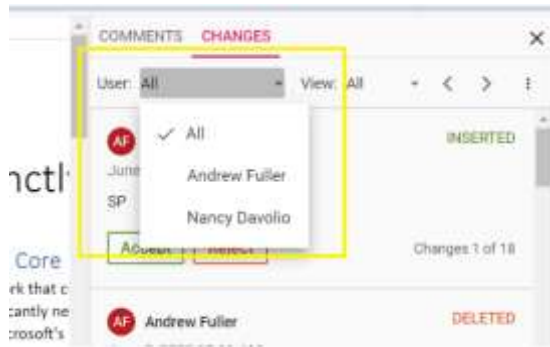
```
*/
```

```
this.$refs.container.documentEditor.selection.navigatePreviousRevision();
```

```
,
```

### Filtering changes based on user

In DocumentEditor, we have built-in review panel in which we have provided support for filtering changes based on the user.



### Protect the document in track changes only mode

Document Editor provides support for protecting the document with **RevisionsOnly** protection. In this protection, all the users are allowed to view the document and do their corrections, but they cannot accept or reject any tracked changes in the document. Later, the author can view their corrections and accept or reject the changes.

Document editor provides an option to protect and unprotect document using [enforceProtection](#) and [stopProtection](#) API.

The following example code illustrates how to enforce and stop protection in Document editor container.

```
`ts
<template>
<div id="app">

<ejs-documenteditorcontainer ref='container' :serviceUrl='serviceUrl' height="590px" id='container'
:enableToolbar='true'></ejs-documenteditorcontainer>

</div>
</template>
<script>
import Vue from 'vue';

import { DocumentEditorContainerPlugin, DocumentEditorContainerComponent, Toolbar } from
'@syncfusion/ej2-vue-documenteditor';

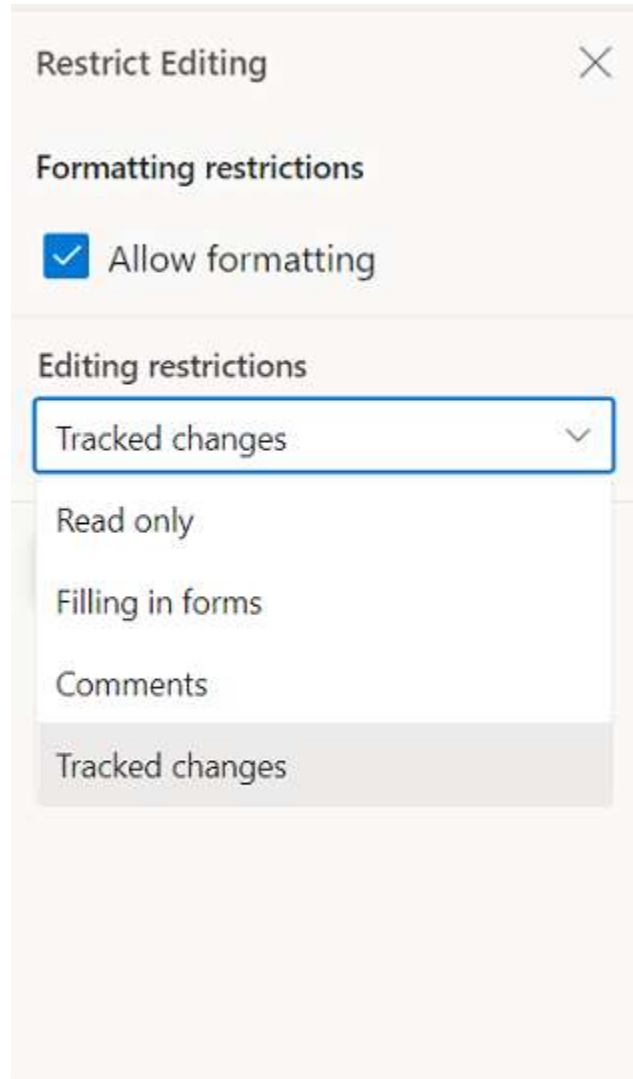
Vue.use(DocumentEditorContainerPlugin);

export default {
  data() {
    return { serviceUrl: 'https://ej2services.syncfusion.com/production/web-services/api/documenteditor/' };
  },
  provide: {
    //Inject require modules.
    DocumentEditorContainer: [Toolbar]
  },
  mounted(){
    //enforce protection
    this.$refs.container.ej2Instances.documentEditor.editor.enforceProtection('123','RevisionsOnly');
    //stop the document protection
    this.$refs.container.ej2Instances.documentEditor.editor.stopProtection('123');
  }
}
```

```
</script>
```

```
,
```

Tracked changes only protection can be enabled in UI by using [Restrict Editing pane](#)



Note: In enforce Protection method, first parameter denotes password and second parameter denotes protection type. Possible values of protection type are NoProtection | ReadOnly | FormFieldsOnly | CommentsOnly | RevisionsOnly. In stop protection method, parameter denotes the password.

#### Event

You can restrict the accept and reject changes based on the author name. The following example demonstrates how to restrict an author from accept/reject changes.

```
`ts
```

```
<template>
```

```
<div>
```

```
<div>
```

```

<div>
<DocumentEditorContainerComponent
ref="container"
style="display: block;"
:height="'590px'"
@beforeAcceptRejectChanges="beforeAcceptRejectChanges"
:enableToolbar="true"
/>
</div>
</div>
</div>
</template>
<script>
import Vue from 'vue';
import { DocumentEditorContainerPlugin, DocumentEditorContainerComponent, Toolbar } from
 '@syncfusion/ej2-vue-documenteditor';
Vue.use(DocumentEditorContainerPlugin);
export default {
  components: {
    DocumentEditorContainerComponent
  },
  methods: {
    beforeAcceptRejectChanges(args) {
      // Check the author of the revision
      if (args.author !== 'Hary') {
        // Cancel the accept/reject action
        args.cancel = true;
      }
    }
  }
};
</script>

```



## Fields in Vue Document editor component

Document Editor has preservation support for all types of fields in an existing word document without any data loss.

### Adding Fields

You can add a field to the document by using [insertField](#) method in [Editor](#) module.

The following example code illustrates how to insert merge field programmatically by providing the field code and field result.

```
`ts
let fieldCode: string = 'MERGEFIELD First Name \* MERGEFORMAT ';
let fieldResult: string = '«First Name»';
this.$refs.documenteditor.ej2Instances.editor.insertField(fieldCode, fieldResult);
`
```

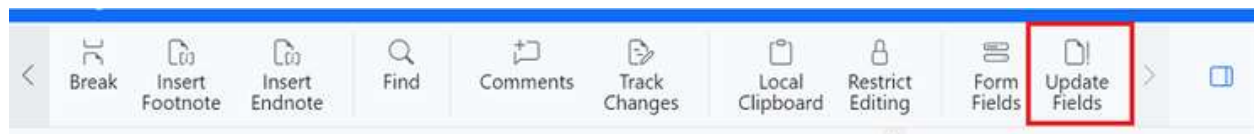
Note: Document editor does not validate/process the field code/field result. it simply inserts the field with specified field information.

### Update fields

Document Editor provides support for updating bookmark cross reference field. The following example code illustrates how to update bookmark cross reference field.

```
`ts
//Update all the bookmark cross reference field in the document.
this.$refs.documenteditor.ej2Instances.updateFields();
`
```

Bookmark cross reference fields can be updated through UI by using update fields option in **Toolbar**.



The following type of fields are automatically updated in Document Editor.

- Numpages
- Section
- Page

### Get field info

You can get field code and field result of the current selected field by using [getFieldInfo](#) method in the [Selection](#) module.

```
`ts
//Gets the field information of the selected field.
let fieldInfo: FieldInfo = this.$refs.documenteditor.ej2Instances.selection.getFieldInfo();
`
```

Note: For nested fields, this method returns combined field code and result.

#### Set field info

You can modify the field code and field result of the current selected field by using [setFieldInfo](#) method in the [Editor](#) module.

```
`ts
//Gets the field information for the selected field.
let fieldInfo: FieldInfo = this.$refs.documenteditor.ej2Instances.selection.getFieldInfo();
//Modify field code
fieldInfo.code = 'MERGEFIELD First Name \* MERGEFORMAT ';
//Modify field result
fieldInfo.result = '«First Name»';
//Modify field code and result of the current selected field.
this.$refs.documenteditor.ej2Instances.editor.setFieldInfo(fieldInfo);`
```

Note: For nested field, entire field gets replaced completely with the specified field information.

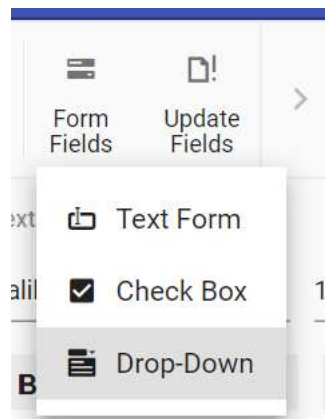
See Also

[Mail merge using DocIO](#)

[Mail merge demo](#)

#### Form fields in Vue Document editor component

Document Editor Container component provide support for inserting Text, CheckBox, DropDown form fields through in-built toolbar.



#### Insert form field

Form fields can be inserted using [insertFormField](#) method in editor module.

```
`ts
//Insert Text form field`
```

```
this.$refs.documenteditor.ej2Instances.editor.insertFormField('Text');  
//Insert Checkbox form field  
this.$refs.documenteditor.ej2Instances.editor.insertFormField('CheckBox');  
//Insert Drop down form field  
this.$refs.documenteditor.ej2Instances.editor.insertFormField('Dropdown');  
`
```

#### Get form field names

All the form fields names form current document can be retrieved using [getFormFieldNames\(\)](#).

```
`ts  
this.$refs.documenteditor.ej2Instances.getFormFieldNames();  
`
```

#### Get form field properties

Form field properties can be retrieved using [getFormFieldInfo\(\)](#).

```
`ts  
//Text form field  
var textfieldInfo = this.$refs.documenteditor.ej2Instances.getFormFieldInfo('Text1');  
//Checkbox form field  
var checkboxfieldInfo = this.$refs.documenteditor.ej2Instances.getFormFieldInfo('Check1');  
//Dropdown form field  
var dropdownfieldInfo = this.$refs.documenteditor.ej2Instances.getFormFieldInfo('Drop1');  
`
```

#### Set form field properties

Form field properties can be modified using [setFormFieldInfo\(\)](#).

```
`ts  
// Set text form field properties  
var textfieldInfo = this.$refs.documenteditor.ej2Instances.getFormFieldInfo('Text1');  
textfieldInfo.defaultValue = "Hello";  
textfieldInfo.format = "Uppercase";  
textfieldInfo.type = "Text";  
textfieldInfo.name = "Text2";  
this.$refs.documenteditor.ej2Instances.setFormFieldInfo('Text1',textfieldInfo);  
// Set checkbox form field properties  
var checkboxfieldInfo = this.$refs.documenteditor.ej2Instances.getFormFieldInfo('Check1');  
checkboxfieldInfo.defaultValue = true;  
`
```

```
checkboxfieldInfo.name = "Check2";
this.$refs.documenteditor.ej2Instances.setFormFieldInfo('Check1',checkboxfieldInfo);
// Set checkbox form field properties
var dropdownfieldInfo = this.$refs.documenteditor.ej2Instances.getFormFieldInfo('Drop1');
dropdownfieldInfo.dropDownItems = ['One','Two', 'Three'];
dropdownfieldInfo.name = "Drop2";
this.$refs.documenteditor.ej2Instances.setFormFieldInfo('Drop1',dropdownfieldInfo);
`ts
```

Note:If a form field already exists in the document with the new name specified, the old form field name property will be cleared and it will not be accessible. Ensure the new name is unique.

#### Export form field data

Data of the all the Form fields in the document can be exported using [exportFormData](#).

```
`ts
var formFieldDate = this.$refs.documenteditor.ej2Instances.exportFormData();
`
```

#### Import form field data

Form fields can be prefilled with data using [importFormData](#).

```
`ts
var textformField = {fieldName: 'Text1', value: 'Hello World'};
var checkformField = {fieldName: 'Check1', value: true};
var dropdownformField = {fieldName: 'Drop1', value: 1};
//Import form field data
this.$refs.documenteditor.ej2Instances.importFormData([textformField,checkformField,dropdownformField]);
`
```

#### Reset form fields

Reset all the form fields in current document to default value using [resetFormFields](#).

```
`ts
this.$refs.documenteditor.ej2Instances.resetFormFields();
`
```

#### Protect the document in form filling mode

Document Editor provides support for protecting the document with **FormFieldsOnly** protection. In this protection, user can only fill form fields in the document.

Document editor provides an option to protect and unprotect document using [enforceProtection](#) and [stopProtection](#) API.

The following example code illustrates how to enforce and stop protection in Document editor container.

```
`ts
<template>
<div id="app">
<ejs-documenteditorcontainer ref='container' :serviceUrl='serviceUrl' height="590px" id='container'
:enableToolbar='true'></ejs-documenteditorcontainer>
</div>
</template>
<script>
import Vue from 'vue';
import { DocumentEditorContainerPlugin, DocumentEditorContainerComponent, Toolbar } from
'@syncfusion/ej2-vue-documenteditor';
Vue.use(DocumentEditorContainerPlugin);
export default {
data() {
return { serviceUrl: 'https://ej2services.syncfusion.com/production/web-services/api/documenteditor/' };
},
provide: {
//Inject require modules.
DocumentEditorContainer: [Toolbar]
},
mounted(){
//enforce protection
this.$refs.container.ej2Instances.documentEditor.editor.enforceProtection('123', 'FormFieldsOnly');
//stop the document protection
this.$refs.container.ej2Instances.documentEditor.editor.stopProtection('123');
}
}
</script>
`
```

Note: In enforce Protection method, first parameter denotes password and second parameter denotes protection type. Possible values of protection type are **NoProtection | ReadOnly | FormFieldsOnly**. In stop protection method, parameter denotes the password.

## Clipboard in Vue Document editor component

Document Editor takes advantage of system clipboard and allows you to copy or move a portion of the document into it in HTML format, so that it can be pasted in any application that supports clipboard.

### Copy

Copy a portion of document to system clipboard using built-in context menu of Document Editor. You can also do it programmatically using the following sample code.

```
`javascript
this.$refs.documenteditor.ej2Instances.selection.copy();
`
```

### Cut

Cut a portion of document to system clipboard using built-in context menu of Document Editor. You can also do it programmatically using the following sample code.

```
`javascript
this.$refs.documenteditor.ej2Instances.editor.cut();
`
```

### Paste

Due to limitations, you can paste contents from system clipboard as plain text in Document Editor only using the 'CTRL + V' keyboard shortcut.

### Local paste

Document Editor expose API to enable local paste within the control. On enabling this, the following is performed:

- Selected contents will be stored to an internal clipboard in addition to system clipboard.
- Clipboard paste will be overridden, and internally stored data that has formatted text will be pasted.

Refer to the following sample code.

```
`
<ejs-documenteditor ref="documenteditor" :enableLocalPaste='true' :enableEditor='true'
:isReadOnly=false style="width: 100%;height: 100%;"></ejs-documenteditor>
`
```

By default, **enableLocalPaste** is false.

When local paste is enabled for a Document Editor instance, you can paste contents programmatically if the internal clipboard has stored data during last copy operation. Refer to the following sample code.

```
`javascript
this.$refs.documenteditor.ej2Instances.editor.pasteLocal();
`
```

### EnableLocalPaste behaviour

**| EnableLocalPaste | Paste behavior details |**

|-----|-----|

|True| Allows to paste content that is copied from the same Document Editor component alone and prevents pasting content from system clipboard. Hence the content copied from outside Document Editor component can't be pasted.<br>Browser limitation of pasting from system clipboard using API and context menu options, will be resolved. So, you can copy and paste content within the Document Editor component using API and context menu options too. |

|False| Allows to paste content from system clipboard. Hence the content copied from both the Document Editor component and outside can be pasted.<br>Browser limitation of pasting from system clipboard using API and context menu options, will remain as a limitation. |

Note:

- Keyboard shortcut for pasting will work properly in both cases.
- Copying content from Document Editor component and pasting outside will work properly in both cases.

### Paste with formatting

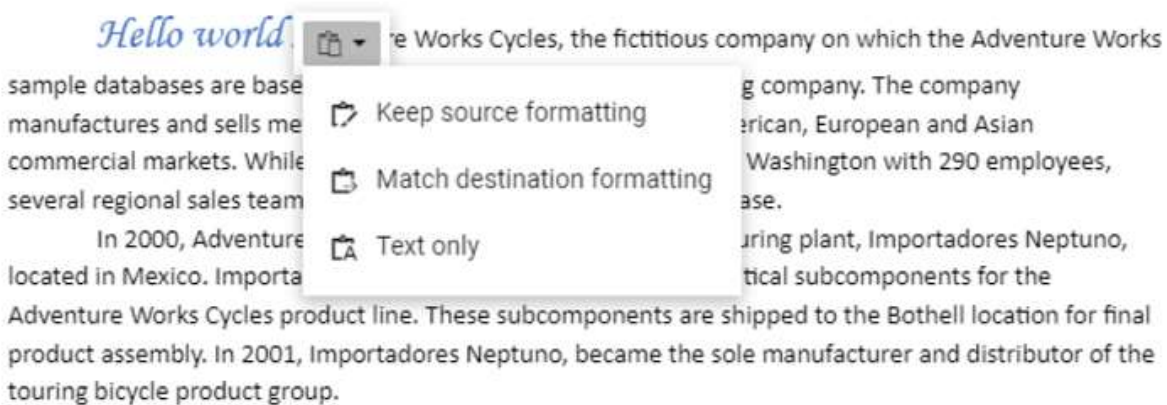
Document Editor provides support to paste the system clipboard data with formatting. To enable clipboard paste with formatting options, set the `enableLocalPaste` property in Document Editor to false and use this .NET Standard library [Syncfusion.EJ2.WordEditor.AspNet.Core](#) by the web API service implementation. This library helps you to paste the system clipboard data with formatting.

You can paste your system clipboard data in the following ways:

- **Keep Source Formatting** This option retains the character styles and direct formatting applied to the copied text. Direct formatting includes characteristics such as font size, italics, or other formatting that is not included in the paragraph style.
- **Match Destination Formatting** This option discards most of the formatting applied directly to the copied text, but it retains the formatting applied for emphasis, such as bold and italic when it is applied to only a portion of the selection. The text takes on the style characteristics of the paragraph where it is pasted. The text also takes on any direct formatting or character style properties of text that immediately precedes the cursor when the text is pasted.
- **Text Only** This option discards all formatting and non-text elements such as pictures or tables. The text takes on the style characteristics of the paragraph where it is pasted and takes on any direct formatting or character style properties of text that immediately precedes the cursor when the text is pasted. Graphical elements are discarded and tables are converted to a series of paragraphs.

This paste option appears as follows.

## Adventure Works Cycles



See Also

- [Feature modules](#)
- [Keyboard shortcuts](#)

## History in Vue Document editor component

Document Editor tracks the history of all editing actions done in the document, which allows undo and redo functionality.

### Enable or disable history

Inject the `EditorHistory` module in your application to provide history preservation functionality for `DocumentEditor`. Refer to the following code example.

```
,
<template>
<div id="app">
<ejs-documenteditor ref="documenteditor" :enableEditor='true' :isReadOnly='false'
:enableEditorHistory='true' style="width: 100%;height: 100%;"></ejs-documenteditor>
</div>
</template>
<script>
import Vue from 'vue'

import { DocumentEditorPlugin, Editor, Selection, EditorHistory } from '@syncfusion/ej2-vue-
documenteditor';

Vue.use(DocumentEditorPlugin);

export default {
data: function() {
```



```

return {
};
},
provide: {
  DocumentEditor : [Editor, Selection, EditorHistory]
}
}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-vue-documenteditor/styles/material.css";
</style>
`

```

You can enable or disable history preservation for a Document Editor instance any time using the `enableEditorHistory` property. Refer to the following sample code.

```

`javascript
this.$refs.documenteditor.enableEditorHistory = false;
`

```

### Undo and redo

You can perform undo and redo by `CTRL+Z` and `CTRL+Y` keyboard shortcuts. Document Editor exposes API to do it programmatically.

To undo the last editing operation in Document Editor, refer to the following sample code.

```

`javascript
this.$refs.documenteditor.ej2Instances.editorHistory.undo();
`

```

To redo the last undone action, refer to the following code example.

```

`javascript
this.$refs.documenteditor.ej2Instances.editorHistory.redo();
`

```

### Stack size

History of editing actions will be maintained in stack, so that the last item will be reverted first. By default, Document Editor limits the size of undo and redo stacks to 500 each respectively. However, you can customize this limit. Refer to the following sample code.

```

`javascript
this.$refs.documenteditor.ej2Instances.editorHistory.undoLimit = 400;
this.$refs.documenteditor.ej2Instances.editorHistory.redoLimit = 400;
`

```

See Also

- [Feature modules](#)
- [Keyboard shortcuts](#)

### Find and replace in Vue Document editor component

The Document Editor component searches a portion of text in the document through a built-in interface called **OptionsPane** or rich APIs. When used in combination with selection performs various operations on the search results like replacing it with some other text, highlighting it, making it bolder, and more.

#### Options pane

This provides the options to search for a portion of text in the document. After search operation is completed, the search results will be displayed in a list and options to navigate between them. The current occurrence of matched text or all occurrences with another text can be replaced by switching to **Replace** tab. This pane is opened using the keyboard shortcut **CTRL+F**. You can also open it programmatically using the following sample code.

```
<template>
<div id="app" height="350px">
<div>
<button v-on:click='showOptionsPane' >Save</button>
</div>
<ejs-documenteditor ref="documenteditor" :enableEditor='true' :enableSearch='true'
:enableOptionsPane='true' :isReadOnly='false' height="370px" style="width: 100%;"></ejs-
documenteditor>
</div>
</template>
<script>
import Vue from 'vue'

import { DocumentEditorPlugin, Selection, Editor, Search, OptionsPane } from '@syncfusion/ej2-vue-
documenteditor';

Vue.use(DocumentEditorPlugin);

export default {
data: function() {
return {
};
},
provide: {
```

```
//Inject require modules.
DocumentEditor : [ Selection, Editor, Search, OptionsPane]
}
methods: {
  showOptionsPane: function() {
    //Open options pane.
    this.$refs.documenteditor.showOptionsPane();
  },
  mounted() {
    let sfdt: string = `{
      "sections": [
        {
          "blocks": [
            {
              "inlines": [
                {
                  "characterFormat": {
                    "bold": true,
                    "italic": true
                  },
                  "text": "Adventure Works Cycles, the fictitious company on which the AdventureWorks sample
databases are based, is a large, multinational manufacturing company. The company manufactures and
sells metal and composite bicycles to North American, European and Asian commercial markets. While
its base operation is located in Bothell, Washington with 290 employees, several regional sales teams
are located throughout their market base."
                }
              ]
            }
          ]
        }
      ]
    }`;
    this.$refs.documenteditor.open(sfdt);
```

```

}
}
</script>
<style>
@import "../../node_modules/@syncfusion/ej2-vue-documenteditor/styles/material.css";
</style>
`

```

You can close the options pane by pressing **Esc** key.

### Search

The [Search](#) module of Document Editor exposes the following APIs:

API Name	Type	Description
<a href="#">findAll()</a>	Method	Searches for specified text in the whole document and highlights it with yellow.
<a href="#">searchResults</a>	Property	This is an instance of <a href="#">SearchResults</a> .
<a href="#">find()</a>	Method	Find immediate occurrence of specified text from cursor position in the document and highlights it with yellow.

#### *Find the immediate occurrence in the document*

Using [find\(\)](#) method, you can find the immediate occurrence of specified text from current cursor position in the document.

The following example code illustrates how to use find in Document editor.

```

`ts
this.$refs.documenteditor.ej2instances.search.find('Some text', 'None');
`

```

Note: Second parameter is optional parameter and it denotes find Options. Possible values of find options are **'None' | 'WholeWord' | 'CaseSensitive' | 'CaseSensitiveWholeWord'**.

#### *Find all the occurrences in the document*

Using [findAll\(\)](#) method, you can find all the occurrences of specified text in the whole document and highlight it with yellow.

The following example code illustrates how to find All the text in the document.

```

`ts
this.$refs.documenteditor.ej2instances.search.findAll('Some text', 'None');
`

```

Note: Second parameter is optional parameter and it denotes find Options. Possible values of find options are **'None' | 'WholeWord' | 'CaseSensitive' | 'CaseSensitiveWholeWord'**.

## Search results

The [SearchResults](#) class provides information about the search results after a search operation is completed that can be identified using the [searchResultsChange](#) event. This will expose the following APIs:

API Name	Type	Description
<a href="#">length</a>	Property	Returns the total number of results found on the search.
<a href="#">index</a>	Property	Returns the index of selected search result. You can change the value for this property to move the selection.
<a href="#">replaceAll()</a>	Method	Replaces all the occurrences with specified text.
<a href="#">clear()</a>	Method	Clears the search result.

## Replace all the occurrences

Using [replaceAll](#), you can replace all the occurrences with specified text.

The following example code illustrates how to use replace All in Document editor.

```
`ts
```

```
this.$refs.documentEditor.ej2Instances.search.findAll ('Some text');
```

```
// Replace all the searched text with word 'Mike'
```

```
this.$refs.documentEditor.ej2Instances.search.searchResults.replaceAll("Mike");
```

```
`
```

## Replace

Using [insertText](#), you can replace the current searched text with specified text and it replace single occurrence.

Note: This [insertText](#) API accepts following control characters

- \* New line characters ("[\r](#)", "[\r\n](#)", "[\n](#)") - Inserts a new paragraph and appends the remaining text to the new paragraph.

- \* Line break character ("[\v](#)") - Moves the remaining text to start in new line.

- \* Tab character ("[\t](#)") - Allocates a tab space and continue the next character.

The following example code illustrates how to find a text in the document and replace each occurrence of the text one by one programmatically.

```
`ts
```

```
this.$refs.container.ej2instances.documentEditor.search.findAll('works');
```

```
let searchLength: number = container.documentEditor.search.searchResults.length;
```

```
for (let i = 0; i < searchLength; i++) {
```

```
// It will move selection to specific searched index,move to each occurrence one by one
```

```
this.$refs.container.ej2instances.documentEditor.search.searchResults.index = i;
```

```
// Replace it with some text
```

```
this.$refs.container.ej2instances.documentEditor.editor.insertText('Hello');
}
this.$refs.container.ej2Instances.documentEditor.search.searchResults.clear();
`
```

### SearchResultsChange event

[DocumentEditor](#) exposes the [searchResultsChange](#) event that will be triggered whenever search results are changed. Consider the following scenarios:

- A search operation is completed with some results.
- The results are replaced with some other text, since it will be cleared automatically.
- The results are cleared explicitly.

Refer to the following code example.

```
`ts
documenteditor.searchResultsChange = function() {
};
`
```

### Customize find and replace

Using the exposed APIs, you can customize the find and replace functionality in your application. Refer to the following sample code.

#### **APP.VUE**

```
<template>
  <div id="app" height="350px">
    <div>
      <table>
        <tr>
          <td>
            <label>Text to find:</label>
          </td>
          <td>
            <input type="text" id="find_text" />
          </td>
        </tr>
        <tr>
          <td>
            <label>Text to replace:</label>
          </td>
          <td>
            <input type="text" id="replace_text" />
          </td>
        </tr>
        <tr>
          <td colspan="2">
            <button id="replace_all" v-
on:click='onReplaceButtonClick' style="float:right;margin-top: 10px;"
class="e-control e-btn">Replace All</button>
          </td>
        </tr>
      </table>
    </div>
  </div>
</template>
```

```

        </tr>
      </table>
    </div>
    <ejs-documenteditor ref="documenteditor" :enableEditor='true'
:enableSearch='true' :enableOptionsPane='true' :isReadOnly='false'
height="370px" style="width: 100%;"></ejs-documenteditor>
  </div>
</template>
<script>
  import Vue from 'vue'
  import { DocumentEditorPlugin, Selection, Editor, Search,
OptionsPane } from '@syncfusion/ej2-vue-documenteditor';
  Vue.use(DocumentEditorPlugin);
  export default {
    data: function() {
      return {
      };
    },
    provide: {
      //Inject require modules.
      DocumentEditor : [ Selection, Editor, Search, OptionsPane]
    },
    methods: {
      onReplaceButtonClick: function() {
        let textToFind: string =
(document.getElementById('find_text') as HTMLInputElement).value;
        let textToReplace: string =
(document.getElementById('replace_text') as HTMLInputElement).value;
        if (textToFind !== '') {
          // Find all the occurrences of given text

this.$refs.documenteditor.ej2Instances.search.findAll(textToFind);
          if
(this.$refs.documenteditor.ej2Instances.searchModule.searchResults.length >
0) {
            // Replace all the occurrences of given text

this.$refs.documenteditor.ej2Instances.search.searchResults.replaceAll(textT
oReplace);
          }
        }
      },
      mounted() {
        let sfdt: string = `{
          "sections": [
            {
              "blocks": [
                {
                  "inlines": [
                    {
                      "characterFormat": {
                        "bold": true,
                        "italic": true
                      },
                      "text": "Adventure Works Cycles,
the fictitious company on which the AdventureWorks sample databases are

```

based, is a large, multinational manufacturing company. The company manufactures and sells metal and composite bicycles to North American, European and Asian commercial markets. While its base operation is located in Bothell, Washington with 290 employees, several regional sales teams are located throughout their market base."

```

    }
  ]
}

} `;
//Open the default document in Document Editor
this.$refs.documenteditor.open(sfdt);
}
}
</script>
<style>
  @import "../node_modules/@syncfusion/ej2-vue-
documenteditor/styles/material.css";
</style>
```

```
{% previewsample "page.domainurl/code-snippet/document-editor/replace-all-cs1" %}
```

## Keyboard shortcut in Vue Document editor component

## Text formatting

The following table lists the default keyboard shortcuts in Document Editor for formatting text:

Key combination	Description
----- -----	
Ctrl + B	Toggles the bold property of selected text.
Ctrl + I	Toggles the italic property of selected text.
Ctrl + U	Toggles the underline property of selected text.
Ctrl + +	Toggles the subscript formatting of selected text.
Ctrl + Shift + +	Toggles the superscript formatting of selected contents.
Ctrl + }	Increases the actual font size of selected text by one point.
Ctrl + {	Decreases the actual font size of selected text by one point.

## Paragraph formatting

The following table lists the default keyboard shortcuts for formatting the paragraph:

Key combination	Description
Ctrl + E	Selected paragraphs are center aligned.
Ctrl + J	Selected paragraphs are justified.
Ctrl + L	Selected paragraphs are left aligned.



|Ctrl + R | Selected paragraphs are right aligned. |

|Ctrl + 1 | Single line spacing is applied for selected paragraphs. |

|Ctrl + 5 | 1.5 line spacing is applied for selected paragraphs. |

|Ctrl + 2 | Double spacing is applied for selected paragraphs. |

|Ctrl + 0 | No spacing is applied before the selected paragraphs. |

|Ctrl + M | Increases the left indent of selected paragraphs by a factor of 36 points. |

|Ctrl + Shift + M | Decreases the left indent of selected paragraphs by a factor of 36 points. |

|Ctrl + \* | Show/Hide the hidden characters like spaces, tab, paragraph marks, and breaks. |

#### Clipboard

Key Combination	Description
----- -----	
Ctrl + C	Copies selected contents to the clipboard.
Ctrl + V	Pastes plain text content from the clipboard.
Ctrl + X	Moves selected content to the clipboard.

#### Keyboard shortcut to navigate around the document

Key Combination	Description
----- -----	
Left arrow	Moves the cursor position one character to the left.
Right arrow	Moves the cursor position one character to the right.
Down arrow	Moves the cursor position down one line.
Up arrow	Moves the cursor position up one line.
Ctrl + Left arrow	Moves the cursor position one word to the left.
Ctrl + Right arrow	Moves the cursor position one word to the right.
Ctrl + Up arrow	Moves the cursor position one paragraph up.
Ctrl + Down arrow	Moves the cursor position one paragraph down.
Tab (in table)	Moves the cursor position one cell to the right.
Shift + Tab (in table)	Moves the cursor position one cell to the left.
Home	Moves the cursor position to the start of a line.
End	Moves the cursor position to the end of a line.
Page up	Moves the cursor position one screen up.
Page down	Moves the cursor position one screen down.
Ctrl + Home	Moves the cursor position to the start of a document.
Ctrl + End	Moves the cursor position to the end of a document.

### Keyboard shortcut to extend selection

Key Combination	Description
-----------------	-------------

-----	-----
-------	-------

Shift + Left arrow	Extends selection one character to the left.
--------------------	----------------------------------------------

Shift + Right arrow	Extends selection one character to the right.
---------------------	-----------------------------------------------

Shift + Down arrow	Extends selection one line downward.
--------------------	--------------------------------------

Shift + Up arrow	Extends selection one line upward.
------------------	------------------------------------

Shift + Home	Extends selection to the start of a line.
--------------	-------------------------------------------

Shift + End	Extends Selection to the end of a line.
-------------	-----------------------------------------

Ctrl + A	Extends selection to the entire document.
----------	-------------------------------------------

Ctrl + Shift + Left arrow	Extends selection one word to the left.
---------------------------	-----------------------------------------

Ctrl + Shift + Right arrow	Extends selection one word to the right.
----------------------------	------------------------------------------

Ctrl + Shift + Down arrow	Extends selection to the end of a paragraph.
---------------------------	----------------------------------------------

Ctrl + Shift + Up arrow	Extends selection to the start of a paragraph.
-------------------------	------------------------------------------------

Ctrl + Shift + Home	Extends selection to the start of a document.
---------------------	-----------------------------------------------

Ctrl + Shift + End	Extends selection to the end of a document.
--------------------	---------------------------------------------

### Find and Replace

Key Combination	Description
-----------------	-------------

-----	-----
-------	-------

Ctrl + F	Opens options pane.
----------	---------------------

Ctrl + H	Opens replace tab in options pane.
----------	------------------------------------

### Create, Save and Print document

Key Combination	Description
-----------------	-------------

-----	-----
-------	-------

Ctrl + N	Opens empty document.
----------	-----------------------

Ctrl + S	Saves the document in SFDt format.
----------	------------------------------------

Ctrl + P	Prints the document.
----------	----------------------

### Edit Operation

Key Combination	Description
-----------------	-------------

-----	-----
-------	-------

Backspace	Deletes one character to the left.
-----------	------------------------------------

Delete	Deletes one character to the right.
--------	-------------------------------------

Ctrl + Z	Undo last performed action.
----------	-----------------------------

Ctrl + Y	Redo last undo action.
----------	------------------------

### Insert special characters

Key Combination	Description
----- -----	
Ctrl + Enter	Inserts page break.
Shift + Enter	Inserts line break.

### Dialog

Key Combination	Description
----- -----	
Ctrl + F	Opens options pane.
Ctrl + D	Opens font dialog.
Ctrl + K	Opens hyperlink dialog.

### See Also

- [How to override the keyboard shortcuts](#)

## Scrolling zooming in Vue Document editor component

The Document Editor renders the document as page by page. You can scroll through the pages by mouse wheel or touch interactions. You can also scroll through the page by using 'scrollToPage()' method of Document Editor instance. Refer to the following code example.

### APP.VUE

```
<template>
  <div id="app">
    <ejs-documenteditor ref="documenteditor" height="370px"
style="width: 100%;"></ejs-documenteditor>
  </div>
</template>
<script>
  import Vue from 'vue'
  import { DocumentEditorPlugin } from '@syncfusion/ej2-vue-
documenteditor';
  Vue.use(DocumentEditorPlugin);
  export default {
    data: function() {
      return {
    },
    mounted: function() {
      let defaultDocument: object = {
        "sections": [
          {
            "blocks": [
              {
                "paragraphFormat": {
                  "styleName": "Normal"
                },
            },
            "inlines": [
```

```

        {
            "text": "First page"
        }
    ],
    },
    "headersFooters": {},
},
{
    "blocks": [
        {
            "paragraphFormat": {
                "styleName": "Normal"
            },
            "inlines": [
                {
                    "text": "Second page"
                }
            ]
        }
    ],
    "headersFooters": {},
},
],
"characterFormat": {},
"paragraphFormat": {},
"background": {
    "color": "#FFFFFF"
},
"styles": [
    {
        "type": "Paragraph",
        "name": "Normal",
        "next": "Normal"
    },
    {
        "type": "Character",
        "name": "Default Paragraph Font"
    }
]
}
this.$refs.documenteditor.open(JSON.stringify(defaultDocument));
//Scroll to specified page.
this.$refs.documenteditor.scrollToPage(2);
}
</script>
<style>
    @import "../node_modules/@syncfusion/ej2-vue-
documenteditor/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/document-editor/scrolling-zooming-cs1" %}

Calling this method brings the specified page into view but doesn't move selection. Hence this method will work by default. That is, it works even if selection is not enabled.

In case, if you wish to move the selection to any page in Document Editor and bring it into view, you can use 'goToPage()' method of selection instance. Refer to the following code example.

#### APP.VUE

```
<template>
  <div id="app">
    <ejs-documenteditor ref="documenteditor" :enableSelection='true'
    :isReadOnly=false height="370px" style="width: 100%;"></ejs-documenteditor>
  </div>
</template>
<script>
  import Vue from 'vue'
  import { DocumentEditorPlugin, Selection } from '@syncfusion/ej2-vue-
documenteditor';
  Vue.use(DocumentEditorPlugin);
  export default {
    data: function() {
      return {
      };
    },
    provide: {
      DocumentEditor : [Selection]
    },
    mounted: function() {
      let defaultDocument: object = {
        "sections": [
          {
            "blocks": [
              {
                "paragraphFormat": {
                  "styleName": "Normal"
                },
                "inlines": [
                  {
                    "text": "First page"
                  }
                ]
              }
            ],
            "headersFooters": {},
          },
          {
            "blocks": [
              {
                "paragraphFormat": {
                  "styleName": "Normal"
                },
                "inlines": [
                  {
                    "text": "Second page"
                  }
                ]
              }
            ],
            "headersFooters": {},
          },
        ],
      },
    },
  }
}
```

```

        {
            "blocks": [
                {
                    "paragraphFormat": {
                        "styleName": "Normal"
                    },
                    "inlines": [
                        {
                            "text": "Third page"
                        }
                    ]
                }
            ],
            "headersFooters": {},
        }
    ],
    "characterFormat": {},
    "paragraphFormat": {},
    "background": {
        "color": "#FFFFFF"
    },
    "styles": [
        {
            "type": "Paragraph",
            "name": "Normal",
            "next": "Normal"
        },
        {
            "type": "Character",
            "name": "Default Paragraph Font"
        }
    ]
}
this.$refs.documenteditor.open(JSON.stringify(defaultDocument));
//Navigate to specified page.
this.$refs.documenteditor.ej2Instances.selection.goToPage(3);
}
}
</script>
<style>
    @import "../node_modules/@syncfusion/ej2-vue-
documenteditor/styles/material.css";
</style>

```

```
{% previewsample "page.domainurl/code-snippet/document-editor/scrolling-zooming-cs2" %}
```

### Zooming

You can scale the contents in Document Editor ranging from 10% to 500% of the actual size. You can achieve this using mouse or touch interactions. You can also use 'zoomFactor' property of Document Editor instance. The value can be specified in a range from 0.1 to 5. Refer to the following code example.

```

<template>
<div id="app">

```

```
<ejs-documenteditor ref="documenteditor" style="width: 100%;height: 100%;"></ejs-documenteditor>
</div>
</template>
<script>
import Vue from 'vue'
import { DocumentEditorPlugin } from '@syncfusion/ej2-vue-documenteditor';
Vue.use(DocumentEditorPlugin);
export default {
  data: function() {
    return {
    };
  },
  mounted: function() {
    //Set zoom factor.
    this.$refs.documenteditor.zoomFactor = 3;
  }
}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-vue-documenteditor/styles/material.css";
</style>
`
```

### Page Fit Type

Apart from specifying the zoom factor as value, the Document Editor provides option to specify page fit options such as fit to full page or fit to page width. You can set this option using 'fitPage' method of Document Editor instance. Refer to the following code example.

```
`
<template>
<div id="app">
<ejs-documenteditor ref="documenteditor" height="370px" style="width: 100%;"></ejs-
documenteditor>
</div>
</template>
<script>
```

```

import Vue from 'vue'
import { DocumentEditorPlugin } from '@syncfusion/ej2-vue-documenteditor';
Vue.use(DocumentEditorPlugin);
export default {
  data: function() {
    return {
    };
  },
  mounted: function() {
    //Set zoom factor to fit page width.
    this.$refs.documenteditor.fitPage('FitPageWidth');
  }
}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-vue-documenteditor/styles/material.css";
</style>
`

```

### Zoom option using UI

The following code example shows how to provide zoom options in Document Editor.

#### APP.VUE

```

<template>
  <div id="app">
    <ejs-documenteditor ref="documenteditor" v-
bind:selectionChange='onSelectionChange' v-bind:viewChange='onViewChange' v-
bind:documentChange='onDocumentChange' :enableSelection='true'
:isReadOnly=false height="370px" style="width: 100%;></ejs-documenteditor>
    <div id="documenteditor_statusbar">
      <label style="margin-top: 6px;margin-right: 2px">Page
</label>
      <div class="single-line e-de-pagenumbers-text" v-
on:keydown='pageKeydownEvent' v-on:click='pagerClickEvent'
id="editablePageNumber" style="font-size:12px;font-weight: 700;display:
inline-flex;height: 17px;padding: 0px 4px;" contenteditable="false">
        <label id="documenteditor_page_no" style="text-
transform:capitalize;white-space:pre;overflow:hidden;user-
select:none;cursor:text;height:17px;max-width:150px">{{currentPage}}</label>
      </div>
      <label id="documenteditor_pagecount_separator"
style="margin-left:2px;letter-spacing: 1.05px">of</label>
      <label id="documenteditor_pagecount" style="margin-
left:6px;letter-spacing: 1.05px">{{pageCount}}</label>
    </div>
  </div>

```



```

        <ejs-dropdownbutton ref="de_zoom" :items="zoomItems"
class="e-de-statusbar-zoom" :content="zoomContent" v-bind:select="onZoom"
title="Zoom level. Click or tap to open the Zoom options."></ejs-
dropdownbutton>
    </div>
</div>
</template>
<script>
    import Vue from 'vue'
    import { DocumentEditorPlugin, Selection } from '@syncfusion/ej2-vue-
documenteditor';
    import { DropDownButtonPlugin } from '@syncfusion/ej2-vue-splitbuttons';
    Vue.use(DocumentEditorPlugin);
    Vue.use(DropDownButtonPlugin);
    export default {
        props: ['pageCount', 'currentPage', 'zoomContent'],
        data: function() {
            return {
                zoomContent: "100%",
                zoomItems: [
                    {
                        text: '150%',
                    },
                    {
                        text: '125%',
                    },
                    {
                        text: '100%',
                    },
                    {
                        text: '75%',
                    },
                    {
                        text: '50%',
                    },
                    {
                        separator: true
                    },
                    {
                        text: 'Fit one page'
                    },
                    {
                        text: 'Fit page width',
                    },
                ],
                pageCount: 1,
                currentPage: 1
            };
        },
        provide: {
            //Inject require modules.
            DocumentEditor : [Selection]
        },
        methods :{
            onViewChange: function(args) {
                //Update page number on view change.
                this.currentPage = args.startPage;
            },

```

```

        onSelectionChange : function(args) {
            //Get current page number.
            this.currentPage =
this.$refs.documenteditor.ej2Instances.selection.startPage;
        }
        onDocumentChange: function() {
            //Update page count.
            this.pageCount =
this.$refs.documenteditor.ej2Instances.pageCount;
            //Update zoom factor.
            this.zoomContent =
Math.round(this.$refs.documenteditor.ej2Instances.zoomFactor * 100) + '%';
        },
        onZoom: function (args) {
            let zoomValue = args.item.text;
            if (zoomValue.match('Fit one page')) {

this.$refs.documenteditor.ej2Instances.fitPage('FitOnePage');
            } else if (zoomValue.match('Fit page width')) {

this.$refs.documenteditor.ej2Instances.fitPage('FitPageWidth');
            } else {
                this.$refs.documenteditor.ej2Instances.zoomFactor =
parseInt(zoomValue, 0) / 100;
            }
            this.zoomContent =
Math.round(this.$refs.documenteditor.ej2Instances.zoomFactor * 100) + '%';
        },
        pageKeydownEvent: function (e) {
            if (e.which === 13) {
                e.preventDefault();
                let pageNumber =
parseInt(document.getElementById("editablePageNumber").textContent, 0);
                if (pageNumber >
this.$refs.documenteditor.ej2Instances.pageCount) {
                    this.updatePageNumber();
                } else {

this.$refs.documenteditor.ej2Instances.selection.goToPage(parseInt(document.
getElementById("editablePageNumber").textContent, 0));
                }

document.getElementById("editablePageNumber").contentEditable = 'false';
                if
(document.getElementById("editablePageNumber").textContent === '') {
                    this.updatePageNumber();
                }
            }
            if (e.which > 64) {
                e.preventDefault();
            }
        },
        pageBlurEvent: function () {
            if
(document.getElementById("editablePageNumber").textContent === '' ||
parseInt(document.getElementById("editablePageNumber").textContent, 0) >
this.$refs.documenteditor.ej2Instances.pageCount) {

```

```

        this.updatePageNumber();
    }

    document.getElementById("editablePageNumber").contentEditable = 'false';
    },
    pagerClickEvent: function () {

    document.getElementById("editablePageNumber").contentEditable = 'true';
    document.getElementById("editablePageNumber").focus();

    window.getSelection().selectAllChildren(document.getElementById("editablePage
eNumber"));
    },
    updatePageNumber: function () {
        this.currentPage =
this.$refs.documenteditor.ej2Instances.selection.startPage.toString();
    }
    },
    mounted: function() {
        let defaultDocument: object = {
            "sections": [
                {
                    "blocks": [
                        {
                            "paragraphFormat": {
                                "styleName": "Normal"
                            },
                            "inlines": [
                                {
                                    "text": "First page"
                                }
                            ]
                        }
                    ],
                    "headersFooters": {},
                },
                {
                    "blocks": [
                        {
                            "paragraphFormat": {
                                "styleName": "Normal"
                            },
                            "inlines": [
                                {
                                    "text": "Second page"
                                }
                            ]
                        }
                    ],
                    "headersFooters": {},
                },
                {
                    "blocks": [
                        {
                            "paragraphFormat": {
                                "styleName": "Normal"
                            }

```

```

        "inlines": [
            {
                "text": "Third page"
            }
        ]
    },
    "headersFooters": {},
}
],
"characterFormat": {},
"paragraphFormat": {},
"background": {
    "color": "#FFFFFF"
},
"styles": [
    {
        "type": "Paragraph",
        "name": "Normal",
        "next": "Normal"
    },
    {
        "type": "Character",
        "name": "Default Paragraph Font"
    }
]
}
//Open default document in Document Editor.
this.$refs.documenteditor.open(JSON.stringify(defaultDocument));

document.getElementById('editablePageNumber').addEventListener('blur', this.pageBlurEvent);
}
}
</script>
<style>
    @import "../node_modules/@syncfusion/ej2-vue-documenteditor/styles/material.css";
    .single-line {
        cursor: text !important;
        outline: none;
    }
    .single-line:hover {
        border-color: #e4e4e4 !important;
    }
    [contenteditable="true"].single-line {
        white-space: nowrap;
        border-color: #e4e4e4 !important;
    }
    .e-de-pagenunder-text:hover {
        background-color: #f4f4f4 !important;
    }
    [contenteditable="true"].single-line * {
        white-space: nowrap;
    }
    .e-de-statusbar-zoom {
        float: right;

```

```

        text-align: center;
        padding: 2px;
        line-height: 19px;
        margin-top: 1px;
    }
</style>

```

{% previewsample "page.domainurl/code-snippet/document-editor/scrolling-zooming-cs3" %}

## Print in Vue Document editor component

To print the document, use the [print](#) method from Document Editor instance.

Refer to the following example for showing a document and print it.

### APP.VUE

```

<template>
  <div id="app">
    <div>
      <button v-on:click='print' >Print</button>
    </div>
    <ejs-documenteditor ref="documenteditor" :enablePrint='true'
height="370px" style="width: 100%;"></ejs-documenteditor>
  </div>
</template>
<script>
  import Vue from 'vue'
  import { DocumentEditorPlugin, Print } from '@syncfusion/ej2-vue-
documenteditor';
  Vue.use(DocumentEditorPlugin);
  export default {
    data: function() {
      return {
      };
    },
    provide: {
      DocumentEditor : [Print]
    },
    methods: {
      print: function() {
        //Print the content of the Document Editor.
        this.$refs.documenteditor.print();
      }
    },
    mounted: function() {
      let sfdt: string = `{
        "sections": [
          {
            "blocks": [
              {
                "inlines": [
                  {
                    "characterFormat": {
                      "bold": true,
                      "italic": true
                    },
                    "text": "Hello World"

```

```

    }
    ],
    },
    ],
    "headersFooters": {
    }
  }
]
} `;
//Open default document in Document Editor.
this.$refs.documenteditor.open(sfdt);
}
}
</script>
<style>
  @import "../node_modules/@syncfusion/ej2-vue-
documenteditor/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/document-editor/print-cs1" %}

Refer to the following example for creating a document and print it.

#### APP.VUE

```

<template>
  <div id="app">
    <div>
      <button v-on:click='print' >Print</button>
    </div>
    <ejs-documenteditor ref="documenteditor" :isReadOnly='false'
:enablePrint='true' :enableEditor='true' :enableSelection='true'
:enableEditorHistory='true' height="370px" style="width: 100%;"></ejs-
documenteditor>
  </div>
</template>
<script>
  import Vue from 'vue'
  import { DocumentEditorPlugin, Print, Editor, Selection, EditorHistory }
from '@syncfusion/ej2-vue-documenteditor';
  Vue.use(DocumentEditorPlugin);
  export default {
    data: function() {
      return {
      };
    },
    provide: {
      //Inject require modules.
      DocumentEditor : [Print, Editor, Selection, EditorHistory]
    },
    methods: {
      print: function() {
        //Print the content of the Document Editor.
        this.$refs.documenteditor.print();
      }
    }
  }
}

```

```
</script>
<style>
  @import "../node_modules/@syncfusion/ej2-vue-
documenteditor/styles/material.css";
</style>
```

```
{% previewsample "page.domainurl/code-snippet/document-editor/print-cs2" %}
```

### Improve print quality

Document editor provides an option to improve the print quality using [printDevicePixelRatio](#) in Document editor settings. Document editor using canvas approach to render content. Then, canvas are converted to image and it process for print. Using printDevicePixelRatio API, you can increase the image quality based on your requirement.

The following example code illustrates how to improve the print quality in Document editor container.

```
<template>
<div id="app">
  <ejs-documenteditorcontainer ref='documenteditor' :serviceUrl='serviceUrl'
:documentEditorSettings='settings' height="590px" id='container' :enableToolbar='true'></ejs-
documenteditorcontainer>
</div>
</template>
<script>
import Vue from 'vue';

import { DocumentEditorContainerPlugin, DocumentEditorContainerComponent,Toolbar} from
'@syncfusion/ej2-vue-documenteditor';

Vue.use(DocumentEditorContainerPlugin);

export default {
  data() {
    return { serviceUrl:'https://ej2services.syncfusion.com/production/web-services/api/documenteditor/',
settings:{ printDevicePixelRatio: 2 } };
  },
  provide: {
    //Inject require modules.
    DocumentEditorContainer: [Toolbar]
  }
}
</script>
```

,

Note: By default, printDevicePixelRatio value is 1.

#### Print using window object

You can print the document in Document Editor by passing the window instance. This is useful to implement print in third party frameworks such as electron, where the window instance will not be available. Refer to the following example.

```
this.$refs.documenteditor.print(window);
```

#### Page setup

Some of the print options cannot be configured using JavaScript. Refer to the following links to learn more about the browser page setup:

- [Chrome](#)
- [Firefox](#)

However, you can customize margins, paper, and layout options by modifying the section format properties using page setup dialog

,

```
<template>
<div id="app">
<ejs-documenteditor ref="documenteditor" :isReadOnly='false' :enablePrint='true' :enableEditor='true'
:enableSelection='true' :enableEditorHistory='true' :enablePageSetupDialog='true' height="370px"
style="width: 100%;"></ejs-documenteditor>
</div>
</template>
<script>
import Vue from 'vue'

import { DocumentEditorPlugin, Print, Editor, Selection, EditorHistory, PageSetupDialog } from
'@syncfusion/ej2-vue-documenteditor';

Vue.use(DocumentEditorPlugin);

export default {
data: function() {
return {
};
},
provide: {
DocumentEditor : [Print, Editor, Selection, EditorHistory, PageSetupDialog]
},

```



```

mounted: function() {
this.$refs.documenteditor.showPageSetupDialog();
}
}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-vue-documenteditor/styles/material.css";
</style>
`

```

By customizing margins, papers, and layouts, the layout of the document will be changed in Document Editor. To modify these options during print operation, serialize the document as SFDT using the [serialize](#) method in Document Editor instance and open the SFDT data in another instance of Document Editor in separate window.

The following example shows how to customize layout options only for printing.

#### APP.VUE

```

<template>
  <div id="app">
    <div>
      <button v-on:click='print' >Print</button>
    </div>
    <ejs-documenteditor ref="documenteditor" :isReadOnly='false'
:enablePrint='true' :enableEditor='true' :enableSelection='true'
:enableEditorHistory='true' :enableSfdtExport='true' height="370px"
style="width: 100%;"></ejs-documenteditor>
    <ejs-documenteditor ref="pagesetup_documenteditor"
:isReadOnly='false' :enablePrint='true' :enableEditor='true'
:enableSelection='true' :enableEditorHistory='true' :enableSfdtExport='true'
height="370px" style="width: 100%;"></ejs-documenteditor>
  </div>
</template>
<script>
import Vue from 'vue'
import { DocumentEditorPlugin, Print, Editor, Selection, EditorHistory,
SfdtExport } from '@syncfusion/ej2-vue-documenteditor';
Vue.use(DocumentEditorPlugin);
export default {
  data: function() {
    return {
    };
  },
  provide: {
    DocumentEditor : [Print, Editor, Selection, EditorHistory,
SfdtExport]
  },
  methods: {
    print: function() {
      let sfdtData = this.$refs.documenteditor.serialize();

```

```

        this.$refs.pagesetup_documenteditor.open(sfddata);
        //Set A5 paper size

this.$refs.pagesetup_documenteditor.ej2Instances.selection.sectionFormat.pag
eWidth = 419.55;

this.$refs.pagesetup_documenteditor.ej2Instances.selection.sectionFormat.pag
eHeight = 595.30;
        this.$refs.pagesetup_documenteditor.print();
    }
}
}
</script>
<style>
    @import "../node_modules/@syncfusion/ej2-vue-
documenteditor/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/document-editor/print-cs3" %}

### Dialog in Vue Document editor component

Document Editor provides dialog support to major operations such as insert or edit hyperlink, formatting text, paragraph, style, list and table properties.

#### Font Dialog

Font dialog allows you to modify all text properties for selected contents at once such as bold, italic, underline, font size, font color, strikethrough, subscript and superscript.

Refer to the following example.

#### APP.VUE

```

<template>
  <div id="app">
    <div>
      <button v-on:click='showFontDialog' >Open dialog</button>
    </div>
    <ejs-documenteditor ref="documenteditor" :enableSelection='true'
:isReadOnly='false' :enableEditor='true' :enableFontDialog='true'
:enableSfdtExport='true' height="370px" style="width: 100%;"></ejs-
documenteditor>
  </div>
</template>
<script>
  import Vue from 'vue'
  import { DocumentEditorPlugin, Selection, Editor, FontDialog,
SfdtExport } from '@syncfusion/ej2-vue-documenteditor';
  Vue.use(DocumentEditorPlugin);
  export default {
    data: function() {
      return {
      };
    },
    provide: {
      //Inject require modules.
      DocumentEditor : [Selection, Editor, FontDialog, SfdtExport]
    }
  }

```

```

    }
    methods: {
      showFontDialog: function() {
        //Open the font dialog.
        this.$refs.documenteditor.showDialog('Font');
      }
    }
  }
</script>
<style>
  @import "../node_modules/@syncfusion/ej2-vue-
documenteditor/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/document-editor/dialog-cs1" %}

### Paragraph dialog

This dialog allows modifying the paragraph formatting for selection at once such as text alignment, indentation, and spacing.

To open this dialog, refer to the following example.

### APP.VUE

```

<template>
  <div id="app">
    <div>
      <button v-on:click='showParagraphDialog' >Open dialog</button>
    </div>
    <ejs-documenteditor ref="documenteditor" :enableSelection='true'
:isReadOnly='false' :enableEditor='true' :enableParagraphDialog='true'
:enableSfdtExport='true' height="370px" style="width: 100%;"></ejs-
documenteditor>
    </div>
  </template>
<script>
  import Vue from 'vue'
  import { DocumentEditorPlugin, Selection, Editor, ParagraphDialog,
SfdtExport } from '@syncfusion/ej2-vue-documenteditor';
  Vue.use(DocumentEditorPlugin);
  export default {
    data: function() {
      return {
      };
    },
    provide: {
      //Inject require modules.
      DocumentEditor : [Selection, Editor, ParagraphDialog,
SfdtExport]
    },
    methods: {
      showParagraphDialog: function() {
        //Open the paragraph dialog.
        this.$refs.documenteditor.showDialog('Paragraph');
      }
    }
  }

```

```

    }
  </script>
  <style>
    @import ".../node_modules/@syncfusion/ej2-vue-
documenteditor/styles/material.css";
  </style>

```

{% previewsample "page.domainurl/code-snippet/document-editor/dialog-cs2" %}

### Table dialog

This dialog allows creating and inserting a table at cursor position by specifying the required number of rows and columns.

To open this dialog, refer to the following example.

### APP.VUE

```

<template>
  <div id="app" style="height:400px">
    <div>
      <button v-on:click='showTableDialog' >Open dialog</button>
    </div>
    <ejs-documenteditor ref="documenteditor" :enableSelection='true'
:isReadOnly='false' :enableEditor='true' :enableTableDialog='true'
:enableSfdtExport='true' style="width: 100%;" height="370px" ></ejs-
documenteditor>
  </div>
</template>
<script>
  import Vue from 'vue'
  import { DocumentEditorPlugin, Selection, Editor, TableDialog,
SfdtExport } from '@syncfusion/ej2-vue-documenteditor';
  Vue.use(DocumentEditorPlugin);
  export default {
    data: function() {
      return {
      };
    },
    provide: {
      //Inject require modules
      DocumentEditor : [Selection, Editor, TableDialog, SfdtExport]
    },
    methods: {
      showTableDialog: function() {
        //Open the table dialog.
        this.$refs.documenteditor.showDialog('Table');
      }
    }
  }
</script>
<style>
  @import ".../node_modules/@syncfusion/ej2-vue-
documenteditor/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/document-editor/dialog-cs3" %}

## Bookmark dialog

This dialog allows you to perform the following operations:

- View all bookmarks.
- Navigate to a bookmark.
- Create a bookmark at current selection.
- Delete an existing bookmark.

To open this dialog, refer to the following example.

### APP.VUE

```
<template>
  <div id="app" style="height:400px">
    <div>
      <button v-on:click='showBookmarkDialog' >Open
dialog</button>
    </div>
    <ejs-documenteditor ref="documenteditor" :enableSelection='true'
:isReadOnly='false' :enableEditor='true' :enableBookmarkDialog='true'
:enableSfdtExport='true' height="370px" style="width: 100%;"></ejs-
documenteditor>
  </div>
</template>
<script>
  import Vue from 'vue'
  import { DocumentEditorPlugin, Selection, Editor, BookmarkDialog,
SfdtExport } from '@syncfusion/ej2-vue-documenteditor';
  Vue.use(DocumentEditorPlugin);
  export default {
    data: function() {
      return {
      };
    },
    provide: {
      //Inject require modules
      DocumentEditor : [Selection, Editor, BookmarkDialog,
SfdtExport]
    },
    methods: {
      showBookmarkDialog: function() {
        //Open the bookmark dialog.
        this.$refs.documenteditor.showDialog('Bookmark');
      }
    }
  }
</script>
<style>
  @import "../node_modules/@syncfusion/ej2-vue-
documenteditor/styles/material.css";
</style>
```

{% previewsample "page.domainurl/code-snippet/document-editor/dialog-cs4" %}

### Hyperlink dialog

This dialog allows editing or inserting a hyperlink at cursor position.

To open this dialog, refer to the following example.

#### APP.VUE

```
<template>
  <div id="app" style="height:400px">
    <div>
      <button v-on:click='showHyperlinkDialog' >Open
dialog</button>
    </div>
    <ejs-documenteditor ref="documenteditor" :enableSelection='true'
:isReadOnly='false' :enableEditor='true' :enableHyperlinkDialog='true'
:enableSfdtExport='true' height="370px" style="width: 100%;"></ejs-
documenteditor>
  </div>
</template>
<script>
import Vue from 'vue'
import { DocumentEditorPlugin, Selection, Editor, HyperlinkDialog,
SfdtExport } from '@syncfusion/ej2-vue-documenteditor';
Vue.use(DocumentEditorPlugin);
export default {
  data: function() {
    return {
    };
  },
  provide: {
    //Inject require modules
    DocumentEditor : [Selection, Editor, HyperlinkDialog,
SfdtExport]
  },
  methods: {
    showHyperlinkDialog: function() {
      //Opens hyperlink dialog.
      this.$refs.documenteditor.showDialog('Hyperlink');
    }
  }
}
</script>
<style>
  @import "../node_modules/@syncfusion/ej2-vue-
documenteditor/styles/material.css";
</style>
```

{% previewsample "page.domainurl/code-snippet/document-editor/dialog-cs5" %}

### Table of contents dialog

This dialog allows creating and inserting table of contents at cursor position. If the table of contents already exists at cursor position, you can customize its properties.

To open this dialog, refer to the following example.

,

```

<template>
<div id="app" style="height:400px">
<div>
<button v-on:click='showTableOfContentsDialog' >Open dialog</button>
</div>
<ejs-documenteditor ref="documenteditor" :enableSelection='true' :isReadOnly='false'
:enableEditor='true' :enableTableOfContentsDialog='true' :enableSfdtExport='true' height="370px"
style="width: 100%;"></ejs-documenteditor>
</div>
</template>
<script>
import Vue from 'vue'

import { DocumentEditorPlugin, Selection, Editor, TableOfContentsDialog, SfdtExport } from
'@syncfusion/ej2-vue-documenteditor';

Vue.use(DocumentEditorPlugin);

export default {
data: function() {
return {
};
},
provide: {
//Inject require modules
DocumentEditor : [Selection, Editor, TableOfContentsDialog, SfdtExport]
}
methods: {
showTableOfContentsDialog: function() {
//Open the table of contents dialog.
this.$refs.documenteditor.showDialog('TableOfContents');
}
}
}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-vue-documenteditor/styles/material.css";

```

```
</style>
```

```
,
```

### Styles Dialog

This dialog allows managing the styles in a document. It will display all the styles in the document with options to modify the properties of the existing style or create new style with the help of 'Style dialog'. Refer to the following example.

```
,
```

```
<template>
```

```
<div id="app" style="height:400px">
```

```
<div>
```

```
<button v-on:click='showStylesDialog' >Open dialog</button>
```

```
</div>
```

```
<ejs-documenteditor ref="documenteditor" :enableSelection='true' :isReadOnly='false'
:enableEditor='true' :enableStylesDialog='true' :enableSfdtExport='true' :enableStyleDialog='true'
height="370px" style="width: 100%;"></ejs-documenteditor>
```

```
</div>
```

```
</template>
```

```
<script>
```

```
import Vue from 'vue'
```

```
import { DocumentEditorPlugin, Selection, Editor, StylesDialog, StyleDialog, SfdtExport } from
'@syncfusion/ej2-vue-documenteditor';
```

```
Vue.use(DocumentEditorPlugin);
```

```
export default {
```

```
data: function() {
```

```
return {
```

```
};
```

```
},
```

```
provide: {
```

```
//Inject require modules
```

```
DocumentEditor : [Selection, Editor, StylesDialog, StyleDialog, SfdtExport]
```

```
}
```

```
methods: {
```

```
showStylesDialog: function() {
```

```
//Open the styles dialog.
```

```
this.$refs.documenteditor.showDialog('Styles');
```



```

}
}
}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-vue-documenteditor/styles/material.css";
</style>
`

```

### Style dialog

You can directly use this dialog for modifying any existing style or add new style by providing the style name.

To open this dialog, refer to the following example.

```

`
<template>
<div id="app" style="height:400px">
<div>
<button v-on:click='showStyleDialog' >Open dialog</button>
</div>
<ejs-documenteditor ref="documenteditor" :enableSelection='true' :isReadOnly='false'
:enableEditor='true' :enableSfdtExport='true' :enableStyleDialog='true' height="370px" style="width:
100%;"></ejs-documenteditor>
</div>
</template>
<script>
import Vue from 'vue'

import { DocumentEditorPlugin, Selection, Editor, StyleDialog, SfdtExport } from '@syncfusion/ej2-vue-
documenteditor';

Vue.use(DocumentEditorPlugin);

export default {
data: function() {
return {
};
},
provide: {
//Inject require modules

```

```

DocumentEditor : [Selection, Editor, StyleDialog, SfdtExport]
}
methods: {
  showStylesDialog: function() {
    //Open styles dialog.
    this.$refs.documenteditor.showDialog('Style');
  }
}
}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-vue-documenteditor/styles/material.css";
</style>
`

```

### List dialog

This dialog allows creating a new list or modifying existing lists in the document.

To open this dialog, refer to the following example.

```

<template>
<div id="app" style="height:400px">
<div>
<button v-on:click='showListDialog' >Open dialog</button>
</div>
<ejs-documenteditor ref="documenteditor" :enableSelection='true' :isReadOnly='false'
:enableEditor='true' :enableSfdtExport='true' :enableListDialog='true' height="370px" style="width:
100%;"></ejs-documenteditor>
</div>
</template>
<script>
import Vue from 'vue'

import { DocumentEditorPlugin, Selection, Editor, ListDialog, SfdtExport } from '@syncfusion/ej2-vue-
documenteditor';

Vue.use(DocumentEditorPlugin);

export default {
`

```

```

data: function() {
  return {
  };
},
provide: {
  //Inject require modules
  DocumentEditor : [Selection, Editor, ListDialog, SfdtExport]
}
methods: {
  showListDialog: function() {
    //Open list dialog.
    this.$refs.documenteditor.showDialog('List');
  }
}
}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-vue-documenteditor/styles/material.css";
</style>
`

```

### Borders and shading dialog

This dialog allows customizing the border style, border width, and background color of the table or selected cells.

To open this dialog, refer to the following example.

```

`
<template>
<div id="app" style="height:400px">
<div>
<button v-on:click='showBordersAndShadingDialog' >Open dialog</button>
</div>
<ejs-documenteditor ref="documenteditor" :enableSelection='true' :isReadOnly='false'
:enableEditor='true' :enableSfdtExport='true' :enableBordersAndShadingDialog='true' height="370px"
style="width: 100%;"></ejs-documenteditor>
</div>
`

```

```

</template>
<script>
import Vue from 'vue'

import { DocumentEditorPlugin, Selection, Editor, BordersAndShadingDialog, SfdtExport } from
 '@syncfusion/ej2-vue-documenteditor';

Vue.use(DocumentEditorPlugin);

export default {
  data: function() {
    return {
    };
  },
  provide: {
    //Inject require modules
    DocumentEditor : [Selection, Editor, BordersAndShadingDialog, SfdtExport]
  }
  methods: {
    showBordersAndShadingDialog: function() {
      //Open borders and shading dialog.
      this.$refs.documenteditor.showDialog('BordersAndShading');
    }
  }
}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-vue-documenteditor/styles/material.css";
</style>

```

### Table options dialog

This dialog allows customizing the default cell margins and spacing between each cells of the selected table.

To open this dialog, refer to the following example.

```

<template>
<div id="app" style="height:400px">

```

```
<div>
<button v-on:click='showTableOptionsDialog' >Open dialog</button>
</div>

<ejs-documenteditor ref="documenteditor" :enableSelection='true' :isReadOnly='false'
:enableEditor='true' :enableSfdtExport='true' :enableTableOptionsDialog='true' height="370px"
style="width: 100%;"></ejs-documenteditor>
</div>
</template>
<script>
import Vue from 'vue'

import { DocumentEditorPlugin, Selection, Editor, TableOptionsDialog, SfdtExport } from
'@syncfusion/ej2-vue-documenteditor';

Vue.use(DocumentEditorPlugin);

export default {
data: function() {
return {
};
},
provide: {
//Inject require modules
DocumentEditor : [Selection, Editor, TableOptionsDialog, SfdtExport]
}
methods: {
showTableOptionsDialog: function() {
//Open table options dialog.
this.$refs.documenteditor.showDialog('TableOptions');
}
},
mounted() {
this.$refs.documenteditor.ej2Instances.editor.insertTable(2,2);
}
}
</script>
<style>
```

```
@import "../../node_modules/@syncfusion/ej2-vue-documenteditor/styles/material.css";
</style>
```

### Table properties dialog

This dialog allows customizing the table, row, and cell properties of the selected table.

To open this dialog, refer to the following example.

```
<template>
<div id="app" style="height:400px">
<div>
<button v-on:click='showTablePropertiesDialog' >Open dialog</button>
</div>
<ejs-documenteditor ref="documenteditor" :enableSelection='true' :isReadOnly='false'
:enableEditor='true' :enableSfdtExport='true' :enableTablePropertiesDialog='true' height="370px"
style="width: 100%;"></ejs-documenteditor>
</div>
</template>
<script>
import Vue from 'vue'

import { DocumentEditorPlugin, Selection, Editor, TablePropertiesDialog, SfdtExport } from
'@syncfusion/ej2-vue-documenteditor';

Vue.use(DocumentEditorPlugin);

export default {
data: function() {
return {
};
},
provide: {
//Inject require modules
DocumentEditor : [Selection, Editor, TablePropertiesDialog, SfdtExport]
}
methods: {
showTablePropertiesDialog: function() {
//Open table properties dialog.
```

```

this.$refs.documenteditor.showDialog('TableProperties');
}
},
mounted() {
this.$refs.documenteditor.ej2Instances.editor.insertTable(2,2);
}
}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-vue-documenteditor/styles/material.css";
</style>
`

```

### Page setup dialog

This dialog allows customizing margins, size, and layout options for pages of the section.

To open this dialog, refer to the following example.

#### APP.VUE

```

<template>
  <div id="app" style="height:400px">
    <div>
      <button v-on:click='showPageSetupDialog' >Open
dialog</button>
    </div>
    <ejs-documenteditor ref="documenteditor" :enableSelection='true'
:isReadOnly='false' :enableEditor='true' :enableSfdtExport='true'
:enablePageSetupDialog='true' height="370px" style="width: 100%;"></ejs-
documenteditor>
    </div>
  </div>
</template>
<script>
  import Vue from 'vue'
  import { DocumentEditorPlugin, Selection, Editor, PageSetupDialog,
SfdtExport } from '@syncfusion/ej2-vue-documenteditor';
  Vue.use(DocumentEditorPlugin);
  export default {
    data: function() {
      return {
      };
    },
    provide: {
      //Inject require modules.
      DocumentEditor : [Selection, Editor, PageSetupDialog,
SfdtExport]
    },
    methods: {
      showPageSetupDialog: function() {
        //Open page setup dialog.

```

```

        this.$refs.documenteditor.showDialog('PageSetup');
    }
}
}
</script>
<style>
    @import "../node_modules/@syncfusion/ej2-vue-
documenteditor/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/document-editor/dialog-cs6" %}

See Also

- [Feature module](#)

## Right to left in Vue Document editor component

Document Editor provides RTL (right-to-left) support. This can be enabled using the “enableRtl” property.

### APP.VUE

```

<template>
  <div id="app">
    <ejs-documenteditor ref='documenteditor' id='container'
pageOutline='#E0E0E0' height="370px" style="width: 100%;"
:enablePrint='true' :enableSfdtExport='true' :enableSelection='true'
:enableContextMenu='true' :enableSearch='true'
enableOptionsPane='true':enableWordExport='true' :enableTextExport='true'
:enableEditor='true' :enableImageResizer='true' :enableEditorHistory='true'
enableHyperlinkDialog='true' :enableTableDialog='true'
:enableBookmarkDialog='true':enableTableOfContentsDialog='true'
:enablePageSetupDialog='true' :enableStyleDialog='true'
:enableListDialog='true' :enableParagraphDialog='true'
:enableFontDialog='true' :enableTablePropertiesDialog='true'
:enableBordersAndShadingDialog='true' :enableTableOptionsDialog='true'
:enableRtl='true' locale='ar-AE'></ejs-documenteditor>
  </div>
</template>
<script>
  import Vue from 'vue';
  import { DocumentEditorPlugin, DocumentEditorComponent, Print,
SfdtExport, WordExport, TextExport, Selection, Search, Editor, ImageResizer,
EditorHistory, ContextMenu, OptionsPane, HyperlinkDialog, TableDialog,
BookmarkDialog, TableOfContentsDialog, PageSetupDialog, StyleDialog,
ListDialog, ParagraphDialog, BulletsAndNumberingDialog, FontDialog,
TablePropertiesDialog, BordersAndShadingDialog, TableOptionsDialog,
CellOptionsDialog, StylesDialog } from '@syncfusion/ej2-vue-documenteditor';
  import { L10n, setCulture } from '@syncfusion/ej2-base';
  Vue.use(DocumentEditorPlugin);
  L10n.load({
    'ar-AE': {
      'documenteditor': {
        'Table': 'الجدول',
        'Row': 'الصف',

```



```

'Cell': 'الخلية',
'Ok': 'موافق',
'Cancel': 'إلغاء الأمر',
'Size': 'حجم',
'Preferred Width': 'العرض المفضل',
'Points': 'نقاط',
'Percent': 'المائه',
'Measure in': 'القياس في',
'Alignment': 'محاذاة',
'Left': 'ليسار',
'Center': 'مركز',
'Right': 'الحق',
'Justify': 'تبرير',
'Indent from left': 'مسافة بادئه من اليسار',
'Borders and Shading': 'الحدود والتظليل',
'Options': 'خيارات',
'Specify height': 'تحديد الارتفاع',
'At least': 'الاقل',
'Exactly': 'تماما',
'Row height is': 'ارتفاع الصف هو',
'Allow row to break across pages': 'السماح بفصل الصف عبر الصفحات',
'Repeat as header row at the top of each page': 'تكرار كصف راس في',
'اعلي كل صفحه',
'Vertical alignment': 'محاذاة عمودي',
'Top': 'أعلى',
'Bottom': 'اسفل',
'Default cell margins': 'هوامش الخلية الافتراضية',
'Default cell spacing': 'تباعد الخلايا الافتراضي',
'Allow spacing between cells': 'السماح بالتباعد بين الخلايا',
'Cell margins': 'هوامش الخلية',
'Same as the whole table': 'نفس الجدول بأكمله',
'Borders': 'الحدود',
'None': 'اي',
'Single': 'واحد',
'Dot': 'نقطه',
'DashSmallGap': 'دashed مجاب',
'DashLargeGap': 'الاتحاد الخاص',
'DashDot': 'دashed دوت',
'DashDotDot': 'دashed دوت دوت',
'Double': 'انقر نقرا مزدوجا',
'Triple': 'الثلاثي',
'ThinThickSmallGap': 'فجوة صغيره سميكه رقيق',
'ThickThinSmallGap': 'الفجوة الصغيره رقيقه سميكه',
'ThinThickThinSmallGap': 'فجوة صغيره رقيقه الفجوة الصغيره',
'ThinThickMediumGap': 'فجوة متوسطه سميك',
'ThickThinMediumGap': 'سميكه الفجوة متوسطه رقيقه',
'ThinThickThinMediumGap': 'رقيقه سميكه متوسطه الفجوة',
'ThinThinLargeGap': 'الفجوة الكبيره رقيقه سميكه',
'ThickThinLargeGap': 'فجوة كبيره رقيقه سميك',
'ThinThickThinLargeGap': 'رقيقه سميكه الفجوة الكبيره',
'SingleWavy': 'واحد مائج',
'DoubleWavy': 'مزدوج مائج',
'DashDotStroked': 'اندفاعه نقطه القوية',
'Emboss3D': 'مزخرفD3',
'Engrave3D': 'نقشD3',
'Outset': 'البدايه',
'Inset': 'الداخلي',

```

```

'Thick': 'سميكة',
'Style': 'نمط',
'Width': 'عرض',
'Height': 'ارتفاع',
'Letter': 'رساله',
'Tabloid': 'التابلويد',
'Legal': 'القانونيه',
'Statement': 'بيان',
'Executive': 'التنفيذي',
'A3': 'A3',
'A4': 'A4',
'A5': 'A5',
'B4': 'B4',
'B5': 'B5',
'Custom Size': 'حجم مخصص',
'Different odd and even': 'مختلفه غريبه وحتى',
'Different first page': 'الصفحة الاولى مختلفه',
'From edge': 'من الحافة',
'Header': 'راس',
'Footer': 'تذييل الصفحه',
'Margin': 'الهوامش',
'Paper': 'الورق',
'Layout': 'تخطيط',
'Orientation': 'التوجه',
'Landscape': 'المناظر الطبيعيه',
'Portrait': 'صوره',
'Table Of Contents': 'جدول المحتويات',
'Show page numbers': 'إظهار أرقام الصفحات',
'Right align page numbers': 'محاذاة أرقام الصفحات إلى اليمين',
'Nothing': 'شيء',
'Tab leader': 'قائد علامة التبويب',
'Show levels': 'إظهار المستويات',
'Use hyperlinks instead of page numbers': 'استخدام الارتباطات',
الصفحات 'التشعبية بدلا من أرقام',
'Build table of contents from': 'بناء جدول محتويات من',
'Styles': 'انماط',
'Available styles': 'الأنماط المتوفرة',
'TOC level': 'مستوي جدول المحتويات',
'Heading': 'عنوان',
'Heading 1': 'عنوان 1',
'Heading 2': 'عنوان 2',
'Heading 3': 'عنوان 3',
'Heading 4': 'عنوان 4',
'Heading 5': 'عنوان 5',
'Heading 6': 'عنوان 6',
'List Paragraph': 'فقرة القائمة',
'Normal': 'العادي',
'Outline levels': 'مستويات المخطط التفصيلي',
'Table entry fields': 'حقول إدخال الجدول',
'Modify': 'تعديل',
'Color': 'لون',
'Setting': 'اعداد',
'Box': 'مربع',
'All': 'جميع',
'Custom': 'المخصصه',
'Preview': 'معاينه',
'Shading': 'التظليل',

```

```

'Fill': 'ملء',
'Apply To': 'تنطبق علي',
'Table Properties': 'خصائص الجدول',
'Cell Options': 'خيارات الخلية',
'Table Options': 'خيارات الجدول',
'Insert Table': 'ادراج جدول',
'Number of columns': 'عدد الاعمده',
'Number of rows': 'عدد الصفوف',
'Text to display': 'النص الذي سيتم عرضه',
'Address': 'عنوان',
'Insert Hyperlink': 'ادراج ارتباط تشعبي',
'Edit Hyperlink': 'تحرير ارتباط تشعبي',
'Insert': 'ادراج',
'General': 'العامه',
'Indentation': 'المسافه البادئه',
'Before text': 'قبل النص',
'Special': 'الخاصه',
'First line': 'السطر الأول',
'Hanging': 'معلقه',
'After text': 'بعد النص',
'By': 'من',
'Before': 'قبل',
'Line Spacing': 'تباعد الأسطر',
'After': 'بعد',
'At': 'في',
'Multiple': 'متعدده',
'Spacing': 'تباعد',
'Define new Multilevel list': 'تحديد قائمه متعددة الاصعده جديده',
'List level': 'مستوي القائمة',
'Choose level to modify': 'اختر المستوي الذي تريد تعديله',
'Level': 'مستوي',
'Number format': 'تنسيق الأرقام',
'Number style for this level': 'نمط الرقم لهذا المستوي',
'Enter formatting for number': 'إدخال تنسيق لرقم',
'Start at': 'بداية من',
'Restart list after': 'أعاده تشغيل قائمه بعد',
'Position': 'موقف',
'Text indent at': 'المسافة البادئة للنص في',
'Aligned at': 'محاذاة في',
'Follow number with': 'اتبع الرقم مع',
'Tab character': 'حرف علامة التبويب',
'Space': 'الفضاء',
'Arabic': 'العربية',
'UpRoman': 'حتى الروماني',
'LowRoman': 'الرومانية منخفضه',
'UpLetter': '',
'LowLetter': '',
'Number': 'عدد',
'Leading zero': 'يؤدي صفر',
'Bullet': 'رمضاه',
'Ordinal': 'الترتيبيه',
'Ordinal Text': 'النص الترتيبي',
'For East': 'للشرق',
'No Restart': 'لا أعاده تشغيل',
'Font': 'الخط',
'Font style': 'نمط الخط',
'Underline style': 'نمط التسطير',

```

```

'Font color': 'لون الخط',
'Effects': 'الاثار',
'Strikethrough': 'يتوسطه',
'Superscript': 'مرتفع',
'Subscript': 'منخفض',
'Double strikethrough': 'خط مزدوج يتوسطه خط',
'Regular': 'العادية',
'Bold': 'جريئه',
'Italic': 'مائل',
'Cut': 'قطع',
'Copy': 'نسخ',
'Paste': 'لصق',
'Hyperlink': 'الارتباط التشعبي',
'Open Hyperlink': 'فتح ارتباط تشعبي',
'Copy Hyperlink': 'نسخ ارتباط تشعبي',
'Remove Hyperlink': 'أزاله ارتباط تشعبي',
'Paragraph': 'الفقره',
'Linked(Paragraph and Character)': 'مرتبط (فقره وحرف)',
'Character': 'حرف',
'Merge Cells': 'دمج الخلايا',
'Insert Above': 'ادراج أعلاه',
'Insert Below': 'ادراج أدناه',
'Insert Left': 'ادراج إلى اليسار',
'Insert Right': 'ادراج اليمين',
>Delete': 'حذف',
>Delete Table': 'حذف جدول',
>Delete Row': 'حذف صف',
>Delete Column': 'حذف عمود',
'File Name': 'اسم الملف',
'Format Type': 'نوع التنسيق',
'Save': 'حفظ',
'Navigation': 'التنقل',
'Results': 'نتائج',
'Replace': 'استبدال',
'Replace All': 'استبدال الكل',
'We replaced all': 'استبدلنا جميع',
'Find': 'العثور',
'No matches': 'لا توجد تطابقات',
'All Done': 'كل القيام به',
'Result': 'نتيجه',
'of': 'من',
'instances': 'الحالات',
'with': 'مع',
'Click to follow link': 'انقر لمتابعه الارتباط',
'Continue Numbering': 'متابعه الترقيم',
'Bookmark name': 'اسم الإشارة المرجعية',
'Close': 'اغلق',
'Restart At': 'أعاده التشغيل عند',
'Properties': 'خصائص',
'Name': 'اسم',
'Style type': 'نوع النمط',
'Style based on': 'نمط استنادا إلى',
'Style for following paragraph': 'نمط للفقره التالية',
'Formatting': 'التنسيق',
'Numbering and Bullets': 'الترقيم والتعداد النقطي',
'Numbering': 'ترقيم',
'Update Field': 'تحديث الحقل',

```

```
'Edit Field': 'تحرير الحقل',
'Bookmark': 'الإشارة المرجعية',
'Page Setup': 'اعداد الصفحة',
'No bookmarks found': 'لم يتم العثور علي إشارات مرجعيه',
'Number format tooltip information': 'تنسيق رقم أحادي المستوي:' +
'</br>' + ']' + '%[مستوي الاعداد] [لاحقه]' + '</br>'
// tslint:disable-next-line:max-line-length
+ 'علي سبيل المثال ، "الفصل %1". سيتم عرض التقييم مثل' +
'</br>' + 'الفصل الثاني- البند' + '</br>...' + 'الفصل الأول- البند' + '</br>'
// tslint:disable-next-line:max-line-length
+ '%[بادئه]' + '</br>' + 'تنسيق رقم متعدد الإعدادات' + '</br>' + '[المستوي]' + '</br>' + ']' + '%[لاحقه]'
+ 'علي سبيل المثال ، "%1.%2". سيتم عرض التقييم مثل' +
'</br>' + 'البند 1.1' + '</br>' + 'البند 1.2' + '</br>...' + '</br>' + '1. نون-البند',
'Format': 'تنسيق',
>Create New Style': 'إنشاء نمط جديد',
'Modify Style': 'تعديل النمط',
'New': 'الجديد',
'Bullets': 'الرصاص',
'Use bookmarks': 'استخدام الإشارات المرجعية',
'Table of Contents': 'جدول المحتويات',
'AutoFit': 'الاحتواء',
'AutoFit to Contents': 'احتواء تلقائي للمحتويات',
'AutoFit to Window': 'احتواء تلقائي للإطار',
'Fixed Column Width': 'عرض العمود الثابت',
'Reset': 'اعاده تعيين',
'Match case': 'حاله الميارة',
'Whole words': 'كلمات كامل',
>Add': 'اضافه',
'Go To': 'الانتقال إلى',
'Search for': 'البحث عن',
'Replace with': 'استبدال',
'TOC 1': 'جدول المحتويات 1',
'TOC 2': 'جدول المحتويات 2',
'TOC 3': 'جدول المحتويات 3',
'TOC 4': 'جدول المحتويات 4',
'TOC 5': 'جدول المحتويات 5',
'TOC 6': 'جدول المحتويات 6',
'TOC 7': 'جدول المحتويات 7',
'TOC 8': 'جدول المحتويات 8',
'TOC 9': 'جدول المحتويات 9',
'Right-to-left': 'من اليمين إلى اليسار',
'Left-to-right': 'من اليسار إلى اليمين',
'Direction': 'الاتجاه',
'Table direction': 'اتجاه الجدول',
'Indent from right': 'مسافة بادئه من اليمين',
'Page': 'صفحه',
'Fit one page': 'احتواء صفحه واحد',
'Fit page width': 'احتواء عرض الصفحة',
// tslint:disable-next-line:max-line-length
'The current page number in the document. Click or tap to navigate specific page.': 'رقم الصفحة الحالية في المستند. انقر أو اضغط للتنقل في صفحه معينه',
}
```

```

    'colorpicker': {
      'Apply': 'تطبيق',
      'Cancel': 'إلغاء الأمر',
      'ModeSwitcher': 'مفتاح كهربائي الوضع'
    }
  }
});
export default {
  data: function() {
    return {
    };
  },
  provide: {
    DocumentEditor: [Print, SfdtExport, WordExport, TextExport,
Selection, Search, Editor, ImageResizer, EditorHistory, ContextMenu,
OptionsPane, HyperlinkDialog, TableDialog, BookmarkDialog,
TableOfContentsDialog, PageSetupDialog, StyleDialog, ListDialog,
ParagraphDialog, BulletsAndNumberingDialog, FontDialog,
TablePropertiesDialog, BordersAndShadingDialog, TableOptionsDialog,
CellOptionsDialog, StylesDialog]
  },
  mounted() {
    this.$refs.documenteditor.ej2Instances.locale = 'ar-AE';
    this.$refs.documenteditor.ej2Instances.isReadOnly = false;
    let sfdt: string = `{
      "sections": [
        {
          "blocks": [
            {
              "characterFormat": {
                "fontSize": 18.0,
                "fontFamily": "Calibri",
                "fontFamilyBidi": "Calibri"
              },
              "paragraphFormat": {
                "beforeSpacing": 18.0,
                "afterSpacing": 30.0,
                "styleName": "Heading 1",
                "bidi": true
              },
              "inlines": [
                {
                  "name": "_GoBack",
                  "bookmarkType": 0
                },
                {
                  "name": "_GoBack",
                  "bookmarkType": 1
                },
                {
                  "text": "اعمال المغامرة دورات",
                  "characterFormat": {
                    "fontSize": 18.0,
                    "bidi": true,
                    "fontSizeBidi": 18.0
                  }
                }
              ]
            }
          ]
        }
      ]
    }
  }
}

```

```

    ]
    },
    ],
    "headersFooters": {},
    "sectionFormat": {
      "headerDistance": 36.0,
      "footerDistance": 36.0,
      "pageWidth": 612.0,
      "pageHeight": 792.0,
      "leftMargin": 72.0,
      "rightMargin": 72.0,
      "topMargin": 72.0,
      "bottomMargin": 72.0,
      "differentFirstPage": false,
      "differentOddAndEvenPages": false,
      "bidi": false
    }
  },
  ],
  "styles": []
} `;
this.$refs.documenteditor.open(sfdd);
}
});
</script>
<style>
  @import "../node_modules/@syncfusion/ej2-vue-
documenteditor/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/document-editor/right-to-left-cs1" %}

### Chart in Vue Document editor component

Document Editor provides chart preservation support. Using Document Editor, you can see the chart reports from your Word document.

The following example shows chart preservation in Document Editor.

#### APP.VUE

```

<template>
  <div id="app">
    <ejs-documenteditor ref="documenteditor" id="container_1"
height="600px" ></ejs-documenteditor>
  </div>
</template>
<script>
  import Vue from 'vue'
  import { DocumentEditorPlugin } from '@syncfusion/ej2-vue-
documenteditor';
  Vue.use(DocumentEditorPlugin);
  export default {
    data: function() {
      return {
      };
    },
  },

```

```

    mounted: function() {
      let sfdt: string =
`{"sections":[{"sectionFormat":{"pageWidth":612,"pageHeight":792,"leftMargin":72,"rightMargin":72,"topMargin":72,"bottomMargin":72,"differentFirstPage":false,"differentOddAndEvenPages":false,"headerDistance":36,"footerDistance":36,"bidi":false},"blocks":[{"paragraphFormat":{"textAlignment":"Center","afterSpacing":0,"lineSpacing":1,"lineSpacingType":"Multiple","styleName":"Normal","listFormat":{},"characterFormat":{"bold":true,"fontSize":12,"fontFamily":"Verdana","fontSizeBidi":12,"fontFamilyBidi":"Verdana"},"inlines":[{"characterFormat":{"bold":true,"fontSize":14,"fontFamily":"Verdana","fontColor":"#17365DFF","styleName":"a","fontSizeBidi":14,"fontFamilyBidi":"Verdana"},"text":"Northwind Management Report"}]}],{"paragraphFormat":{"afterSpacing":0,"lineSpacing":1,"lineSpacingType":"Multiple","styleName":"Normal","listFormat":{},"characterFormat":{"fontSize":10,"fontFamily":"Verdana","fontSizeBidi":10,"fontFamilyBidi":"Verdana"},"inlines":[]},{"paragraphFormat":{"afterSpacing":0,"styleName":"Normal","listFormat":{},"characterFormat":{},"inlines":[{"characterFormat":{"fontSize":10,"fontFamily":"Verdana","styleName":"a","fontSizeBidi":10,"fontFamilyBidi":"Verdana"},"text":"This management report provides information obtained through data analysis, regarding the "},{ "characterFormat":{"fontSize":10,"fontFamily":"Verdana","styleName":"a","fontSizeBidi":10,"fontFamilyBidi":"Verdana"},"text":"performance of Northwind Traders. This report will pay particular"}],{"characterFormat":{"fontSize":10,"fontFamily":"Verdana","styleName":"a","fontSizeBidi":10,"fontFamilyBidi":"Verdana"},"text":""}, {"characterFormat":{"fontSize":10,"fontFamily":"Verdana","styleName":"a","fontSizeBidi":10,"fontFamilyBidi":"Verdana"},"text":" attention to the "}, {"characterFormat":{"fontSize":10,"fontFamily":"Verdana","styleName":"a","fontSizeBidi":10,"fontFamilyBidi":"Verdana"},"text":"best-selling products, of our company."}, {"characterFormat":{"fontSize":10,"fontFamily":"Verdana","fontSizeBidi":10,"fontFamilyBidi":"Times New Roman"},"text":"The best-selling products of Northwind Traders"}, {"characterFormat":{"fontSize":10,"fontFamily":"Verdana","fontSizeBidi":10,"fontFamilyBidi":"Times New Roman"},"text":"Company as follows:"}]}],{"paragraphFormat":{"afterSpacing":0,"styleName":"Normal","listFormat":{},"characterFormat":{},"inlines":[]}, {"rows":[{"cells":[{"blocks":[{"paragraphFormat":{"rightIndent":26.850000381469727,"styleName":"Normal","listFormat":{},"characterFormat":{},"inlines":[{"characterFormat":{"fontSize":10,"fontFamily":"Verdana","fontSizeBidi":10,"fontFamilyBidi":"Times New Roman"},"text":"S.No"}]}], "cellFormat":{"borders":{"top":{"color":"#4472C4FF","hasNoneStyle":false,"lineStyle":"Single","lineWidth":0.5,"shadow":false,"space":0},"left":{"color":"#4472C4FF","hasNoneStyle":false,"lineStyle":"Single","lineWidth":0.5,"shadow":false,"space":0},"right":{"color":"#8EADBF","hasNoneStyle":false,"lineStyle":"Single","lineWidth":0.5,"shadow":false,"space":0},"bottom":{"color":"#4472C4FF","hasNoneStyle":false,"lineStyle":"Single","lineWidth":0.5,"shadow":false,"space":0},"diagonalDown":{"color":"#000000","hasNoneStyle":false,"lineStyle":"None","lineWidth":0,"shadow":false,"space":0},"diagonalUp":{"color":"#000000","hasNoneStyle":false,"lineStyle":"None","lineWidth":0,"shadow":false,"space":0},"horizontal":{"color":"#000000","hasNoneStyle":false,"lineStyle":"None","lineWidth":0,"shadow":false,"space":0},"vertical":{"color":"#000000","hasNoneStyle":false,"lineStyle":"None","lineWidth":0,"shadow":false,"space":0}},"shading":{"backgroundColor":"#4472C4FF","foregroundColor":"empty","textureStyle":"TextureNone"},"preferredWidth":13.420000076293945,"preferredWidthType":"Percent","cellWidth":64.71214527422465,"columnSpan":1,"rowSpan":1,"verticalAlignment":"Top"},"columnIndex":0},{ "blocks":[{"paragraphFormat":{"styleName":"Normal","listFormat":{},"characterFormat":{"fontSize":10,"fontFamily":"Verdana","fontSizeBidi":10,"fontFamilyBidi":"Times New Roman"},"text":"S.No"}]}], "cellFormat":{"borders":{"top":{"color":"#4472C4FF","hasNoneStyle":false,"lineStyle":"Single","lineWidth":0.5,"shadow":false,"space":0},"left":{"color":"#4472C4FF","hasNoneStyle":false,"lineStyle":"Single","lineWidth":0.5,"shadow":false,"space":0},"right":{"color":"#8EADBF","hasNoneStyle":false,"lineStyle":"Single","lineWidth":0.5,"shadow":false,"space":0},"bottom":{"color":"#4472C4FF","hasNoneStyle":false,"lineStyle":"Single","lineWidth":0.5,"shadow":false,"space":0},"diagonalDown":{"color":"#000000","hasNoneStyle":false,"lineStyle":"None","lineWidth":0,"shadow":false,"space":0},"diagonalUp":{"color":"#000000","hasNoneStyle":false,"lineStyle":"None","lineWidth":0,"shadow":false,"space":0},"horizontal":{"color":"#000000","hasNoneStyle":false,"lineStyle":"None","lineWidth":0,"shadow":false,"space":0},"vertical":{"color":"#000000","hasNoneStyle":false,"lineStyle":"None","lineWidth":0,"shadow":false,"space":0}},"shading":{"backgroundColor":"#4472C4FF","foregroundColor":"empty","textureStyle":"TextureNone"},"preferredWidth":13.420000076293945,"preferredWidthType":"Percent","cellWidth":64.71214527422465,"columnSpan":1,"rowSpan":1,"verticalAlignment":"Top"},"columnIndex":0},{ "blocks":[{"paragraphFormat":{"styleName":"Normal","listFormat":{},"characterFormat":{"fontSize":10,"fontFamily":"Verdana","fontSizeBidi":10,"fontFamilyBidi":"Times New Roman"},"text":"S.No"}]}], "cellFormat":{"borders":{"top":{"color":"#4472C4FF","hasNoneStyle":false,"lineStyle":"Single","lineWidth":0.5,"shadow":false,"space":0},"left":{"color":"#4472C4FF","hasNoneStyle":false,"lineStyle":"Single","lineWidth":0.5,"shadow":false,"space":0},"right":{"color":"#8EADBF","hasNoneStyle":false,"lineStyle":"Single","lineWidth":0.5,"shadow":false,"space":0},"bottom":{"color":"#4472C4FF","hasNoneStyle":false,"lineStyle":"Single","lineWidth":0.5,"shadow":false,"space":0},"diagonalDown":{"color":"#000000","hasNoneStyle":false,"lineStyle":"None","lineWidth":0,"shadow":false,"space":0},"diagonalUp":{"color":"#000000","hasNoneStyle":false,"lineStyle":"None","lineWidth":0,"shadow":false,"space":0},"horizontal":{"color":"#000000","hasNoneStyle":false,"lineStyle":"None","lineWidth":0,"shadow":false,"space":0},"vertical":{"color":"#000000","hasNoneStyle":false,"lineStyle":"None","lineWidth":0,"shadow":false,"space":0}},"shading":{"backgroundColor":"#4472C4FF","foregroundColor":"empty","textureStyle":"TextureNone"},"preferredWidth":13.420000076293945,"preferredWidthType":"Percent","cellWidth":64.71214527422465,"columnSpan":1,"rowSpan":1,"verticalAlignment":"Top"},"columnIndex":0}]}], "tableFormat":{"width":612,"height":792,"borderStyle":"Single","borderWidth":1,"shading":{"backgroundColor":"#4472C4FF","foregroundColor":"empty","textureStyle":"TextureNone"},"preferredWidth":13.420000076293945,"preferredWidthType":"Percent","cellWidth":64.71214527422465,"columnSpan":1,"rowSpan":1,"verticalAlignment":"Top"},"columnIndex":0}]}]}`

```



```

erFormat": {}, "inlines": [{"characterFormat": {"fontSize": 10, "fontFamily": "Verdana", "fontSizeBidi": 10, "fontFamilyBidi": "Times New Roman"}, "text": "Product Name"}], "cellFormat": {"borders": {"top": {"color": "#4472C4FF", "hasNoneStyle": false, "lineStyle": "Single", "lineWidth": 0.5, "shadow": false, "space": 0}, "left": {"color": "#8EAADBFF", "hasNoneStyle": false, "lineStyle": "Single", "lineWidth": 0.5, "shadow": false, "space": 0}, "right": {"color": "#8EAADBFF", "hasNoneStyle": false, "lineStyle": "Single", "lineWidth": 0.5, "shadow": false, "space": 0}, "bottom": {"color": "#4472C4FF", "hasNoneStyle": false, "lineStyle": "Single", "lineWidth": 0.5, "shadow": false, "space": 0}, "diagonalDown": {"color": "#000000", "hasNoneStyle": false, "lineStyle": "None", "lineWidth": 0, "shadow": false, "space": 0}, "diagonalUp": {"color": "#000000", "hasNoneStyle": false, "lineStyle": "None", "lineWidth": 0, "shadow": false, "space": 0}, "horizontal": {"color": "#000000", "hasNoneStyle": false, "lineStyle": "None", "lineWidth": 0, "shadow": false, "space": 0}, "vertical": {"color": "#000000", "hasNoneStyle": false, "lineStyle": "None", "lineWidth": 0, "shadow": false, "space": 0}}, "shading": {"backgroundColor": "#4472C4FF", "foregroundColor": "empty", "textureStyle": "TextureNone"}, "preferredWidth": 48.86000061035156, "preferredWidthType": "Percent", "cellWidth": 292.87942351880633, "columnSpan": 1, "rowSpan": 1, "verticalAlignment": "Top"}, {"blocks": [{"paragraphFormat": {"styleName": "Normal", "listFormat": {}, "characterFormat": {}, "inlines": [{"characterFormat": {"fontSize": 10, "fontFamily": "Verdana", "fontSizeBidi": 10, "fontFamilyBidi": "Times New Roman"}, "text": "Sum of Sales(in $)"}], "cellFormat": {"borders": {"top": {"color": "#4472C4FF", "hasNoneStyle": false, "lineStyle": "Single", "lineWidth": 0.5, "shadow": false, "space": 0}, "left": {"color": "#8EAADBFF", "hasNoneStyle": false, "lineStyle": "Single", "lineWidth": 0.5, "shadow": false, "space": 0}, "right": {"color": "#4472C4FF", "hasNoneStyle": false, "lineStyle": "Single", "lineWidth": 0.5, "shadow": false, "space": 0}, "bottom": {"color": "#4472C4FF", "hasNoneStyle": false, "lineStyle": "Single", "lineWidth": 0.5, "shadow": false, "space": 0}, "diagonalDown": {"color": "#000000", "hasNoneStyle": false, "lineStyle": "None", "lineWidth": 0, "shadow": false, "space": 0}, "diagonalUp": {"color": "#000000", "hasNoneStyle": false, "lineStyle": "None", "lineWidth": 0, "shadow": false, "space": 0}, "horizontal": {"color": "#000000", "hasNoneStyle": false, "lineStyle": "None", "lineWidth": 0, "shadow": false, "space": 0}, "vertical": {"color": "#000000", "hasNoneStyle": false, "lineStyle": "None", "lineWidth": 0, "shadow": false, "space": 0}}, "shading": {"backgroundColor": "#4472C4FF", "foregroundColor": "empty", "textureStyle": "TextureNone"}, "preferredWidth": 37.720001220703125, "preferredWidthType": "Percent", "cellWidth": 117.95841899993776, "columnSpan": 1, "rowSpan": 1, "verticalAlignment": "Top"}, {"height": 14.399999618530273, "allowBreakAcrossPages": true, "heightType": "Exactly", "isHeader": false, "borders": {"top": {"color": "#8EAADBFF", "hasNoneStyle": false, "lineStyle": "Single", "lineWidth": 0.5, "shadow": false, "space": 0}, "left": {"color": "#8EAADBFF", "hasNoneStyle": false, "lineStyle": "Single", "lineWidth": 0.5, "shadow": false, "space": 0}, "right": {"color": "#8EAADBFF", "hasNoneStyle": false, "lineStyle": "Single", "lineWidth": 0.5, "shadow": false, "space": 0}, "bottom": {"color": "#8EAADBFF", "hasNoneStyle": false, "lineStyle": "Single", "lineWidth": 0.5, "shadow": false, "space": 0}, "diagonalDown": {"color": "#000000", "hasNoneStyle": false, "lineStyle": "None", "lineWidth": 0, "shadow": false, "space": 0}, "diagonalUp": {"color": "#000000", "hasNoneStyle": false, "lineStyle": "None", "lineWidth": 0, "shadow": false, "space": 0}, "horizontal": {"color": "#8EAADBFF", "hasNoneStyle": false, "lineStyle": "Single", "lineWidth": 0.5, "shadow": false, "space": 0}, "vertical": {"color": "#8EAADBFF", "hasNoneStyle": false, "lineStyle": "Single", "lineWidth": 0.5, "shadow": false, "space": 0}}, "gridBefore": 0, "gridBeforeWidth": 0, "gridBeforeWidthType": "Point", "gridAfter": 0, "gridAfterWidth": 0, "gridAfterWidthType": "Point"}, {"cells": [{"blocks": [{"paragraphFormat": {"styleName": "Normal", "listFormat": {}, "characterFormat": {}, "inlines": [{"characterFormat": {"fontSize": 10, "fontFamily": "Verdana", "fontSizeBidi": 10, "fontFamilyBidi": "Times New Roman"}, "text": "1"}], "cellFormat": {"borders": {"top": {"color": "#8EAADBFF", "hasNoneStyle": false, "lineStyle": "Single", "lineWidth": 0.5, "shadow": false, "spa

```

```

ce":0}, "left": {"color": "#8EADBFF", "hasNoneStyle": false, "lineStyle": "Single",
"lineWidth": 0.5, "shadow": false, "space": 0}, "right": {"color": "#8EADBFF", "has
NoneStyle": false, "lineStyle": "Single", "lineWidth": 0.5, "shadow": false, "space"
:0}, "bottom": {"color": "#8EADBFF", "hasNoneStyle": false, "lineStyle": "Single",
"lineWidth": 0.5, "shadow": false, "space": 0}, "diagonalDown": {"color": "#000000",
"hasNoneStyle": false, "lineStyle": "None", "lineWidth": 0, "shadow": false, "space"
:0}, "diagonalUp": {"color": "#000000", "hasNoneStyle": false, "lineStyle": "None",
"lineWidth": 0, "shadow": false, "space": 0}, "horizontal": {"color": "#000000", "has
NoneStyle": false, "lineStyle": "None", "lineWidth": 0, "shadow": false, "space": 0},
"vertical": {"color": "#000000", "hasNoneStyle": false, "lineStyle": "None", "lineW
idth": 0, "shadow": false, "space": 0}}, "shading": {"backgroundColor": "#D9E2F3FF",
"foregroundColor": "empty", "textureStyle": "TextureNone"}, "preferredWidth": 13.
420000076293945, "preferredWidthType": "Percent", "cellWidth": 64.71214527422465
, "columnSpan": 1, "rowSpan": 1, "verticalAlignment": "Top", "columnIndex": 0}, {""bl
ocks": [{"paragraphFormat": {"styleName": "Normal", "listFormat": {}}, "characterF
ormat": {}, "inlines": [{"characterFormat": {"fontSize": 10, "fontFamily": "Verdana
", "fontSizeBidi": 10, "fontFamilyBidi": "Times New Roman"}, "text": "Côte de
Blaye"}]}], "cellFormat": {"borders": {"top": {"color": "#8EADBFF", "hasNoneStyle
": false, "lineStyle": "Single", "lineWidth": 0.5, "shadow": false, "space": 0}, "left
": {"color": "#8EADBFF", "hasNoneStyle": false, "lineStyle": "Single", "lineWidth"
: 0.5, "shadow": false, "space": 0}, "right": {"color": "#8EADBFF", "hasNoneStyle": f
alse, "lineStyle": "Single", "lineWidth": 0.5, "shadow": false, "space": 0}, "bottom"
: {"color": "#8EADBFF", "hasNoneStyle": false, "lineStyle": "Single", "lineWidth":
0.5, "shadow": false, "space": 0}, "diagonalDown": {"color": "#000000", "hasNoneStyl
e": false, "lineStyle": "None", "lineWidth": 0, "shadow": false, "space": 0}, "diagona
lUp": {"color": "#000000", "hasNoneStyle": false, "lineStyle": "None", "lineWidth":
0, "shadow": false, "space": 0}, "horizontal": {"color": "#000000", "hasNoneStyle": f
alse, "lineStyle": "None", "lineWidth": 0, "shadow": false, "space": 0}, "vertical": {
"color": "#000000", "hasNoneStyle": false, "lineStyle": "None", "lineWidth": 0, "sha
dow": false, "space": 0}}, "shading": {"backgroundColor": "#D9E2F3FF", "foregroundC
olor": "empty", "textureStyle": "TextureNone"}, "preferredWidth": 48.860000610351
56, "preferredWidthType": "Percent", "cellWidth": 292.87942351880633, "columnSpan
": 1, "rowSpan": 1, "verticalAlignment": "Top", "columnIndex": 1}, {""blocks": [{"par
agraphFormat": {"styleName": "Normal", "listFormat": {}}, "characterFormat": {}, "i
nlines": [{"characterFormat": {"fontSize": 10, "fontFamily": "Verdana", "fontSizeB
idi": 10, "fontFamilyBidi": "Times New
Roman"}, "text": "141.396"}]}], "cellFormat": {"borders": {"top": {"color": "#8EAD
BFF", "hasNoneStyle": false, "lineStyle": "Single", "lineWidth": 0.5, "shadow": fals
e, "space": 0}, "left": {"color": "#8EADBFF", "hasNoneStyle": false, "lineStyle": "S
ingle", "lineWidth": 0.5, "shadow": false, "space": 0}, "right": {"color": "#8EADBFF
", "hasNoneStyle": false, "lineStyle": "Single", "lineWidth": 0.5, "shadow": false, "
space": 0}, "bottom": {"color": "#8EADBFF", "hasNoneStyle": false, "lineStyle": "Si
ngle", "lineWidth": 0.5, "shadow": false, "space": 0}, "diagonalDown": {"color": "#00
0000", "hasNoneStyle": false, "lineStyle": "None", "lineWidth": 0, "shadow": false, "
space": 0}, "diagonalUp": {"color": "#000000", "hasNoneStyle": false, "lineStyle": "
None", "lineWidth": 0, "shadow": false, "space": 0}, "horizontal": {"color": "#000000
", "hasNoneStyle": false, "lineStyle": "None", "lineWidth": 0, "shadow": false, "spac
e": 0}, "vertical": {"color": "#000000", "hasNoneStyle": false, "lineStyle": "None",
"lineWidth": 0, "shadow": false, "space": 0}}, "shading": {"backgroundColor": "#D9E2
F3FF", "foregroundColor": "empty", "textureStyle": "TextureNone"}, "preferredWidt
h": 37.720001220703125, "preferredWidthType": "Percent", "cellWidth": 117.9584189
9993776, "columnSpan": 1, "rowSpan": 1, "verticalAlignment": "Top", "columnIndex":
2}], "rowFormat": {"height": 14.3999999618530273, "allowBreakAcrossPages": true, "h
eightType": "Exactly", "isHeader": false, "borders": {"top": {"color": "#8EADBFF",
"hasNoneStyle": false, "lineStyle": "Single", "lineWidth": 0.5, "shadow": false, "sp
ace": 0}, "left": {"color": "#8EADBFF", "hasNoneStyle": false, "lineStyle": "Single
", "lineWidth": 0.5, "shadow": false, "space": 0}, "right": {"color": "#8EADBFF", "ha

```

```

sNoneStyle":false,"lineStyle":"Single","lineWidth":0.5,"shadow":false,"space
":0},"bottom":{"color":"#8EADBFF","hasNoneStyle":false,"lineStyle":"Single"
,"lineWidth":0.5,"shadow":false,"space":0},"diagonalDown":{"color":"#000000"
,"hasNoneStyle":false,"lineStyle":"None","lineWidth":0,"shadow":false,"space
":0},"diagonalUp":{"color":"#000000","hasNoneStyle":false,"lineStyle":"None"
,"lineWidth":0,"shadow":false,"space":0},"horizontal":{"color":"#8EADBFF","
hasNoneStyle":false,"lineStyle":"Single","lineWidth":0.5,"shadow":false,"spa
ce":0},"vertical":{"color":"#8EADBFF","hasNoneStyle":false,"lineStyle":"Sin
gle","lineWidth":0.5,"shadow":false,"space":0}},"gridBefore":0,"gridBeforeWi
dth":0,"gridBeforeWidthType":"Point","gridAfter":0,"gridAfterWidth":0,"gridA
fterWidthType":"Point"}},{ "cells": [{ "blocks": [{ "paragraphFormat": { "styleName
": "Normal", "listFormat": {}, "characterFormat": {}, "inlines": [{ "characterForma
t": { "fontSize": 10, "fontFamily": "Verdana", "fontSizeBidi": 10, "fontFamilyBidi":
"Times New
Roman"}, "text": "2"} ] } ] } ], "cellFormat": { "borders": { "top": { "color": "#8EADBFF", "
hasNoneStyle": false, "lineStyle": "Single", "lineWidth": 0.5, "shadow": false, "spa
ce": 0 }, "left": { "color": "#8EADBFF", "hasNoneStyle": false, "lineStyle": "Single"
, "lineWidth": 0.5, "shadow": false, "space": 0 }, "right": { "color": "#8EADBFF", "has
NoneStyle": false, "lineStyle": "Single", "lineWidth": 0.5, "shadow": false, "space"
: 0 }, "bottom": { "color": "#8EADBFF", "hasNoneStyle": false, "lineStyle": "Single",
"lineWidth": 0.5, "shadow": false, "space": 0 }, "diagonalDown": { "color": "#000000",
"hasNoneStyle": false, "lineStyle": "None", "lineWidth": 0, "shadow": false, "space"
: 0 }, "diagonalUp": { "color": "#000000", "hasNoneStyle": false, "lineStyle": "None",
"lineWidth": 0, "shadow": false, "space": 0 }, "horizontal": { "color": "#000000", "has
NoneStyle": false, "lineStyle": "None", "lineWidth": 0, "shadow": false, "space": 0 },
"vertical": { "color": "#000000", "hasNoneStyle": false, "lineStyle": "None", "lineW
idth": 0, "shadow": false, "space": 0 } }, "shading": { "backgroundColor": "#FFFFFFF
F", "foregroundColor": "empty", "textureStyle": "TextureNone" }, "preferredWidth": 13.
420000076293945, "preferredWidthType": "Percent", "cellWidth": 64.71214527422465
, "columnSpan": 1, "rowSpan": 1, "verticalAlignment": "Top", "columnIndex": 0 }, { "bl
ocks": [{ "paragraphFormat": { "styleName": "Normal", "listFormat": {}, "characterF
ormat": {}, "inlines": [{ "characterFormat": { "fontSize": 10, "fontFamily": "Verdana
", "fontSizeBidi": 10, "fontFamilyBidi": "Times New Roman"}, "text": "Thüringer
Rostbratwurst"} ] } ] } ], "cellFormat": { "borders": { "top": { "color": "#8EADBFF", "hasN
oneStyle": false, "lineStyle": "Single", "lineWidth": 0.5, "shadow": false, "space":
0 }, "left": { "color": "#8EADBFF", "hasNoneStyle": false, "lineStyle": "Single", "li
neWidth": 0.5, "shadow": false, "space": 0 }, "right": { "color": "#8EADBFF", "hasNone
Style": false, "lineStyle": "Single", "lineWidth": 0.5, "shadow": false, "space": 0 },
"bottom": { "color": "#8EADBFF", "hasNoneStyle": false, "lineStyle": "Single", "lin
eWidth": 0.5, "shadow": false, "space": 0 }, "diagonalDown": { "color": "#000000", "has
NoneStyle": false, "lineStyle": "None", "lineWidth": 0, "shadow": false, "space": 0 },
"diagonalUp": { "color": "#000000", "hasNoneStyle": false, "lineStyle": "None", "lin
eWidth": 0, "shadow": false, "space": 0 }, "horizontal": { "color": "#000000", "hasNone
Style": false, "lineStyle": "None", "lineWidth": 0, "shadow": false, "space": 0 }, "ver
tical": { "color": "#000000", "hasNoneStyle": false, "lineStyle": "None", "lineWidth
": 0, "shadow": false, "space": 0 } }, "shading": { "backgroundColor": "#FFFFFFF
F", "foregroundColor": "empty", "textureStyle": "TextureNone" }, "preferredWidth": 48.8600
0061035156, "preferredWidthType": "Percent", "cellWidth": 292.87942351880633, "co
lumnSpan": 1, "rowSpan": 1, "verticalAlignment": "Top", "columnIndex": 1 }, { "blocks
": [{ "paragraphFormat": { "styleName": "Normal", "listFormat": {}, "characterForma
t": {}, "inlines": [{ "characterFormat": { "fontSize": 10, "fontFamily": "Verdana", "f
ontSizeBidi": 10, "fontFamilyBidi": "Times New
Roman"}, "text": "80.368"} ] } ] } ], "cellFormat": { "borders": { "top": { "color": "#8EADB
FF", "hasNoneStyle": false, "lineStyle": "Single", "lineWidth": 0.5, "shadow": false
, "space": 0 }, "left": { "color": "#8EADBFF", "hasNoneStyle": false, "lineStyle": "Si
ngle", "lineWidth": 0.5, "shadow": false, "space": 0 }, "right": { "color": "#8EADBFF",
"hasNoneStyle": false, "lineStyle": "Single", "lineWidth": 0.5, "shadow": false, "s

```

```

"page":0,"bottom":{"color":"#8EAADBFF","hasNoneStyle":false,"lineStyle":"Single",
"lineWidth":0.5,"shadow":false,"space":0},"diagonalDown":{"color":"#000000",
"hasNoneStyle":false,"lineStyle":"None","lineWidth":0,"shadow":false,"space":0},
"diagonalUp":{"color":"#000000","hasNoneStyle":false,"lineStyle":"None",
"lineWidth":0,"shadow":false,"space":0},"horizontal":{"color":"#000000",
"hasNoneStyle":false,"lineStyle":"None","lineWidth":0,"shadow":false,"space":0},
"vertical":{"color":"#000000","hasNoneStyle":false,"lineStyle":"None",
"lineWidth":0,"shadow":false,"space":0},"shading":{"backgroundColor":"#FFFFFF",
"foregroundColor":"empty","textureStyle":"TextureNone"},"preferredWidth":
37.720001220703125,"preferredWidthType":"Percent","cellWidth":117.95841899
993776,"columnSpan":1,"rowSpan":1,"verticalAlignment":"Top"},"columnIndex":2
}],
"rowFormat":{"height":14.399999618530273,"allowBreakAcrossPages":true,"heightType":
"Exactly","isHeader":false,"borders":{"top":{"color":"#8EAADBFF","hasNoneStyle":
false,"lineStyle":"Single","lineWidth":0.5,"shadow":false,"space":0},
"left":{"color":"#8EAADBFF","hasNoneStyle":false,"lineStyle":"Single",
"lineWidth":0.5,"shadow":false,"space":0},
"right":{"color":"#8EAADBFF","hasNoneStyle":false,"lineStyle":"Single",
"lineWidth":0.5,"shadow":false,"space":0},
"bottom":{"color":"#8EAADBFF","hasNoneStyle":false,"lineStyle":"Single",
"lineWidth":0.5,"shadow":false,"space":0},
"diagonalDown":{"color":"#000000",
"hasNoneStyle":false,"lineStyle":"None","lineWidth":0,"shadow":false,"space":0},
"diagonalUp":{"color":"#000000","hasNoneStyle":false,"lineStyle":"None",
"lineWidth":0,"shadow":false,"space":0},
"horizontal":{"color":"#8EAADBFF","hasNoneStyle":false,"lineStyle":"Single",
"lineWidth":0.5,"shadow":false,"space":0},
"vertical":{"color":"#8EAADBFF","hasNoneStyle":false,"lineStyle":"Single",
"lineWidth":0.5,"shadow":false,"space":0}},
"gridBefore":0,"gridBeforeWidth":0,"gridAfter":0,"gridAfterWidth":0,
"gridAfterWidthType":"Point"},"cells":[{"blocks":[{"paragraphFormat":{"styleName":
"Normal","listFormat":{},"characterFormat":{"fontSize":10,"fontFamily":"Verdana",
"fontSizeBidi":10,"fontFamilyBidi":"Times New Roman"},
"text":"3"}]}]}],
"cellFormat":{"borders":{"top":{"color":"#8EAADBFF","hasNoneStyle":false,
"lineStyle":"Single","lineWidth":0.5,"shadow":false,"space":0},
"left":{"color":"#8EAADBFF","hasNoneStyle":false,"lineStyle":"Single",
"lineWidth":0.5,"shadow":false,"space":0},
"right":{"color":"#8EAADBFF","hasNoneStyle":false,"lineStyle":"Single",
"lineWidth":0.5,"shadow":false,"space":0},
"bottom":{"color":"#8EAADBFF","hasNoneStyle":false,"lineStyle":"Single",
"lineWidth":0.5,"shadow":false,"space":0},
"diagonalDown":{"color":"#000000",
"hasNoneStyle":false,"lineStyle":"None","lineWidth":0,"shadow":false,"space":0},
"diagonalUp":{"color":"#000000","hasNoneStyle":false,"lineStyle":"None",
"lineWidth":0,"shadow":false,"space":0},
"horizontal":{"color":"#000000","hasNoneStyle":false,"lineStyle":"None",
"lineWidth":0,"shadow":false,"space":0},
"vertical":{"color":"#000000","hasNoneStyle":false,"lineStyle":"None",
"lineWidth":0,"shadow":false,"space":0}},
"shading":{"backgroundColor":"#D9E2F3FF",
"foregroundColor":"empty","textureStyle":"TextureNone"},"preferredWidth":13.
420000076293945,"preferredWidthType":"Percent","cellWidth":64.71214527422465,
"columnSpan":1,"rowSpan":1,"verticalAlignment":"Top"},"columnIndex":0},
{"blocks":[{"paragraphFormat":{"styleName":"Normal","listFormat":{},"characterFormat":
{},"inlines":[{"characterFormat":{"fontSize":10,"fontFamily":"Verdana",
"fontSizeBidi":10,"fontFamilyBidi":"Times New Roman"},
"text":"Raclette Courdavault"}]}]}],
"cellFormat":{"borders":{"top":{"color":"#8EAADBFF","hasNoneStyle":false,
"lineStyle":"Single","lineWidth":0.5,"shadow":false,"space":0},
"left":{"color":"#8EAADBFF","hasNoneStyle":false,"lineStyle":"Single",
"lineWidth":0.5,"shadow":false,"space":0},
"right":{"color":"#8EAADBFF","hasNoneStyle":false,"lineStyle":"Single",
"lineWidth":0.5,"shadow":false,"space":0},
"bottom":{"color":"#8EAADBFF","hasNoneStyle":false,"lineStyle":"Single",
"lineWidth":0.5,"shadow":false,"space":0},
"diagonalDown":{"color":"#000000",
"hasNoneStyle":false,"lineStyle":"None","lineWidth":0,"shadow":false,"space":0},

```

```

diagonalUp":{"color":"#000000","hasNoneStyle":false,"lineStyle":"None","lineWidth":0,"shadow":false,"space":0},"horizontal":{"color":"#000000","hasNoneStyle":false,"lineStyle":"None","lineWidth":0,"shadow":false,"space":0},"vertical":{"color":"#000000","hasNoneStyle":false,"lineStyle":"None","lineWidth":0,"shadow":false,"space":0}},"shading":{"backgroundColor":"#D9E2F3FF","foregroundColor":"empty","textureStyle":"TextureNone"},"preferredWidth":48.86000061035156,"preferredWidthType":"Percent","cellWidth":292.87942351880633,"columnSpan":1,"rowSpan":1,"verticalAlignment":"Top"},"columnIndex":1},{
"blocks":
[{"paragraphFormat":{"styleName":"Normal","listFormat":{},"characterFormat":{},"inlines":[{"characterFormat":{"fontSize":10,"fontFamily":"Verdana","fontSizeBidi":10,"fontFamilyBidi":"Times New Roman"},"text":"71.155"}]}],"cellFormat":{"borders":{"top":{"color":"#8EAADBFF","hasNoneStyle":false,"lineStyle":"Single","lineWidth":0.5,"shadow":false,"space":0},"left":{"color":"#8EAADBFF","hasNoneStyle":false,"lineStyle":"Single","lineWidth":0.5,"shadow":false,"space":0},"right":{"color":"#8EAADBFF","hasNoneStyle":false,"lineStyle":"Single","lineWidth":0.5,"shadow":false,"space":0},"bottom":{"color":"#8EAADBFF","hasNoneStyle":false,"lineStyle":"Single","lineWidth":0.5,"shadow":false,"space":0},"diagonalDown":{"color":"#000000","hasNoneStyle":false,"lineStyle":"None","lineWidth":0,"shadow":false,"space":0},"diagonalUp":{"color":"#000000","hasNoneStyle":false,"lineStyle":"None","lineWidth":0,"shadow":false,"space":0},"horizontal":{"color":"#000000","hasNoneStyle":false,"lineStyle":"None","lineWidth":0,"shadow":false,"space":0},"vertical":{"color":"#000000","hasNoneStyle":false,"lineStyle":"None","lineWidth":0,"shadow":false,"space":0}},"shading":{"backgroundColor":"#D9E2F3FF","foregroundColor":"empty","textureStyle":"TextureNone"},"preferredWidth":37.720001220703125,"preferredWidthType":"Percent","cellWidth":117.95841899993776,"columnSpan":1,"rowSpan":1,"verticalAlignment":"Top"},"columnIndex":2}],
"rowFormat":{"height":14.399999618530273,"allowBreakAcrossPages":true,"rightType":"Exactly","isHeader":false,"borders":{"top":{"color":"#8EAADBFF","hasNoneStyle":false,"lineStyle":"Single","lineWidth":0.5,"shadow":false,"space":0},"left":{"color":"#8EAADBFF","hasNoneStyle":false,"lineStyle":"Single","lineWidth":0.5,"shadow":false,"space":0},"right":{"color":"#8EAADBFF","hasNoneStyle":false,"lineStyle":"Single","lineWidth":0.5,"shadow":false,"space":0},"bottom":{"color":"#8EAADBFF","hasNoneStyle":false,"lineStyle":"Single","lineWidth":0.5,"shadow":false,"space":0},"diagonalDown":{"color":"#000000","hasNoneStyle":false,"lineStyle":"None","lineWidth":0,"shadow":false,"space":0},"diagonalUp":{"color":"#000000","hasNoneStyle":false,"lineStyle":"None","lineWidth":0,"shadow":false,"space":0},"horizontal":{"color":"#8EAADBFF","hasNoneStyle":false,"lineStyle":"Single","lineWidth":0.5,"shadow":false,"space":0},"vertical":{"color":"#8EAADBFF","hasNoneStyle":false,"lineStyle":"Single","lineWidth":0.5,"shadow":false,"space":0}},"gridBefore":0,"gridBeforeWidth":0,"gridBeforeWidthType":"Point","gridAfter":0,"gridAfterWidth":0,"gridAfterWidthType":"Point"}},{
"cells":
[{"blocks":
[{"paragraphFormat":{"styleName":"Normal","listFormat":{},"characterFormat":{},"inlines":
[{"characterFormat":{"fontSize":10,"fontFamily":"Verdana","fontSizeBidi":10,"fontFamilyBidi":"Times New Roman"},"text":"4"}]}]}],"cellFormat":
{"borders":
{"top":
{"color":"#8EAADBFF","hasNoneStyle":false,"lineStyle":"Single","lineWidth":0.5,"shadow":false,"space":0},
"left":
{"color":"#8EAADBFF","hasNoneStyle":false,"lineStyle":"Single","lineWidth":0.5,"shadow":false,"space":0},
"right":
{"color":"#8EAADBFF","hasNoneStyle":false,"lineStyle":"Single","lineWidth":0.5,"shadow":false,"space":0},
"bottom":
{"color":"#8EAADBFF","hasNoneStyle":false,"lineStyle":"Single","lineWidth":0.5,"shadow":false,"space":0},
"diagonalDown":
{"color":"#000000","hasNoneStyle":false,"lineStyle":"None","lineWidth":0,"shadow":false,"space":0},
"diagonalUp":
{"color":"#000000","hasNoneStyle":false,"lineStyle":"None","lineWidth":0,"shadow":false,"space":0},
"horizontal":
{"color":"#000000","hasNoneStyle":false,"lineStyle":"None","lineWidth":0,"shadow":false,"space":0},
"vertical":
{"color":"#000000","hasNoneStyle":false,"lineStyle":"None","lineWidth":0,"shadow":false,"space":0},

```

```

"vertical":{"color":"#000000","hasNoneStyle":false,"lineStyle":"None","lineWidth":0,"shadow":false,"space":0}},{"shading":{"backgroundColor":"#FFFFFF","foregroundColor":"empty","textureStyle":"TextureNone"},"preferredWidth":13.420000076293945,"preferredWidthType":"Percent","cellWidth":64.71214527422465,"columnSpan":1,"rowSpan":1,"verticalAlignment":"Top"},"columnIndex":0},{
"blocks":[{"paragraphFormat":{"styleName":"Normal","listFormat":{}},{"characterFormat":{},"inlines":[{"characterFormat":{"fontSize":10,"fontFamily":"Verdana","fontSizeBidi":10,"fontFamilyBidi":"Times New Roman"},"text":"Tarte au sucre"}]}]},"cellFormat":{"borders":{"top":{"color":"#8EADBFF","hasNoneStyle":false,"lineStyle":"Single","lineWidth":0.5,"shadow":false,"space":0},"left":{"color":"#8EADBFF","hasNoneStyle":false,"lineStyle":"Single","lineWidth":0.5,"shadow":false,"space":0},"right":{"color":"#8EADBFF","hasNoneStyle":false,"lineStyle":"Single","lineWidth":0.5,"shadow":false,"space":0},"bottom":{"color":"#8EADBFF","hasNoneStyle":false,"lineStyle":"Single","lineWidth":0.5,"shadow":false,"space":0},"diagonalDown":{"color":"#000000","hasNoneStyle":false,"lineStyle":"None","lineWidth":0,"shadow":false,"space":0},"diagonalUp":{"color":"#000000","hasNoneStyle":false,"lineStyle":"None","lineWidth":0,"shadow":false,"space":0},"horizontal":{"color":"#000000","hasNoneStyle":false,"lineStyle":"None","lineWidth":0,"shadow":false,"space":0},"vertical":{"color":"#000000","hasNoneStyle":false,"lineStyle":"None","lineWidth":0,"shadow":false,"space":0}},{"shading":{"backgroundColor":"#FFFFFF","foregroundColor":"empty","textureStyle":"TextureNone"},"preferredWidth":48.86000061035156,"preferredWidthType":"Percent","cellWidth":292.87942351880633,"columnSpan":1,"rowSpan":1,"verticalAlignment":"Top"},"columnIndex":1},{
"blocks":[{"paragraphFormat":{"styleName":"Normal","listFormat":{}},{"characterFormat":{},"inlines":[{"characterFormat":{"fontSize":10,"fontFamily":"Verdana","fontSizeBidi":10,"fontFamilyBidi":"Times New Roman"},"text":"47.234"}]}]},"bookmarkType":1,"name":"_GoBack"}]}]},"cellFormat":{"borders":{"top":{"color":"#8EADBFF","hasNoneStyle":false,"lineStyle":"Single","lineWidth":0.5,"shadow":false,"space":0},"left":{"color":"#8EADBFF","hasNoneStyle":false,"lineStyle":"Single","lineWidth":0.5,"shadow":false,"space":0},"right":{"color":"#8EADBFF","hasNoneStyle":false,"lineStyle":"Single","lineWidth":0.5,"shadow":false,"space":0},"bottom":{"color":"#8EADBFF","hasNoneStyle":false,"lineStyle":"Single","lineWidth":0.5,"shadow":false,"space":0},"diagonalDown":{"color":"#000000","hasNoneStyle":false,"lineStyle":"None","lineWidth":0,"shadow":false,"space":0},"diagonalUp":{"color":"#000000","hasNoneStyle":false,"lineStyle":"None","lineWidth":0,"shadow":false,"space":0},"horizontal":{"color":"#000000","hasNoneStyle":false,"lineStyle":"None","lineWidth":0,"shadow":false,"space":0},"vertical":{"color":"#000000","hasNoneStyle":false,"lineStyle":"None","lineWidth":0,"shadow":false,"space":0}},{"shading":{"backgroundColor":"#FFFFFF","foregroundColor":"empty","textureStyle":"TextureNone"},"preferredWidth":37.720001220703125,"preferredWidthType":"Percent","cellWidth":117.95841899993776,"columnSpan":1,"rowSpan":1,"verticalAlignment":"Top"},"columnIndex":2}],{"rowFormat":{"height":14.399999618530273,"allowBreakAcrossPages":true,"heightType":"Exactly","isHeader":false,"borders":{"top":{"color":"#8EADBFF","hasNoneStyle":false,"lineStyle":"Single","lineWidth":0.5,"shadow":false,"space":0},"left":{"color":"#8EADBFF","hasNoneStyle":false,"lineStyle":"Single","lineWidth":0.5,"shadow":false,"space":0},"right":{"color":"#8EADBFF","hasNoneStyle":false,"lineStyle":"Single","lineWidth":0.5,"shadow":false,"space":0},"bottom":{"color":"#8EADBFF","hasNoneStyle":false,"lineStyle":"Single","lineWidth":0.5,"shadow":false,"space":0},"diagonalDown":{"color":"#000000","hasNoneStyle":false,"lineStyle":"None","lineWidth":0,"shadow":false,"space":0},"diagonalUp":{"color":"#000000","hasNoneStyle":false,"lineStyle":"None","lineWidth":0,"shadow":false,"space":0},"horizontal":{"color":"#8EADBFF","hasNoneStyle":false,"lineStyle":"Single","lineWidth":0.5,"shadow":false,"space":0},"vertical":{"

```



```

color":"#8EADBFF","hasNoneStyle":false,"lineStyle":"Single","lineWidth":0.5
,"shadow":false,"space":0}},{"gridBefore":0,"gridBeforeWidth":0,"gridBeforeWi
dthType":"Point","gridAfter":0,"gridAfterWidth":0,"gridAfterWidthType":"Poin
t"}},{{"cells":[{"blocks":[{"paragraphFormat":{"styleName":"Normal","listForm
at":{},"characterFormat":{},"inlines":[{"characterFormat":{},"bookmarkType"
:0,"name":"_GoBack"},"characterFormat":{"fontSize":10,"fontFamily":"Verdana
","fontSizeBidi":10,"fontFamilyBidi":"Times New
Roman"},"text":"5"}]}]},"cellFormat":{"borders":{"top":{"color":"#8EADBFF","
hasNoneStyle":false,"lineStyle":"Single","lineWidth":0.5,"shadow":false,"spa
ce":0},"left":{"color":"#8EADBFF","hasNoneStyle":false,"lineStyle":"Single"
,"lineWidth":0.5,"shadow":false,"space":0},"right":{"color":"#8EADBFF","has
NoneStyle":false,"lineStyle":"Single","lineWidth":0.5,"shadow":false,"space"
:0},"bottom":{"color":"#8EADBFF","hasNoneStyle":false,"lineStyle":"Single",
"lineWidth":0.5,"shadow":false,"space":0},"diagonalDown":{"color":"#000000",
"hasNoneStyle":false,"lineStyle":"None","lineWidth":0,"shadow":false,"space"
:0},"diagonalUp":{"color":"#000000","hasNoneStyle":false,"lineStyle":"None",
"lineWidth":0,"shadow":false,"space":0},"horizontal":{"color":"#000000","has
NoneStyle":false,"lineStyle":"None","lineWidth":0,"shadow":false,"space":0},
"vertical":{"color":"#000000","hasNoneStyle":false,"lineStyle":"None","lineW
idth":0,"shadow":false,"space":0},"shading":{"backgroundColor":"#D9E2F3FF",
"foregroundColor":"empty","textureStyle":"TextureNone"},"preferredWidth":13.
420000076293945,"preferredWidthType":"Percent","cellWidth":64.71214527422465
,"columnSpan":1,"rowSpan":1,"verticalAlignment":"Top"},"columnIndex":0},{{"bl
ocks":[{"paragraphFormat":{"styleName":"Normal","listFormat":{},"characterF
ormat":{},"inlines":[{"characterFormat":{"fontSize":10,"fontFamily":"Verdana
","fontSizeBidi":10,"fontFamilyBidi":"Times New Roman"},"text":"Camembert
Pierrot
"}]}]},"cellFormat":{"borders":{"top":{"color":"#8EADBFF","hasNoneStyle":fal
se,"lineStyle":"Single","lineWidth":0.5,"shadow":false,"space":0},"left":{"c
olor":"#8EADBFF","hasNoneStyle":false,"lineStyle":"Single","lineWidth":0.5,
"shadow":false,"space":0},"right":{"color":"#8EADBFF","hasNoneStyle":false,
"lineStyle":"Single","lineWidth":0.5,"shadow":false,"space":0},"bottom":{"co
lor":"#8EADBFF","hasNoneStyle":false,"lineStyle":"Single","lineWidth":0.5,"
shadow":false,"space":0},"diagonalDown":{"color":"#000000","hasNoneStyle":fa
lse,"lineStyle":"None","lineWidth":0,"shadow":false,"space":0},"diagonalUp":
{"color":"#000000","hasNoneStyle":false,"lineStyle":"None","lineWidth":0,"sh
adow":false,"space":0},"horizontal":{"color":"#000000","hasNoneStyle":false,
"lineStyle":"None","lineWidth":0,"shadow":false,"space":0},"vertical":{"colo
r":"#000000","hasNoneStyle":false,"lineStyle":"None","lineWidth":0,"shadow":
false,"space":0},"shading":{"backgroundColor":"#D9E2F3FF","foregroundColor"
:"empty","textureStyle":"TextureNone"},"preferredWidth":48.86000061035156,"p
referredWidthType":"Percent","cellWidth":292.87942351880633,"columnSpan":1,"
rowSpan":1,"verticalAlignment":"Top"},"columnIndex":1},{{"blocks":[{"paragrap
hFormat":{"styleName":"Normal","listFormat":{},"characterFormat":{},"inline
s":[{"characterFormat":{"fontSize":10,"fontFamily":"Verdana","fontSizeBidi":
10,"fontFamilyBidi":"Times New
Roman"},"text":"46.825"}]}]},"cellFormat":{"borders":{"top":{"color":"#8EADB
FF","hasNoneStyle":false,"lineStyle":"Single","lineWidth":0.5,"shadow":false
,"space":0},"left":{"color":"#8EADBFF","hasNoneStyle":false,"lineStyle":"Si
ngle","lineWidth":0.5,"shadow":false,"space":0},"right":{"color":"#8EADBFF",
"hasNoneStyle":false,"lineStyle":"Single","lineWidth":0.5,"shadow":false,"s
pace":0},"bottom":{"color":"#8EADBFF","hasNoneStyle":false,"lineStyle":"Sin
gle","lineWidth":0.5,"shadow":false,"space":0},"diagonalDown":{"color":"#000
000","hasNoneStyle":false,"lineStyle":"None","lineWidth":0,"shadow":false,"s
pace":0},"diagonalUp":{"color":"#000000","hasNoneStyle":false,"lineStyle":"N
one","lineWidth":0,"shadow":false,"space":0},"horizontal":{"color":"#000000",
"hasNoneStyle":false,"lineStyle":"None","lineWidth":0,"shadow":false,"space

```

```

:0,"vertical":{"color":"#000000","hasNoneStyle":false,"lineStyle":"None",
"lineWidth":0,"shadow":false,"space":0}},{"shading":{"backgroundColor":"#D9E2F
3FF","foregroundColor":"empty","textureStyle":"TextureNone"},"preferredWidth
":37.720001220703125,"preferredWidthType":"Percent","cellWidth":117.95841899
993776,"columnSpan":1,"rowSpan":1,"verticalAlignment":"Top"},"columnIndex":2
}],"rowFormat":{"height":14.399999618530273,"allowBreakAcrossPages":true,"he
ightType":"Exactly","isHeader":false,"borders":{"top":{"color":"#8EADBFF",
"hasNoneStyle":false,"lineStyle":"Single","lineWidth":0.5,"shadow":false,"spa
ce":0},"left":{"color":"#8EADBFF","hasNoneStyle":false,"lineStyle":"Single"
,"lineWidth":0.5,"shadow":false,"space":0},"right":{"color":"#8EADBFF","has
NoneStyle":false,"lineStyle":"Single","lineWidth":0.5,"shadow":false,"space"
:0},"bottom":{"color":"#8EADBFF","hasNoneStyle":false,"lineStyle":"Single",
"lineWidth":0.5,"shadow":false,"space":0},"diagonalDown":{"color":"#000000",
"hasNoneStyle":false,"lineStyle":"None","lineWidth":0,"shadow":false,"space"
:0},"diagonalUp":{"color":"#000000","hasNoneStyle":false,"lineStyle":"None",
"lineWidth":0,"shadow":false,"space":0},"horizontal":{"color":"#8EADBFF","h
asNoneStyle":false,"lineStyle":"Single","lineWidth":0.5,"shadow":false,"spac
e":0},"vertical":{"color":"#8EADBFF","hasNoneStyle":false,"lineStyle":"Sing
le","lineWidth":0.5,"shadow":false,"space":0}},"gridBefore":0,"gridBeforeWid
th":0,"gridBeforeWidthType":"Point","gridAfter":0,"gridAfterWidth":0,"gridAf
terWidthType":"Point"}},{ "cells":[{"blocks":[{"paragraphFormat":{"styleName"
:"Normal","listFormat":{},"characterFormat":{},"inlines":[{"characterFormat
":{"fontSize":10,"fontFamily":"Verdana","fontSizeBidi":10,"fontFamilyBidi":
"Times New
Roman"},"text":"6"}]}]}],"cellFormat":{"borders":{"top":{"color":"#8EADBFF",
"hasNoneStyle":false,"lineStyle":"Single","lineWidth":0.5,"shadow":false,"spa
ce":0},"left":{"color":"#8EADBFF","hasNoneStyle":false,"lineStyle":"Single"
,"lineWidth":0.5,"shadow":false,"space":0},"right":{"color":"#8EADBFF","has
NoneStyle":false,"lineStyle":"Single","lineWidth":0.5,"shadow":false,"space"
:0},"bottom":{"color":"#8EADBFF","hasNoneStyle":false,"lineStyle":"Single",
"lineWidth":0.5,"shadow":false,"space":0},"diagonalDown":{"color":"#000000",
"hasNoneStyle":false,"lineStyle":"None","lineWidth":0,"shadow":false,"space"
:0},"diagonalUp":{"color":"#000000","hasNoneStyle":false,"lineStyle":"None",
"lineWidth":0,"shadow":false,"space":0},"horizontal":{"color":"#000000","has
NoneStyle":false,"lineStyle":"None","lineWidth":0,"shadow":false,"space":0},
"vertical":{"color":"#000000","hasNoneStyle":false,"lineStyle":"None","lineW
idth":0,"shadow":false,"space":0}},{"shading":{"backgroundColor":"#FFFFFFF",
"foregroundColor":"empty","textureStyle":"TextureNone"},"preferredWidth":13.
420000076293945,"preferredWidthType":"Percent","cellWidth":64.71214527422465
,"columnSpan":1,"rowSpan":1,"verticalAlignment":"Top"},"columnIndex":0},{ "bl
ocks":[{"paragraphFormat":{"styleName":"Normal","listFormat":{},"characterF
ormat":{},"inlines":[{"characterFormat":{"fontSize":10,"fontFamily":"Verdana
","fontSizeBidi":10,"fontFamilyBidi":"Times New Roman"},"text":"Gnocchi di
nonna
Alice"}]}]}],"cellFormat":{"borders":{"top":{"color":"#8EADBFF", "hasNoneStyle
":false,"lineStyle":"Single","lineWidth":0.5,"shadow":false,"space":0},"left
":{"color":"#8EADBFF","hasNoneStyle":false,"lineStyle":"Single","lineWidth"
:0.5,"shadow":false,"space":0},"right":{"color":"#8EADBFF","hasNoneStyle":f
alse,"lineStyle":"Single","lineWidth":0.5,"shadow":false,"space":0},"bottom"
":{"color":"#8EADBFF","hasNoneStyle":false,"lineStyle":"Single","lineWidth":
0.5,"shadow":false,"space":0},"diagonalDown":{"color":"#000000","hasNoneStyl
e":false,"lineStyle":"None","lineWidth":0,"shadow":false,"space":0},"diagona
lUp":{"color":"#000000","hasNoneStyle":false,"lineStyle":"None","lineWidth":
0,"shadow":false,"space":0},"horizontal":{"color":"#000000","hasNoneStyle":f
alse,"lineStyle":"None","lineWidth":0,"shadow":false,"space":0},"vertical":{
"color":"#000000","hasNoneStyle":false,"lineStyle":"None","lineWidth":0,"sha
dow":false,"space":0}},{"shading":{"backgroundColor":"#FFFFFFF","foregroundC

```



```

color":"empty","textureStyle":"TextureNone"},"preferredWidth":48.860000610351
56,"preferredWidthType":"Percent","cellWidth":292.87942351880633,"columnSpan
":1,"rowSpan":1,"verticalAlignment":"Top"},"columnIndex":1},{ "blocks":[{"par
agraphFormat":{"styleName":"Normal","listFormat":{},"characterFormat":{},"i
nlines":[{"characterFormat":{"fontSize":10,"fontFamily":"Verdana","fontSizeB
idi":10,"fontFamilyBidi":"Times New
Roman"},"text":"42.593"}]}]},"cellFormat":{"borders":{"top":{"color":"#8EAADB
FF","hasNoneStyle":false,"lineStyle":"Single","lineWidth":0.5,"shadow":false
,"space":0},"left":{"color":"#8EAADBFF","hasNoneStyle":false,"lineStyle":"Si
ngle","lineWidth":0.5,"shadow":false,"space":0},"right":{"color":"#8EAADBFF"
,"hasNoneStyle":false,"lineStyle":"Single","lineWidth":0.5,"shadow":false,"s
pace":0},"bottom":{"color":"#8EAADBFF","hasNoneStyle":false,"lineStyle":"Sin
gle","lineWidth":0.5,"shadow":false,"space":0},"diagonalDown":{"color":"#000
000","hasNoneStyle":false,"lineStyle":"None","lineWidth":0,"shadow":false,"s
pace":0},"diagonalUp":{"color":"#000000","hasNoneStyle":false,"lineStyle":"N
one","lineWidth":0,"shadow":false,"space":0},"horizontal":{"color":"#000000"
,"hasNoneStyle":false,"lineStyle":"None","lineWidth":0,"shadow":false,"space
":0},"vertical":{"color":"#000000","hasNoneStyle":false,"lineStyle":"None","
lineWidth":0,"shadow":false,"space":0},"shading":{"backgroundColor":"#FFFFF
FFF","foregroundColor":"empty","textureStyle":"TextureNone"},"preferredWidth
":37.720001220703125,"preferredWidthType":"Percent","cellWidth":117.95841899
993776,"columnSpan":1,"rowSpan":1,"verticalAlignment":"Top"},"columnIndex":2
}],"rowFormat":{"height":14.399999618530273,"allowBreakAcrossPages":true,"he
ightType":"Exactly","isHeader":false,"borders":{"top":{"color":"#8EAADBFF","
hasNoneStyle":false,"lineStyle":"Single","lineWidth":0.5,"shadow":false,"spa
ce":0},"left":{"color":"#8EAADBFF","hasNoneStyle":false,"lineStyle":"Single"
,"lineWidth":0.5,"shadow":false,"space":0},"right":{"color":"#8EAADBFF","has
NoneStyle":false,"lineStyle":"Single","lineWidth":0.5,"shadow":false,"space"
":0},"bottom":{"color":"#8EAADBFF","hasNoneStyle":false,"lineStyle":"Single"
,"lineWidth":0.5,"shadow":false,"space":0},"diagonalDown":{"color":"#000000"
,"hasNoneStyle":false,"lineStyle":"None","lineWidth":0,"shadow":false,"space
":0},"diagonalUp":{"color":"#000000","hasNoneStyle":false,"lineStyle":"None"
,"lineWidth":0,"shadow":false,"space":0},"horizontal":{"color":"#8EAADBFF","h
asNoneStyle":false,"lineStyle":"Single","lineWidth":0.5,"shadow":false,"spac
e":0},"vertical":{"color":"#8EAADBFF","hasNoneStyle":false,"lineStyle":"Sing
le","lineWidth":0.5,"shadow":false,"space":0},"gridBefore":0,"gridBeforeWid
th":0,"gridBeforeWidthType":"Point","gridAfter":0,"gridAfterWidth":0,"gridAf
terWidthType":"Point"}},{ "cells":[{"blocks":[{"paragraphFormat":{"styleName"
:"Normal","listFormat":{},"characterFormat":{},"inlines":[{"characterFormat
":{"fontSize":10,"fontFamily":"Verdana","fontSizeBidi":10,"fontFamilyBidi":"
Times New
Roman"},"text":"7"}]}]}]},"cellFormat":{"borders":{"top":{"color":"#8EAADBFF","
hasNoneStyle":false,"lineStyle":"Single","lineWidth":0.5,"shadow":false,"spa
ce":0},"left":{"color":"#8EAADBFF","hasNoneStyle":false,"lineStyle":"Single"
,"lineWidth":0.5,"shadow":false,"space":0},"right":{"color":"#8EAADBFF","has
NoneStyle":false,"lineStyle":"Single","lineWidth":0.5,"shadow":false,"space"
":0},"bottom":{"color":"#8EAADBFF","hasNoneStyle":false,"lineStyle":"Single"
,"lineWidth":0.5,"shadow":false,"space":0},"diagonalDown":{"color":"#000000"
,"hasNoneStyle":false,"lineStyle":"None","lineWidth":0,"shadow":false,"space
":0},"diagonalUp":{"color":"#000000","hasNoneStyle":false,"lineStyle":"None"
,"lineWidth":0,"shadow":false,"space":0},"horizontal":{"color":"#000000","has
NoneStyle":false,"lineStyle":"None","lineWidth":0,"shadow":false,"space":0},
"vertical":{"color":"#000000","hasNoneStyle":false,"lineStyle":"None","lineW
idth":0,"shadow":false,"space":0},"shading":{"backgroundColor":"#D9E2F3FF",
"foregroundColor":"empty","textureStyle":"TextureNone"},"preferredWidth":13.
420000076293945,"preferredWidthType":"Percent","cellWidth":64.71214527422465
,"columnSpan":1,"rowSpan":1,"verticalAlignment":"Top"},"columnIndex":0},{ "bl

```

```

ocks": [{"paragraphFormat": {"styleName": "Normal", "listFormat": {}}, "characterFormat": {}, "inlines": [{"characterFormat": {"fontSize": 10, "fontFamily": "Verdana", "fontSizeBidi": 10, "fontFamilyBidi": "Times New Roman"}, "text": "Manjimup Dried Apples"}]}], "cellFormat": {"borders": {"top": {"color": "#8EAADBFF", "hasNoneStyle": false, "lineStyle": "Single", "lineWidth": 0.5, "shadow": false, "space": 0}, "left": {"color": "#8EAADBFF", "hasNoneStyle": false, "lineStyle": "Single", "lineWidth": 0.5, "shadow": false, "space": 0}, "right": {"color": "#8EAADBFF", "hasNoneStyle": false, "lineStyle": "Single", "lineWidth": 0.5, "shadow": false, "space": 0}, "bottom": {"color": "#8EAADBFF", "hasNoneStyle": false, "lineStyle": "Single", "lineWidth": 0.5, "shadow": false, "space": 0}, "diagonalDown": {"color": "#000000", "hasNoneStyle": false, "lineStyle": "None", "lineWidth": 0, "shadow": false, "space": 0}, "diagonalUp": {"color": "#000000", "hasNoneStyle": false, "lineStyle": "None", "lineWidth": 0, "shadow": false, "space": 0}, "horizontal": {"color": "#000000", "hasNoneStyle": false, "lineStyle": "None", "lineWidth": 0, "shadow": false, "space": 0}, "vertical": {"color": "#000000", "hasNoneStyle": false, "lineStyle": "None", "lineWidth": 0, "shadow": false, "space": 0}}, "shading": {"backgroundColor": "#D9E2F3FF", "foregroundColor": "empty", "textureStyle": "TextureNone"}, "preferredWidth": 48.86000061035156, "preferredWidthType": "Percent", "cellWidth": 292.87942351880633, "columnSpan": 1, "rowSpan": 1, "verticalAlignment": "Top"}, {"columnIndex": 1}, {"blocks": [{"paragraphFormat": {"styleName": "Normal", "listFormat": {}}, "characterFormat": {}, "inlines": [{"characterFormat": {"fontSize": 10, "fontFamily": "Verdana", "fontSizeBidi": 10, "fontFamilyBidi": "Times New Roman"}, "text": "41.819"}]}], "cellFormat": {"borders": {"top": {"color": "#8EAADBFF", "hasNoneStyle": false, "lineStyle": "Single", "lineWidth": 0.5, "shadow": false, "space": 0}, "left": {"color": "#8EAADBFF", "hasNoneStyle": false, "lineStyle": "Single", "lineWidth": 0.5, "shadow": false, "space": 0}, "right": {"color": "#8EAADBFF", "hasNoneStyle": false, "lineStyle": "Single", "lineWidth": 0.5, "shadow": false, "space": 0}, "bottom": {"color": "#8EAADBFF", "hasNoneStyle": false, "lineStyle": "Single", "lineWidth": 0.5, "shadow": false, "space": 0}, "diagonalDown": {"color": "#000000", "hasNoneStyle": false, "lineStyle": "None", "lineWidth": 0, "shadow": false, "space": 0}, "diagonalUp": {"color": "#000000", "hasNoneStyle": false, "lineStyle": "None", "lineWidth": 0, "shadow": false, "space": 0}, "horizontal": {"color": "#000000", "hasNoneStyle": false, "lineStyle": "None", "lineWidth": 0, "shadow": false, "space": 0}, "vertical": {"color": "#000000", "hasNoneStyle": false, "lineStyle": "None", "lineWidth": 0, "shadow": false, "space": 0}}, "shading": {"backgroundColor": "#D9E2F3FF", "foregroundColor": "empty", "textureStyle": "TextureNone"}, "preferredWidth": 37.720001220703125, "preferredWidthType": "Percent", "cellWidth": 117.95841899993776, "columnSpan": 1, "rowSpan": 1, "verticalAlignment": "Top"}, {"columnIndex": 2}], "rowFormat": {"height": 14.399999618530273, "allowBreakAcrossPages": true, "heightType": "Exactly", "isHeader": false, "borders": {"top": {"color": "#8EAADBFF", "hasNoneStyle": false, "lineStyle": "Single", "lineWidth": 0.5, "shadow": false, "space": 0}, "left": {"color": "#8EAADBFF", "hasNoneStyle": false, "lineStyle": "Single", "lineWidth": 0.5, "shadow": false, "space": 0}, "right": {"color": "#8EAADBFF", "hasNoneStyle": false, "lineStyle": "Single", "lineWidth": 0.5, "shadow": false, "space": 0}, "bottom": {"color": "#8EAADBFF", "hasNoneStyle": false, "lineStyle": "Single", "lineWidth": 0.5, "shadow": false, "space": 0}, "diagonalDown": {"color": "#000000", "hasNoneStyle": false, "lineStyle": "None", "lineWidth": 0, "shadow": false, "space": 0}, "diagonalUp": {"color": "#000000", "hasNoneStyle": false, "lineStyle": "None", "lineWidth": 0, "shadow": false, "space": 0}, "horizontal": {"color": "#8EAADBFF", "hasNoneStyle": false, "lineStyle": "Single", "lineWidth": 0.5, "shadow": false, "space": 0}, "vertical": {"color": "#8EAADBFF", "hasNoneStyle": false, "lineStyle": "Single", "lineWidth": 0.5, "shadow": false, "space": 0}}, "gridBefore": 0, "gridBeforeWidth": 0, "gridBeforeWidthType": "Point", "gridAfter": 0, "gridAfterWidth": 0, "gridAfterWidthType": "Point"}, {"cells": [{"blocks": [{"paragraphFormat": {"styleName": "Normal", "listFormat": {}}, "characterFormat": {}, "inlines": [{"characterFormat": {"fontSize": 10, "fontFamily": "Verdana", "fontSizeBidi": 10, "fontFamilyBidi": "

```

```

Times New
Roman"}, "text": "8" ] ] ] ], "cellFormat": { "borders": { "top": { "color": "#8EADBFF", "hasNoneStyle": false, "lineStyle": "Single", "lineWidth": 0.5, "shadow": false, "space": 0 }, "left": { "color": "#8EADBFF", "hasNoneStyle": false, "lineStyle": "Single", "lineWidth": 0.5, "shadow": false, "space": 0 }, "right": { "color": "#8EADBFF", "hasNoneStyle": false, "lineStyle": "Single", "lineWidth": 0.5, "shadow": false, "space": 0 }, "bottom": { "color": "#8EADBFF", "hasNoneStyle": false, "lineStyle": "Single", "lineWidth": 0.5, "shadow": false, "space": 0 }, "diagonalDown": { "color": "#000000", "hasNoneStyle": false, "lineStyle": "None", "lineWidth": 0, "shadow": false, "space": 0 }, "diagonalUp": { "color": "#000000", "hasNoneStyle": false, "lineStyle": "None", "lineWidth": 0, "shadow": false, "space": 0 }, "horizontal": { "color": "#000000", "hasNoneStyle": false, "lineStyle": "None", "lineWidth": 0, "shadow": false, "space": 0 }, "vertical": { "color": "#000000", "hasNoneStyle": false, "lineStyle": "None", "lineWidth": 0, "shadow": false, "space": 0 } }, "shading": { "backgroundColor": "#FFFFFF", "foregroundColor": "empty", "textureStyle": "TextureNone" }, "preferredWidth": 13.420000076293945, "preferredWidthType": "Percent", "cellWidth": 64.71214527422465, "columnSpan": 1, "rowSpan": 1, "verticalAlignment": "Top" }, { "blocks": [ { "paragraphFormat": { "styleName": "Normal", "listFormat": {} }, "characterFormat": { "fontSize": 10, "fontFamily": "Verdana", "fontSizeBidi": 10, "fontFamilyBidi": "Times New Roman" }, "text": "Alice Mutton" ] ] ], "cellFormat": { "borders": { "top": { "color": "#8EADBFF", "hasNoneStyle": false, "lineStyle": "Single", "lineWidth": 0.5, "shadow": false, "space": 0 }, "left": { "color": "#8EADBFF", "hasNoneStyle": false, "lineStyle": "Single", "lineWidth": 0.5, "shadow": false, "space": 0 }, "right": { "color": "#8EADBFF", "hasNoneStyle": false, "lineStyle": "Single", "lineWidth": 0.5, "shadow": false, "space": 0 }, "bottom": { "color": "#8EADBFF", "hasNoneStyle": false, "lineStyle": "Single", "lineWidth": 0.5, "shadow": false, "space": 0 }, "diagonalDown": { "color": "#000000", "hasNoneStyle": false, "lineStyle": "None", "lineWidth": 0, "shadow": false, "space": 0 }, "diagonalUp": { "color": "#000000", "hasNoneStyle": false, "lineStyle": "None", "lineWidth": 0, "shadow": false, "space": 0 }, "horizontal": { "color": "#000000", "hasNoneStyle": false, "lineStyle": "None", "lineWidth": 0, "shadow": false, "space": 0 }, "vertical": { "color": "#000000", "hasNoneStyle": false, "lineStyle": "None", "lineWidth": 0, "shadow": false, "space": 0 } }, "shading": { "backgroundColor": "#FFFFFF", "foregroundColor": "empty", "textureStyle": "TextureNone" }, "preferredWidth": 48.86000061035156, "preferredWidthType": "Percent", "cellWidth": 292.87942351880633, "columnSpan": 1, "rowSpan": 1, "verticalAlignment": "Top" }, { "blocks": [ { "paragraphFormat": { "styleName": "Normal", "listFormat": {} }, "characterFormat": { "fontSize": 10, "fontFamily": "Verdana", "fontSizeBidi": 10, "fontFamilyBidi": "Times New Roman" }, "text": "32.698" ] ] ], "cellFormat": { "borders": { "top": { "color": "#8EADBFF", "hasNoneStyle": false, "lineStyle": "Single", "lineWidth": 0.5, "shadow": false, "space": 0 }, "left": { "color": "#8EADBFF", "hasNoneStyle": false, "lineStyle": "Single", "lineWidth": 0.5, "shadow": false, "space": 0 }, "right": { "color": "#8EADBFF", "hasNoneStyle": false, "lineStyle": "Single", "lineWidth": 0.5, "shadow": false, "space": 0 }, "bottom": { "color": "#8EADBFF", "hasNoneStyle": false, "lineStyle": "Single", "lineWidth": 0.5, "shadow": false, "space": 0 }, "diagonalDown": { "color": "#000000", "hasNoneStyle": false, "lineStyle": "None", "lineWidth": 0, "shadow": false, "space": 0 }, "diagonalUp": { "color": "#000000", "hasNoneStyle": false, "lineStyle": "None", "lineWidth": 0, "shadow": false, "space": 0 }, "horizontal": { "color": "#000000", "hasNoneStyle": false, "lineStyle": "None", "lineWidth": 0, "shadow": false, "space": 0 }, "vertical": { "color": "#000000", "hasNoneStyle": false, "lineStyle": "None", "lineWidth": 0, "shadow": false, "space": 0 } }, "shading": { "backgroundColor": "#FFFFFF", "foregroundColor": "empty", "textureStyle": "TextureNone" }, "preferredWidth": 37.720001220703125, "preferredWidthType": "Percent", "cellWidth": 117.95841899993776, "columnSpan": 1, "rowSpan": 1, "verticalAlignment": "Top" }, { "rowFormat": { "height": 14.3999999618530273, "allowBreakAcrossPages": true, "heightType": "Exactly", "isHeader": false, "borders": { "top": { "color": "#8EADBFF", "

```

```

hasNoneStyle":false,"lineStyle":"Single","lineWidth":0.5,"shadow":false,"space":0},
"left":{"color":"#8EADBFF","hasNoneStyle":false,"lineStyle":"Single","lineWidth":0.5,"shadow":false,"space":0},
"right":{"color":"#8EADBFF","hasNoneStyle":false,"lineStyle":"Single","lineWidth":0.5,"shadow":false,"space":0},
"bottom":{"color":"#8EADBFF","hasNoneStyle":false,"lineStyle":"Single","lineWidth":0.5,"shadow":false,"space":0},
"diagonalDown":{"color":"#000000","hasNoneStyle":false,"lineStyle":"None","lineWidth":0,"shadow":false,"space":0},
"diagonalUp":{"color":"#000000","hasNoneStyle":false,"lineStyle":"None","lineWidth":0,"shadow":false,"space":0},
"horizontal":{"color":"#8EADBFF","hasNoneStyle":false,"lineStyle":"Single","lineWidth":0.5,"shadow":false,"space":0},
"vertical":{"color":"#8EADBFF","hasNoneStyle":false,"lineStyle":"Single","lineWidth":0.5,"shadow":false,"space":0}},
"gridBefore":0,"gridBeforeWidth":0,"gridAfterWidth":0,"gridAfterWidthType":"Point"},
{"cells":[{"blocks":[{"paragraphFormat":{"styleName":"Normal","listFormat":{},"characterFormat":{"fontSize":10,"fontFamily":"Verdana","fontSizeBidi":10,"fontFamilyBidi":"Times New Roman"},"text":"9"}]}],
"cellFormat":{"borders":{"top":{"color":"#8EADBFF","hasNoneStyle":false,"lineStyle":"Single","lineWidth":0.5,"shadow":false,"space":0},
"left":{"color":"#8EADBFF","hasNoneStyle":false,"lineStyle":"Single","lineWidth":0.5,"shadow":false,"space":0},
"right":{"color":"#8EADBFF","hasNoneStyle":false,"lineStyle":"Single","lineWidth":0.5,"shadow":false,"space":0},
"bottom":{"color":"#8EADBFF","hasNoneStyle":false,"lineStyle":"Single","lineWidth":0.5,"shadow":false,"space":0},
"diagonalDown":{"color":"#000000","hasNoneStyle":false,"lineStyle":"None","lineWidth":0,"shadow":false,"space":0},
"diagonalUp":{"color":"#000000","hasNoneStyle":false,"lineStyle":"None","lineWidth":0,"shadow":false,"space":0},
"horizontal":{"color":"#000000","hasNoneStyle":false,"lineStyle":"None","lineWidth":0,"shadow":false,"space":0},
"vertical":{"color":"#000000","hasNoneStyle":false,"lineStyle":"None","lineWidth":0,"shadow":false,"space":0}},
"shading":{"backgroundColor":"#D9E2F3FF","foregroundColor":"empty","textureStyle":"TextureNone"},"preferredWidth":13.420000076293945,"preferredWidthType":"Percent","cellWidth":64.71214527422465,"columnSpan":1,"rowSpan":1,"verticalAlignment":"Top"},
{"blocks":[{"paragraphFormat":{"styleName":"Normal","listFormat":{},"characterFormat":{"fontSize":10,"fontFamily":"Verdana","fontSizeBidi":10,"fontFamilyBidi":"Times New Roman"},"text":"Carnarvon Tigers"}]}],
"cellFormat":{"borders":{"top":{"color":"#8EADBFF","hasNoneStyle":false,"lineStyle":"Single","lineWidth":0.5,"shadow":false,"space":0},
"left":{"color":"#8EADBFF","hasNoneStyle":false,"lineStyle":"Single","lineWidth":0.5,"shadow":false,"space":0},
"right":{"color":"#8EADBFF","hasNoneStyle":false,"lineStyle":"Single","lineWidth":0.5,"shadow":false,"space":0},
"bottom":{"color":"#8EADBFF","hasNoneStyle":false,"lineStyle":"Single","lineWidth":0.5,"shadow":false,"space":0},
"diagonalDown":{"color":"#000000","hasNoneStyle":false,"lineStyle":"None","lineWidth":0,"shadow":false,"space":0},
"diagonalUp":{"color":"#000000","hasNoneStyle":false,"lineStyle":"None","lineWidth":0,"shadow":false,"space":0},
"horizontal":{"color":"#000000","hasNoneStyle":false,"lineStyle":"None","lineWidth":0,"shadow":false,"space":0},
"vertical":{"color":"#000000","hasNoneStyle":false,"lineStyle":"None","lineWidth":0,"shadow":false,"space":0}},
"shading":{"backgroundColor":"#D9E2F3FF","foregroundColor":"empty","textureStyle":"TextureNone"},"preferredWidth":48.86000061035156,"preferredWidthType":"Percent","cellWidth":292.87942351880633,"columnSpan":1,"rowSpan":1,"verticalAlignment":"Top"},
{"blocks":[{"paragraphFormat":{"styleName":"Normal","listFormat":{},"characterFormat":{"fontSize":10,"fontFamily":"Verdana","fontSizeBidi":10,"fontFamilyBidi":"Times New Roman"},"text":"29.171"}]}],
"cellFormat":{"borders":{"top":{"color":"#8EADBFF","hasNoneStyle":false,"lineStyle":"Single","lineWidth":0.5,"shadow":false

```

```
"space":0}, {"color":"#8EADBFFF","hasNoneStyle":false,"lineStyle":"Single"}, {"lineWidth":0.5, "shadow":false, "space":0}, {"right":{"color":"#8EADBFFF", "hasNoneStyle":false, "lineStyle":"Single", "lineWidth":0.5, "shadow":false, "space":0}, {"bottom":{"color":"#8EADBFFF", "hasNoneStyle":false, "lineStyle":"Single", "lineWidth":0.5, "shadow":false, "space":0}, {"diagonalDown":{"color":"#000000", "hasNoneStyle":false, "lineStyle":"None", "lineWidth":0, "shadow":false, "space":0}, {"diagonalUp":{"color":"#000000", "hasNoneStyle":false, "lineStyle":"None", "lineWidth":0, "shadow":false, "space":0}, {"horizontal":{"color":"#000000", "hasNoneStyle":false, "lineStyle":"None", "lineWidth":0, "shadow":false, "space":0}, {"vertical":{"color":"#000000", "hasNoneStyle":false, "lineStyle":"None", "lineWidth":0, "shadow":false, "space":0}}, {"shading":{"backgroundColor":"#D9E2F3FF", "foregroundColor":"empty", "textureStyle":"TextureNone"}}, {"preferredWidth":37.720001220703125, "preferredWidthType":"Percent", "cellWidth":117.95841899993776, "columnSpan":1, "rowSpan":1, "verticalAlignment":"Top"}, {"columnIndex":2}], {"rowFormat":{"height":14.3999999618530273, "allowBreakAcrossPages":true, "heightType":"Exactly", "isHeader":false, "borders":{"top":{"color":"#8EADBFFF", "hasNoneStyle":false, "lineStyle":"Single", "lineWidth":0.5, "shadow":false, "space":0}, {"left":{"color":"#8EADBFFF", "hasNoneStyle":false, "lineStyle":"Single", "lineWidth":0.5, "shadow":false, "space":0}, {"right":{"color":"#8EADBFFF", "hasNoneStyle":false, "lineStyle":"Single", "lineWidth":0.5, "shadow":false, "space":0}, {"bottom":{"color":"#8EADBFFF", "hasNoneStyle":false, "lineStyle":"Single", "lineWidth":0.5, "shadow":false, "space":0}, {"diagonalDown":{"color":"#000000", "hasNoneStyle":false, "lineStyle":"None", "lineWidth":0, "shadow":false, "space":0}, {"diagonalUp":{"color":"#000000", "hasNoneStyle":false, "lineStyle":"None", "lineWidth":0, "shadow":false, "space":0}, {"horizontal":{"color":"#8EADBFFF", "hasNoneStyle":false, "lineStyle":"Single", "lineWidth":0.5, "shadow":false, "space":0}, {"vertical":{"color":"#8EADBFFF", "hasNoneStyle":false, "lineStyle":"Single", "lineWidth":0.5, "shadow":false, "space":0}}}, {"gridBefore":0, "gridBeforeWidth":0, "gridBeforeWidthType":"Point", "gridAfter":0, "gridAfterWidth":0, "gridAfterWidthType":"Point"}}, {"cells":[{"blocks":[{"paragraphFormat":{"styleName":"Normal", "listFormat":{}}, {"characterFormat":{}}, {"inlines":[{"characterFormat":{"fontSize":10, "fontFamily":"Verdana", "fontSizeBidi":10, "fontFamilyBidi":"Times New Roman"}}, {"text":"10"}]}]}, {"cellFormat":{"borders":{"top":{"color":"#8EADBFFF", "hasNoneStyle":false, "lineStyle":"Single", "lineWidth":0.5, "shadow":false, "space":0}, {"left":{"color":"#8EADBFFF", "hasNoneStyle":false, "lineStyle":"Single", "lineWidth":0.5, "shadow":false, "space":0}, {"right":{"color":"#8EADBFFF", "hasNoneStyle":false, "lineStyle":"Single", "lineWidth":0.5, "shadow":false, "space":0}, {"bottom":{"color":"#8EADBFFF", "hasNoneStyle":false, "lineStyle":"Single", "lineWidth":0.5, "shadow":false, "space":0}, {"diagonalDown":{"color":"#000000", "hasNoneStyle":false, "lineStyle":"None", "lineWidth":0, "shadow":false, "space":0}, {"diagonalUp":{"color":"#000000", "hasNoneStyle":false, "lineStyle":"None", "lineWidth":0, "shadow":false, "space":0}, {"horizontal":{"color":"#000000", "hasNoneStyle":false, "lineStyle":"None", "lineWidth":0, "shadow":false, "space":0}, {"vertical":{"color":"#000000", "hasNoneStyle":false, "lineStyle":"None", "lineWidth":0, "shadow":false, "space":0}}, {"shading":{"backgroundColor":"#FFFFFFF", "foregroundColor":"empty", "textureStyle":"TextureNone"}}, {"preferredWidth":13.420000076293945, "preferredWidthType":"Percent", "cellWidth":64.71214527422465, "columnSpan":1, "rowSpan":1, "verticalAlignment":"Top"}, {"columnIndex":0}, {"blocks":[{"paragraphFormat":{"styleName":"Normal", "listFormat":{}}, {"characterFormat":{}}, {"inlines":[{"characterFormat":{"fontSize":10, "fontFamily":"Verdana", "fontSizeBidi":10, "fontFamilyBidi":"Times New Roman"}}, {"text":"Rössle Sauerkraut."}]}]}, {"cellFormat":{"borders":{"top":{"color":"#8EADBFFF", "hasNoneStyle":false, "lineStyle":"Single", "lineWidth":0.5, "shadow":false, "space":0}, {"left":{"color":"#8EADBFFF", "hasNoneStyle":false, "lineStyle":"Single", "lineWidth":0.5, "shadow":false, "space":0}, {"right":{"color":"#8EADBFFF", "hasNoneStyle":false, "lineStyle":"Single", "lineWidth":0.5, "shadow":false, "space":0}, {"bottom":{"color":"#8EADBFFF", "hasNoneStyle":false, "lineStyle":"Single", "lineWidth":0.5, "shadow":false, "space":0}, {"diagonalDown":{"color":"#000000", "hasNoneStyle":false, "lineStyle":"None", "lineWidth":0, "shadow":false, "space":0}, {"diagonalUp":{"color":"#000000", "hasNoneStyle":false, "lineStyle":"None", "lineWidth":0, "shadow":false, "space":0}, {"horizontal":{"color":"#000000", "hasNoneStyle":false, "lineStyle":"None", "lineWidth":0, "shadow":false, "space":0}, {"vertical":{"color":"#000000", "hasNoneStyle":false, "lineStyle":"None", "lineWidth":0, "shadow":false, "space":0}}, {"shading":{"backgroundColor":"#FFFFFFF", "foregroundColor":"empty", "textureStyle":"TextureNone"}}, {"preferredWidth":13.420000076293945, "preferredWidthType":"Percent", "cellWidth":64.71214527422465, "columnSpan":1, "rowSpan":1, "verticalAlignment":"Top"}, {"columnIndex":0}, {"blocks [{"
```

```

ottom":{"color":"#8EADBFF","hasNoneStyle":false,"lineStyle":"Single","lineW
idth":0.5,"shadow":false,"space":0},"diagonalDown":{"color":"#000000","hasNo
neStyle":false,"lineStyle":"None","lineWidth":0,"shadow":false,"space":0},"d
iagonalUp":{"color":"#000000","hasNoneStyle":false,"lineStyle":"None","lineW
idth":0,"shadow":false,"space":0},"horizontal":{"color":"#000000","hasNoneSt
yle":false,"lineStyle":"None","lineWidth":0,"shadow":false,"space":0},"verti
cal":{"color":"#000000","hasNoneStyle":false,"lineStyle":"None","lineWidth":
0,"shadow":false,"space":0}},"shading":{"backgroundColor":"#FFFFFF","foreg
roundColor":"empty","textureStyle":"TextureNone"},"preferredWidth":48.860000
61035156,"preferredWidthType":"Percent","cellWidth":292.87942351880633,"colu
mnSpan":1,"rowSpan":1,"verticalAlignment":"Top"},"columnIndex":1},{ "blocks":
[{"paragraphFormat":{"styleName":"Normal","listFormat":{},"characterFormat"
:{},"inlines":[{"characterFormat":{"fontSize":10,"fontFamily":"Verdana","fon
tSizeBidi":10,"fontFamilyBidi":"Times New
Roman"},"text":"25.696"}]}]},"cellFormat":{"borders":{"top":{"color":"#8EADB
FF","hasNoneStyle":false,"lineStyle":"Single","lineWidth":0.5,"shadow":false
,"space":0},"left":{"color":"#8EADBFF","hasNoneStyle":false,"lineStyle":"Si
ngle","lineWidth":0.5,"shadow":false,"space":0},"right":{"color":"#8EADBFF"
,"hasNoneStyle":false,"lineStyle":"Single","lineWidth":0.5,"shadow":false,"s
pace":0},"bottom":{"color":"#8EADBFF","hasNoneStyle":false,"lineStyle":"Sin
gle","lineWidth":0.5,"shadow":false,"space":0},"diagonalDown":{"color":"#000
000","hasNoneStyle":false,"lineStyle":"None","lineWidth":0,"shadow":false,"s
pace":0},"diagonalUp":{"color":"#000000","hasNoneStyle":false,"lineStyle":"N
one","lineWidth":0,"shadow":false,"space":0},"horizontal":{"color":"#000000"
,"hasNoneStyle":false,"lineStyle":"None","lineWidth":0,"shadow":false,"space
":0},"vertical":{"color":"#000000","hasNoneStyle":false,"lineStyle":"None","
lineWidth":0,"shadow":false,"space":0}},"shading":{"backgroundColor":"#FFFFF
FF","foregroundColor":"empty","textureStyle":"TextureNone"},"preferredWidth
":37.720001220703125,"preferredWidthType":"Percent","cellWidth":117.95841899
993776,"columnSpan":1,"rowSpan":1,"verticalAlignment":"Top"},"columnIndex":2
}],"rowFormat":{"height":14.399999618530273,"allowBreakAcrossPages":true,"he
ightType":"Exactly","isHeader":false,"borders":{"top":{"color":"#8EADBFF","
hasNoneStyle":false,"lineStyle":"Single","lineWidth":0.5,"shadow":false,"spa
ce":0},"left":{"color":"#8EADBFF","hasNoneStyle":false,"lineStyle":"Single"
,"lineWidth":0.5,"shadow":false,"space":0},"right":{"color":"#8EADBFF","has
NoneStyle":false,"lineStyle":"Single","lineWidth":0.5,"shadow":false,"space
":0},"bottom":{"color":"#8EADBFF","hasNoneStyle":false,"lineStyle":"Single",
"lineWidth":0.5,"shadow":false,"space":0},"diagonalDown":{"color":"#000000",
"hasNoneStyle":false,"lineStyle":"None","lineWidth":0,"shadow":false,"space
":0},"diagonalUp":{"color":"#000000","hasNoneStyle":false,"lineStyle":"None",
"lineWidth":0,"shadow":false,"space":0},"horizontal":{"color":"#8EADBFF","h
asNoneStyle":false,"lineStyle":"Single","lineWidth":0.5,"shadow":false,"spac
e":0},"vertical":{"color":"#8EADBFF","hasNoneStyle":false,"lineStyle":"Sing
le","lineWidth":0.5,"shadow":false,"space":0},"gridBefore":0,"gridBeforeWid
th":0,"gridBeforeWidthType":"Point","gridAfter":0,"gridAfterWidth":0,"gridAf
terWidthType":"Point"}]},"grid":[64.71214527422465,292.87942351880633,117.95
841899993776],"tableFormat":{"borders":{"top":{"color":"#8EADBFF","hasNoneS
yle":false,"lineStyle":"Single","lineWidth":0.5,"shadow":false,"space":0},
"left":{"color":"#8EADBFF","hasNoneStyle":false,"lineStyle":"Single","lineW
idth":0.5,"shadow":false,"space":0},"right":{"color":"#8EADBFF","hasNoneStyl
e":false,"lineStyle":"Single","lineWidth":0.5,"shadow":false,"space":0},"bot
tom":{"color":"#8EADBFF","hasNoneStyle":false,"lineStyle":"Single","lineWid
th":0.5,"shadow":false,"space":0},"diagonalDown":{"color":"#000000","hasNone
Style":false,"lineStyle":"None","lineWidth":0,"shadow":false,"space":0},"dia
gonalUp":{"color":"#000000","hasNoneStyle":false,"lineStyle":"None","lineWid
th":0,"shadow":false,"space":0},"horizontal":{"color":"#8EADBFF","hasNoneSt
yle":false,"lineStyle":"Single","lineWidth":0.5,"shadow":false,"space":0},"v

```



```
critical": {"color": "#8EADDBFF", "hasNoneStyle": false, "lineStyle": "Single", "lineWidth": 0.5, "shadow": false, "space": 0}}, {"shading": {"backgroundColor": "#FFFFFF", "foregroundColor": "empty", "textureStyle": "TextureNone"}, "cellSpacing": 0, "leftIndent": 0, "tableAlignment": "Left", "topMargin": 0, "rightMargin": 0.5, "leftMargin": 0.5, "bottomMargin": 0, "preferredWidth": 475.54998779296875, "preferredWidthType": "Point", "bidi": false, "allowAutoFit": true}, {"description": null, "title": null}, {"paragraphFormat": {"afterSpacing": 0, "styleName": "Normal", "listFormat": {}}, {"characterFormat": {"fontFamily": "Calibri", "fontColor": "#000000FF", "fontFamilyBidi": "Calibri"}, {"inlines": []}, {"paragraphFormat": {"afterSpacing": 0, "styleName": "Normal", "listFormat": {}}, {"characterFormat": {"inlines": [{"characterFormat": {"fontSize": 10, "fontFamily": "Verdana", "fontSizeBidi": 10, "fontFamilyBidi": "Times New Roman"}, "text": "The best-selling product of the company is Cote de Blaye, being part of the Beverages"}]}, {"characterFormat": {"fontSize": 10, "fontFamily": "Verdana", "fontSizeBidi": 10, "fontFamilyBidi": "Times New Roman"}, "text": "category. The contribution of this product to the sum of our sales is $ 141.396."}]}, {"paragraphFormat": {"afterSpacing": 0, "lineSpacing": 1, "lineSpacingType": "Multiple", "styleName": "Normal", "listFormat": {}}, {"characterFormat": {"fontSize": 10, "fontFamily": "Verdana", "fontSizeBidi": 10, "fontFamilyBidi": "Times New Roman"}, {"inlines": []}, {"paragraphFormat": {"afterSpacing": 0, "lineSpacing": 1, "lineSpacingType": "Multiple", "styleName": "Normal", "listFormat": {}}, {"characterFormat": {"fontSize": 10, "fontFamily": "Verdana", "fontSizeBidi": 10, "fontFamilyBidi": "Times New Roman"}, {"inlines": []}, {"paragraphFormat": {"styleName": "Normal", "listFormat": {}}, {"characterFormat": {"inlines": [{"characterFormat": {}, "chartLegend": {"position": "Right", "chartTitleArea": {"fontName": "+mn-1t", "fontSize": 9, "layout": {"layoutX": 0, "layoutY": 0}, "dataFormat": {"fill": {"foreColor": "000000", "rgb": "#000000"}, "line": {"color": "808080", "rgb": "#808080"}}}}}], "chartTitleArea": {"fontName": "+mn-1t", "fontSize": 14, "layout": {"layoutX": 0, "layoutY": 0}, "dataFormat": {"fill": {"foreColor": "000000", "rgb": "#000000"}, "line": {"color": "000000", "rgb": "#000000"}}}], "chartArea": {"foreColor": "#FFFFFFFF", "plotArea": {"foreColor": "#000000FF"}}, {"chartCategory": [{"chartData": [{"yValue": 141.396}], "categoryXName": "Côte de Blaye"}, {"chartData": [{"yValue": 80.368}], "categoryXName": "Thüringer Rostbratwurst"}, {"chartData": [{"yValue": 71.155}], "categoryXName": "Raclette Courdavault"}, {"chartData": [{"yValue": 47.234}], "categoryXName": "Tarte au sucre"}, {"chartData": [{"yValue": 46.825}], "categoryXName": "Camembert Pierrot"}, {"chartData": [{"yValue": 42.593}], "categoryXName": "Gnocchi di nonna Alice"}, {"chartData": [{"yValue": 41.819}], "categoryXName": "Manjimup Dried Apples"}, {"chartData": [{"yValue": 32.698}], "categoryXName": "Alice Mutton"}, {"chartData": [{"yValue": 29.171}], "categoryXName": "Carnarvon Tigers"}, {"chartData": [{"yValue": 25.696}], "categoryXName": "Rössle Sauerkraut"}], "chartSeries": [{"dataPoints": [{"fill": {"foreColor": "4472c4", "rgb": "#4472c4"}, "line": {"color": "ffffff", "rgb": "#ffffff"}}, {"fill": {"foreColor": "ed7d31", "rgb": "#ed7d31"}, "line": {"color": "ffffff", "rgb": "#ffffff"}}, {"fill": {"foreColor": "a5a5a5", "rgb": "#a5a5a5"}, "line": {"color": "ffffff", "rgb": "#ffffff"}}, {"fill": {"foreColor": "ffc000", "rgb": "#ffc000"}, "line": {"color": "ffffff", "rgb": "#ffffff"}}, {"fill": {"foreColor": "5b9bd5", "rgb": "#5b9bd5"}, "line": {"color": "ffffff", "rgb": "#ffffff"}}, {"fill": {"foreColor": "70ad47", "rgb": "#70ad47"}, "line": {"color": "ffffff", "rgb": "#ffffff"}}, {"fill": {"foreColor": "264379", "rgb": "#264379"}, "line": {"color": "ffffff", "rgb": "#ffffff"}}, {"fill": {"foreColor": "9f480e", "rgb": "#9f480e"}, "line": {"color": "ffffff", "rgb": "#ffffff"}}, {"fill": {"foreColor": "636363", "rgb": "#636363"}, "line": {"color": "ffffff", "rgb": "#ffffff"}}, {"fill": {"foreColor": "9a7200", "rgb": "#9a7200"}, "line": {"color": "ffffff", "rgb": "#ffffff"}}, {"fill": {"foreColor": "9a7200", "rgb": "#9a7200"}, "line": {"color": "ffffff", "rgb": "#ffffff"}}, {"seriesName": "Sales"}], "chartPrimaryCategoryAxis": {"chartTitle": null, "chartTitleArea": {"layout": {}, "dataFormat": {"fill":
```

```

: {}, "line": {} } }, "categoryType": "Automatic", "fontSize": 11, "fontName": "Calibri",
"numberFormat": "General", "maximumValue": 0, "minimumValue": 0, "majorUnit": 0,
"hasMajorGridLines": false, "hasMinorGridLines": false, "majorTickMark": "TickMark_
_Outside", "minorTickMark": "TickMark_None", "tickLabelPosition": "TickLabelPosi
tion_NextToAxis", "chartPrimaryValueAxis": { "chartTitle": null, "chartTitleArea
": { "layout": {}, "dataFormat": { "fill": {}, "line": {} } }, "fontSize": 11, "fontName":
"Calibri", "maximumValue": 0, "minimumValue": 0, "majorUnit": 0, "hasMajorGridLines
": false, "hasMinorGridLines": false, "majorTickMark": "TickMark_Outside", "minorT
ickMark": "TickMark_None", "tickLabelPosition": "TickLabelPosition_NextToAxis" }
, "chartTitle": "Best Selling
Products", "chartType": "Pie", "gapWidth": 0, "overlap": 0, "height": 225, "width": 43
2 } ], { "paragraphFormat": { "styleName": "Normal", "listFormat": {} }, "characterFor
mat": { "inlines": [], { "paragraphFormat": { "afterSpacing": 0, "lineSpacing": 1, "
lineSpacingType": "Multiple", "styleName": "Normal", "listFormat": {} }, "character
Format": { "fontSize": 10, "fontFamily": "Verdana", "fontSizeBidi": 10, "fontFamilyB
idi": "Verdana", "inlines": [ { "characterFormat": { "fontSize": 10, "fontFamily": "V
erdana", "styleName": "a", "fontSizeBidi": 10, "fontFamilyBidi": "Verdana", "text"
: "According to the above chart, the total count of the selling products is
24 and the average
", { "characterFormat": { "fontSize": 10, "fontFamily": "Verdana", "styleName": "a",
"fontSizeBidi": 10, "fontFamilyBidi": "Verdana", "text": "sales attributed to
this product is $ 5.891 with highest sale $ 15.810 in the month of May in
", { "characterFormat": { "fontSize": 10, "fontFamily": "Verdana", "styleName": "a",
"fontSizeBidi": 10, "fontFamilyBidi": "Verdana", "text": "2014. In the same
year, in the month of March the same product reached the amount of $
", { "characterFormat": { "fontSize": 10, "fontFamily": "Verdana", "styleName": "a",
"fontSizeBidi": 10, "fontFamilyBidi": "Verdana", "text": "15.019. These were the
highest sales of the product among the other products for the year
", { "characterFormat": { "fontSize": 10, "fontFamily": "Verdana", "styleName": "a",
"fontSizeBidi": 10, "fontFamilyBidi": "Verdana", "text": "2014." } ] ] }, "headersFoo
ters": {} }, { "characterFormat": { "bold": false, "italic": false, "fontSize": 11, "fon
tFamily": "Calibri", "underline": "None", "strikethrough": "None", "baselineAlignm
ent": "Normal", "highlightColor": "NoColor", "fontColor": "#000000", "fontSizeBidi
": 11, "fontFamilyBidi": "Calibri", "paragraphFormat": { "leftIndent": 0, "rightInd
ent": 0, "firstLineIndent": 0, "textAlignment": "Left", "beforeSpacing": 0, "afterSp
acing": 8, "lineSpacing": 1.0791666507720947, "lineSpacingType": "Multiple", "list
Format": {}, "bidi": false, "defaultTabWidth": 36, "enforcement": false, "hashValue
": "", "saltValue": "", "formatting": false, "protectionType": "NoProtection", "styl
es": [ { "name": "Normal", "type": "Paragraph", "paragraphFormat": { "listFormat": {} }
, "characterFormat": {}, "next": "Normal", { "name": "Heading
1", "type": "Paragraph", "paragraphFormat": { "beforeSpacing": 12, "afterSpacing": 3
, "lineSpacing": 1, "lineSpacingType": "Multiple", "outlineLevel": "Level1", "listF
ormat": {} }, "characterFormat": { "bold": true, "fontSize": 16, "fontFamily": "Arial"
, "boldBidi": true, "fontSizeBidi": 16, "fontFamilyBidi": "Arial", "basedOn": "Norm
al", "link": "Heading 1 Char", "next": "Normal", { "name": "Heading 1
Char", "type": "Character", "characterFormat": { "bold": true, "fontSize": 16, "fontF
amily": "Arial", "boldBidi": true, "fontSizeBidi": 16, "fontFamilyBidi": "Arial", "
basedOn": "Default Paragraph Font", { "name": "Default Paragraph
Font", "type": "Character", "characterFormat": {}, { "name": "Balloon
Text", "type": "Paragraph", "paragraphFormat": { "afterSpacing": 0, "lineSpacing": 1
, "lineSpacingType": "Multiple", "listFormat": {} }, "characterFormat": { "fontSize"
: 9, "fontFamily": "Segoe UI", "fontSizeBidi": 9, "fontFamilyBidi": "Segoe
UI", "basedOn": "Normal", "link": "Balloon Text Char", { "name": "Balloon Text
Char", "type": "Character", "characterFormat": { "fontSize": 9, "fontFamily": "Segoe
UI", "fontSizeBidi": 9, "fontFamilyBidi": "Segoe UI", "basedOn": "Default
Paragraph
Font", { "name": "a", "type": "Character", "characterFormat": {}, "basedOn": "Defaul

```



```

t Paragraph Font"}, {"name": "Heading
2", "type": "Paragraph", "paragraphFormat": {"leftIndent": 0, "rightIndent": 0, "fir
stLineIndent": 0, "textAlignment": "Left", "beforeSpacing": 2, "afterSpacing": 0, "l
ineSpacing": 1.0791666507720947, "lineSpacingType": "Multiple", "outlineLevel": "
Level2", "listFormat": {}}, "characterFormat": {"fontSize": 13, "fontFamily": "Cali
bri Light", "fontColor": "#2F5496"}, "basedOn": "Normal", "link": "Heading 2
Char", "next": "Normal"}, {"name": "Heading 2
Char", "type": "Character", "characterFormat": {"fontSize": 13, "fontFamily": "Cali
bri Light", "fontColor": "#2F5496"}, "basedOn": "Default Paragraph
Font"}, {"name": "Heading
3", "type": "Paragraph", "paragraphFormat": {"leftIndent": 0, "rightIndent": 0, "fir
stLineIndent": 0, "textAlignment": "Left", "beforeSpacing": 2, "afterSpacing": 0, "l
ineSpacing": 1.0791666507720947, "lineSpacingType": "Multiple", "outlineLevel": "
Level3", "listFormat": {}}, "characterFormat": {"fontSize": 12, "fontFamily": "Cali
bri Light", "fontColor": "#1F3763"}, "basedOn": "Normal", "link": "Heading 3
Char", "next": "Normal"}, {"name": "Heading 3
Char", "type": "Character", "characterFormat": {"fontSize": 12, "fontFamily": "Cali
bri Light", "fontColor": "#1F3763"}, "basedOn": "Default Paragraph
Font"}, {"name": "Heading
4", "type": "Paragraph", "paragraphFormat": {"leftIndent": 0, "rightIndent": 0, "fir
stLineIndent": 0, "textAlignment": "Left", "beforeSpacing": 2, "afterSpacing": 0, "l
ineSpacing": 1.0791666507720947, "lineSpacingType": "Multiple", "outlineLevel": "
Level4", "listFormat": {}}, "characterFormat": {"italic": true, "fontFamily": "Cali
bri Light", "fontColor": "#2F5496"}, "basedOn": "Normal", "link": "Heading 4
Char", "next": "Normal"}, {"name": "Heading 4
Char", "type": "Character", "characterFormat": {"italic": true, "fontFamily": "Cali
bri Light", "fontColor": "#2F5496"}, "basedOn": "Default Paragraph
Font"}, {"name": "Heading
5", "type": "Paragraph", "paragraphFormat": {"leftIndent": 0, "rightIndent": 0, "fir
stLineIndent": 0, "textAlignment": "Left", "beforeSpacing": 2, "afterSpacing": 0, "l
ineSpacing": 1.0791666507720947, "lineSpacingType": "Multiple", "outlineLevel": "
Level5", "listFormat": {}}, "characterFormat": {"fontFamily": "Calibri
Light", "fontColor": "#2F5496"}, "basedOn": "Normal", "link": "Heading 5
Char", "next": "Normal"}, {"name": "Heading 5
Char", "type": "Character", "characterFormat": {"fontFamily": "Calibri
Light", "fontColor": "#2F5496"}, "basedOn": "Default Paragraph
Font"}, {"name": "Heading
6", "type": "Paragraph", "paragraphFormat": {"leftIndent": 0, "rightIndent": 0, "fir
stLineIndent": 0, "textAlignment": "Left", "beforeSpacing": 2, "afterSpacing": 0, "l
ineSpacing": 1.0791666507720947, "lineSpacingType": "Multiple", "outlineLevel": "
Level6", "listFormat": {}}, "characterFormat": {"fontFamily": "Calibri
Light", "fontColor": "#1F3763"}, "basedOn": "Normal", "link": "Heading 6
Char", "next": "Normal"}, {"name": "Heading 6
Char", "type": "Character", "characterFormat": {"fontFamily": "Calibri
Light", "fontColor": "#1F3763"}, "basedOn": "Default Paragraph
Font"}], "lists": [], "abstractLists": []}`;
    //Open default document in Document Editor.
    this.$refs.documenteditor.open(sfdt);
  }
}
</script>
<style>
  @import "../../../node_modules/@syncfusion/ej2-vue-
documenteditor/styles/material.css";
</style>

```

```
{% previewsample "page.domainurl/code-snippet/document-editor/chart-cs1" %}
```

### Supported Chart Types

The following chart types are supported in Document Editor

- Scatter\_Markers
- Bubble
- Area
- Area\_Stacked
- AreaStacked100
- Bar\_Clustered
- Bar\_Stacked
- BarStacked100
- Column\_Clustered
- Column\_Stacked
- ColumnStacked100
- Pie
- Doughnut
- Line
- Line\_Markers
- LineMarkersStacked
- LineMarkersStacked\_100
- Line\_Stacked
- LineStacked100

### Restrict editing in Vue Document editor component

Document Editor provides support to restrict editing. When the protected document includes range permission, then unique user or user group only authorized to edit separate text area.

#### Set current user

You can use the `currentUser` property to authorize the current document user by name, email, or user group name.

The following code shows how to set currentUser

```
`javascript
this.$refs.doceditcontainer.ej2Instances.documentEditor.currentUser = 'engineer@mycompany.com';
`
```

#### Highlighting the text area

You can highlight the editable region of the current user using the `userColor` property.

The following code shows how to set userColor.

```
`javascript
this.$refs.doceditcontainer.ej2Instances.documentEditor..userColor = '#fff000';
`
```

You can toggle the highlight the editable region value using the "highlightEditableRanges" property.

The following code shows how to toggle the highlight editable region value.

```
`javascript
```

```
this.$refs.doceditcontainer.ej2Instances.documentEditor.documentEditorSettings.highlightEditableRanges = true;
```

```
`
```

## Restrict Editing Pane

Restrict Editing Pane provides the following options to manage the document:

- To apply formatting restrictions to the current document, select the allow formatting check box.
- To apply editing restrictions to the current document, select the read only check box.
- To add users to the current document, select more users option and add user from the popup dialog.
- To include range permission to the current document, select parts of the document and choose users who are allowed to freely edit them from the listed check box.
- To apply the chosen editing restrictions, click the **YES, START ENFORCING PROTECTION** button. A dialog box displays asking for a password to protect.
- To stop protection, select **STOP PROTECTION** button. A dialog box displays asking for a password to stop protection.

The following code shows Restrict Editing Pane. To unprotect the document, use password '123'.

### APP.VUE

```
<template>
  <div id="app">
    <ejs-documenteditorcontainer ref='doceditcontainer'
height="370px" style="width: 100%;" id='container'
:enableToolbar='true'></ejs-documenteditorcontainer>
  </div>
</template>
<script>
import Vue from 'vue';
import { DocumentEditorContainerPlugin,
DocumentEditorContainerComponent, Toolbar } from '@syncfusion/ej2-vue-
documenteditor';
Vue.use(DocumentEditorContainerPlugin);
export default {
  data() {
    return {
    };
  },
  mounted: function() {
    let sfdt: string =
`{"sections":[{"blocks":[{"characterFormat":{"fontSize":14.0,"fontSizeBidi":
14.0},"paragraphFormat":{"lineSpacing":32.0,"lineSpacingType":"Exactly","styleName":"Normal"},"inlines":[{"text":"Name","characterFormat":{"bold":true,"fontSize":14.0,"boldBidi":true,"fontSizeBidi":14.0}},{"text":":","characterFormat":{"fontSize":14.0,"fontSizeBidi":14.0}}]},{"rows":[{"rowFormat":{"allowBreakAcrossPages":true,"isHeader":false,"height":20.0,"heightType":"AtLeast"},"borders":{"left":{"lineStyle":"None","lineWidth":0.0,"shadow":false,"space":0.0,"hasNoneStyle":false},"right":{"lineStyle":"None","lineWidth":0.0,"sh
```

```

adow":false,"space":0.0,"hasNoneStyle":false},"top":{"lineStyle":"None","lineWidth":0.0,"shadow":false,"space":0.0,"hasNoneStyle":false},"bottom":{"lineStyle":"None","lineWidth":0.0,"shadow":false,"space":0.0,"hasNoneStyle":false},"vertical":{"lineStyle":"None","lineWidth":0.0,"shadow":false,"space":0.0,"hasNoneStyle":false},"horizontal":{"lineStyle":"None","lineWidth":0.0,"shadow":false,"space":0.0,"hasNoneStyle":false},"diagonalDown":{"lineStyle":"None","lineWidth":0.0,"shadow":false,"space":0.0,"hasNoneStyle":false},"diagonalUp":{"lineStyle":"None","lineWidth":0.0,"shadow":false,"space":0.0,"hasNoneStyle":false}}},"cells":[{"blocks":[{"paragraphFormat":{"styleName":"Normal"},"inlines":[{"editRangeId":"1348272392","columnFirst":0,"columnLast":0,"user":"engineer@mycompany.com"}],"text":"Enter name"}],"editRangeId":"1348272392","editableRangeStart":{"editRangeId":"1348272392","columnFirst":0,"columnLast":0,"user":"engineer@mycompany.com"}]}],"cellFormat":{"columnSpan":1,"rowSpan":1,"preferredWidth":467.5,"preferredWidthType":"Point","verticalAlignment":"Center","isSamePaddingAsTable":true,"borders":{"left":{"lineStyle":"None","lineWidth":0.0,"shadow":false,"space":0.0,"hasNoneStyle":false},"right":{"lineStyle":"None","lineWidth":0.0,"shadow":false,"space":0.0,"hasNoneStyle":false},"top":{"lineStyle":"None","lineWidth":0.0,"shadow":false,"space":0.0,"hasNoneStyle":false},"bottom":{"lineStyle":"None","lineWidth":0.0,"shadow":false,"space":0.0,"hasNoneStyle":false},"vertical":{"lineStyle":"None","lineWidth":0.0,"shadow":false,"space":0.0,"hasNoneStyle":false},"horizontal":{"lineStyle":"None","lineWidth":0.0,"shadow":false,"space":0.0,"hasNoneStyle":false},"diagonalDown":{"lineStyle":"None","lineWidth":0.0,"shadow":false,"space":0.0,"hasNoneStyle":false},"diagonalUp":{"lineStyle":"None","lineWidth":0.0,"shadow":false,"space":0.0,"hasNoneStyle":false}}}}],"title":null,"description":null,"tableFormat":{"allowAutoFit":true,"leftIndent":0.0,"tableAlignment":"Left","preferredWidthType":"Auto","borders":{"left":{"lineStyle":"Single","lineWidth":0.5,"shadow":false,"space":0.0,"hasNoneStyle":false},"right":{"lineStyle":"Single","lineWidth":0.5,"shadow":false,"space":0.0,"hasNoneStyle":false},"top":{"lineStyle":"Single","lineWidth":0.5,"shadow":false,"space":0.0,"hasNoneStyle":false},"bottom":{"lineStyle":"Single","lineWidth":0.5,"shadow":false,"space":0.0,"hasNoneStyle":false},"vertical":{"lineStyle":"Single","lineWidth":0.5,"shadow":false,"space":0.0,"hasNoneStyle":false},"horizontal":{"lineStyle":"Single","lineWidth":0.5,"shadow":false,"space":0.0,"hasNoneStyle":false},"diagonalDown":{"lineStyle":"None","lineWidth":0.0,"shadow":false,"space":0.0,"hasNoneStyle":false},"diagonalUp":{"lineStyle":"None","lineWidth":0.0,"shadow":false,"space":0.0,"hasNoneStyle":false}}},"bidi":false},"characterFormat":{"bold":true,"fontSize":14.0,"boldBidi":true,"fontSizeBidi":14.0},"paragraphFormat":{"lineSpacing":32.0,"lineSpacingType":"Exactly","styleName":"Normal"},"inlines":[{"text":"Designation:", "characterFormat":{"bold":true,"fontSize":14.0,"boldBidi":true,"fontSizeBidi":14.0}}]},"rows":[{"rowFormat":{"allowBreakAcrossPages":true,"isHeader":false,"height":20.0,"heightType":"AtLeast","borders":{"left":{"lineStyle":"None","lineWidth":0.0,"shadow":false,"space":0.0,"hasNoneStyle":false},"right":{"lineStyle":"None","lineWidth":0.0,"shadow":false,"space":0.0,"hasNoneStyle":false},"top":{"lineStyle":"None","lineWidth":0.0,"shadow":false,"space":0.0,"hasNoneStyle":false},"bottom":{"lineStyle":"None","lineWidth":0.0,"shadow":false,"space":0.0,"hasNoneStyle":false},"vertical":{"lineStyle":"None","lineWidth":0.0,"shadow":false,"space":0.0,"hasNoneStyle":false},"horizontal":{"lineStyle":"None","lineWidth":0.0,"shadow":false,"space":0.0,"hasNoneStyle":false},"diagonalDown":{"lineStyle":"None","lineWidth":0.0,"shadow":false,"space":0.0,"hasNoneStyle":false},"diagonalUp":{"lineStyle":"None","lineWidth":0.0,"shadow":false,"space":0.0,"hasNoneStyle":false}}},"cells":[{"blocks":[{"paragraphFormat":{"styleName":"Normal"},"inlines":[{"editRangeId":"808933422","columnFirst":0,"columnLast":0,"user":"engineer@mycompany.com"}],"text":"Enter designation"}],"editRangeId":"808933422","editableRangeStart":{"editRangeId"

```

```

:"808933422","columnFirst":0,"columnLast":0,"user":"engineer@mycompany.com"}
]]]],{"cellFormat":{"columnSpan":1,"rowSpan":1,"preferredWidth":467.5,"prefer
redWidthType":"Point","verticalAlignment":"Center","isSamePaddingAsTable":tr
ue,"borders":{"left":{"lineStyle":"None","lineWidth":0.0,"shadow":false,"spa
ce":0.0,"hasNoneStyle":false},"right":{"lineStyle":"None","lineWidth":0.0,"s
hadow":false,"space":0.0,"hasNoneStyle":false},"top":{"lineStyle":"None","li
neWidth":0.0,"shadow":false,"space":0.0,"hasNoneStyle":false},"bottom":{"lin
eStyle":"None","lineWidth":0.0,"shadow":false,"space":0.0,"hasNoneStyle":fal
se},"vertical":{"lineStyle":"None","lineWidth":0.0,"shadow":false,"space":0.
0,"hasNoneStyle":false},"horizontal":{"lineStyle":"None","lineWidth":0.0,"sh
adow":false,"space":0.0,"hasNoneStyle":false},"diagonalDown":{"lineStyle":"N
one","lineWidth":0.0,"shadow":false,"space":0.0,"hasNoneStyle":false},"diago
nalUp":{"lineStyle":"None","lineWidth":0.0,"shadow":false,"space":0.0,"hasNo
neStyle":false}}}}]],{"title":null,"description":null,"tableFormat":{"allowA
utoFit":true,"leftIndent":0.0,"tableAlignment":"Left","preferredWidthType":"
Auto","borders":{"left":{"lineStyle":"Single","lineWidth":0.5,"shadow":false
,"space":0.0,"hasNoneStyle":false},"right":{"lineStyle":"Single","lineWidth"
:0.5,"shadow":false,"space":0.0,"hasNoneStyle":false},"top":{"lineStyle":"Si
ngle","lineWidth":0.5,"shadow":false,"space":0.0,"hasNoneStyle":false},"bott
om":{"lineStyle":"Single","lineWidth":0.5,"shadow":false,"space":0.0,"hasNon
eStyle":false},"vertical":{"lineStyle":"Single","lineWidth":0.5,"shadow":fal
se,"space":0.0,"hasNoneStyle":false},"horizontal":{"lineStyle":"Single","lin
ewidth":0.5,"shadow":false,"space":0.0,"hasNoneStyle":false},"diagonalDown":
{"lineStyle":"None","lineWidth":0.0,"shadow":false,"space":0.0,"hasNoneStyle
":false},"diagonalUp":{"lineStyle":"None","lineWidth":0.0,"shadow":false,"sp
ace":0.0,"hasNoneStyle":false}},"bidi":false},"characterFormat":{"bold":tr
ue,"fontSize":14.0,"boldBidi":true,"fontSizeBidi":14.0},"paragraphFormat":{"
lineSpacing":32.0,"lineSpacingType":"Exactly","styleName":"Normal"},"inlines
":[{"text":"Email
Address:", "characterFormat":{"bold":true,"fontSize":14.0,"boldBidi":true,"fo
ntSizeBidi":14.0}}, {"name": "_GoBack", "bookmarkType":0}, {"name": "_GoBack", "bo
okmarkType":1}]], {"rows": [{"rowFormat": {"allowBreakAcrossPages": true, "isHead
er": false, "height": 20.0, "heightType": "AtLeast", "borders": {"left": {"lineStyle
": "None", "lineWidth": 0.0, "shadow": false, "space": 0.0, "hasNoneStyle": false}, "r
ight": {"lineStyle": "None", "lineWidth": 0.0, "shadow": false, "space": 0.0, "hasNon
eStyle": false}, "top": {"lineStyle": "None", "lineWidth": 0.0, "shadow": false, "spa
ce": 0.0, "hasNoneStyle": false}, "bottom": {"lineStyle": "None", "lineWidth": 0.0, "
shadow": false, "space": 0.0, "hasNoneStyle": false}, "vertical": {"lineStyle": "Non
e", "lineWidth": 0.0, "shadow": false, "space": 0.0, "hasNoneStyle": false}, "horizon
tal": {"lineStyle": "None", "lineWidth": 0.0, "shadow": false, "space": 0.0, "hasNone
Style": false}, "diagonalDown": {"lineStyle": "None", "lineWidth": 0.0, "shadow": fa
lse, "space": 0.0, "hasNoneStyle": false}, "diagonalUp": {"lineStyle": "None", "line
Width": 0.0, "shadow": false, "space": 0.0, "hasNoneStyle": false}}}], "cells": [{"blo
cks": [{"paragraphFormat": {"styleName": "Normal"}, "inlines": [{"editRangeId": "8
10441411", "columnFirst": 0, "columnLast": 0, "user": "engineer@mycompany.com"}], {"
text": "Enter email
address"}, {"editRangeId": "810441411", "editableRangeStart": {"editRangeId": "81
0441411", "columnFirst": 0, "columnLast": 0, "user": "engineer@mycompany.com"}]]]
, "cellFormat": {"columnSpan": 1, "rowSpan": 1, "preferredWidth": 467.5, "preferredW
idthType": "Point", "verticalAlignment": "Center", "isSamePaddingAsTable": true, "
borders": {"left": {"lineStyle": "None", "lineWidth": 0.0, "shadow": false, "space":
0.0, "hasNoneStyle": false}, "right": {"lineStyle": "None", "lineWidth": 0.0, "shado
w": false, "space": 0.0, "hasNoneStyle": false}, "top": {"lineStyle": "None", "lineWi
dth": 0.0, "shadow": false, "space": 0.0, "hasNoneStyle": false}, "bottom": {"lineSty
le": "None", "lineWidth": 0.0, "shadow": false, "space": 0.0, "hasNoneStyle": false},
"vertical": {"lineStyle": "None", "lineWidth": 0.0, "shadow": false, "space": 0.0, "h
asNoneStyle": false}, "horizontal": {"lineStyle": "None", "lineWidth": 0.0, "shadow

```

```

":false,"space":0.0,"hasNoneStyle":false},"diagonalDown":{"lineStyle":"None",
,"lineWidth":0.0,"shadow":false,"space":0.0,"hasNoneStyle":false},"diagonalUp":{
{"lineStyle":"None","lineWidth":0.0,"shadow":false,"space":0.0,"hasNoneStyle":false}}}}]]],
"title":null,"description":null,"tableFormat":{"allowAutoFit":true,"leftIndent":0.0,"tableAlignment":"Left",
"preferredWidthType":"Auto","borders":{"left":{"lineStyle":"Single","lineWidth":0.5,"shadow":false,"space":0.0,
"hasNoneStyle":false},"right":{"lineStyle":"Single","lineWidth":0.5,"shadow":false,"space":0.0,"hasNoneStyle":false},
"top":{"lineStyle":"Single","lineWidth":0.5,"shadow":false,"space":0.0,"hasNoneStyle":false},"bottom":{
{"lineStyle":"Single","lineWidth":0.5,"shadow":false,"space":0.0,"hasNoneStyle":false},"vertical":{
{"lineStyle":"Single","lineWidth":0.5,"shadow":false,"space":0.0,"hasNoneStyle":false},"horizontal":{
{"lineStyle":"Single","lineWidth":0.5,"shadow":false,"space":0.0,"hasNoneStyle":false},"diagonalDown":{
{"lineStyle":"None","lineWidth":0.0,"shadow":false,"space":0.0,"hasNoneStyle":false},
"diagonalUp":{
{"lineStyle":"None","lineWidth":0.0,"shadow":false,"space":0.0,"hasNoneStyle":false}},
"bidi":false}},{"characterFormat":{"bold":true,"fontSize":14.0,"boldBidi":true,"fontSizeBidi":14.0},
"paragraphFormat":{"lineSpacing":32.0,"lineSpacingType":"Exactly","styleName":"Normal"},"inlines":[{
"text":"Feedbacks:",
"characterFormat":{"bold":true,"fontSize":14.0,"boldBidi":true,"fontSizeBidi":14.0}}]}],
"rows":[{"rowFormat":{"allowBreakAcrossPages":true,"isHeader":false,"height":20.0,"heightType":"AtLeast",
"borders":{"left":{"lineStyle":"None","lineWidth":0.0,"shadow":false,"space":0.0,"hasNoneStyle":false},
"right":{"lineStyle":"None","lineWidth":0.0,"shadow":false,"space":0.0,"hasNoneStyle":false},
"top":{"lineStyle":"None","lineWidth":0.0,"shadow":false,"space":0.0,"hasNoneStyle":false},
"bottom":{"lineStyle":"None","lineWidth":0.0,"shadow":false,"space":0.0,"hasNoneStyle":false},
"vertical":{"lineStyle":"None","lineWidth":0.0,"shadow":false,"space":0.0,"hasNoneStyle":false},
"horizontal":{"lineStyle":"None","lineWidth":0.0,"shadow":false,"space":0.0,"hasNoneStyle":false},
"diagonalDown":{"lineStyle":"None","lineWidth":0.0,"shadow":false,"space":0.0,"hasNoneStyle":false},
"diagonalUp":{"lineStyle":"None","lineWidth":0.0,"shadow":false,"space":0.0,"hasNoneStyle":false}},
"cells":[{"blocks":[{"paragraphFormat":{"styleName":"Normal"},"inlines":[{"editRangeId":"1016946268",
"columnFirst":0,"columnLast":0,"user":"manager@mycompany.com"}],
"text":"Enter the feedbacks"}],
"editRangeId":"1016946268",
"editableRangeStart":{"editRangeId":"1016946268",
"columnFirst":0,"columnLast":0,"user":"manager@mycompany.com"}]}],
"cellFormat":{"columnSpan":1,"rowSpan":1,"preferredWidth":467.5,"preferredWidthType":"Point",
"verticalAlignment":"Center","isSamePaddingAsTable":true,"borders":{"left":{"lineStyle":"None",
"lineWidth":0.0,"shadow":false,"space":0.0,"hasNoneStyle":false},
"right":{"lineStyle":"None","lineWidth":0.0,"shadow":false,"space":0.0,"hasNoneStyle":false},
"top":{"lineStyle":"None","lineWidth":0.0,"shadow":false,"space":0.0,"hasNoneStyle":false},
"bottom":{"lineStyle":"None","lineWidth":0.0,"shadow":false,"space":0.0,"hasNoneStyle":false},
"vertical":{"lineStyle":"None","lineWidth":0.0,"shadow":false,"space":0.0,"hasNoneStyle":false},
"horizontal":{"lineStyle":"None","lineWidth":0.0,"shadow":false,"space":0.0,"hasNoneStyle":false},
"diagonalDown":{"lineStyle":"None","lineWidth":0.0,"shadow":false,"space":0.0,"hasNoneStyle":false},
"diagonalUp":{"lineStyle":"None","lineWidth":0.0,"shadow":false,"space":0.0,"hasNoneStyle":false}},
"title":null,"description":null,"tableFormat":{"allowAutoFit":true,"leftIndent":0.0,"tableAlignment":"Left",
"preferredWidthType":"Auto","borders":{"left":{"lineStyle":"Single","lineWidth":0.5,"shadow":false,
"space":0.0,"hasNoneStyle":false},
"right":{"lineStyle":"Single","lineWidth":0.5,"shadow":false,"space":0.0,"hasNoneStyle":false},
"top":{"lineStyle":"Single","lineWidth":0.5,"shadow":false,"space":0.0,"hasNoneStyle":false},
"bottom":{"lineStyle":"Single","lineWidth":0.5,"shadow":false,"space":0.0,"hasNoneStyle":false},
"vertical":{"lineStyle":"Single","lineWidth":0.5,"shadow":false,"space":0.0,"hasNoneStyle":false},
"horizontal":{"lineStyle":"Single","line

```

```

Width":0.5,"shadow":false,"space":0.0,"hasNoneStyle":false},"diagonalDown":{
"lineStyle":"None","lineWidth":0.0,"shadow":false,"space":0.0,"hasNoneStyle"
:false},"diagonalUp":{"lineStyle":"None","lineWidth":0.0,"shadow":false,"spa
ce":0.0,"hasNoneStyle":false}},"bidi":false}},{"characterFormat":{"bold":tru
e,"fontSize":14.0,"boldBidi":true,"fontSizeBidi":14.0},"paragraphFormat":{"l
ineSpacing":32.0,"lineSpacingType":"Exactly","styleName":"Normal"},"inlines"
:[{"text":"Review
comments":"","characterFormat":{"bold":true,"fontSize":14.0,"boldBidi":true,"f
ontSizeBidi":14.0}}}],{"rows":[{"rowFormat":{"allowBreakAcrossPages":true,"i
sHeader":false,"height":20.0,"heightType":"AtLeast","borders":{"left":{"line
Style":"None","lineWidth":0.0,"shadow":false,"space":0.0,"hasNoneStyle":fals
e},"right":{"lineStyle":"None","lineWidth":0.0,"shadow":false,"space":0.0,"h
asNoneStyle":false},"top":{"lineStyle":"None","lineWidth":0.0,"shadow":false
,"space":0.0,"hasNoneStyle":false},"bottom":{"lineStyle":"None","lineWidth":
0.0,"shadow":false,"space":0.0,"hasNoneStyle":false},"vertical":{"lineStyle"
:"None","lineWidth":0.0,"shadow":false,"space":0.0,"hasNoneStyle":false},"ho
rizontal":{"lineStyle":"None","lineWidth":0.0,"shadow":false,"space":0.0,"ha
sNoneStyle":false},"diagonalDown":{"lineStyle":"None","lineWidth":0.0,"shado
w":false,"space":0.0,"hasNoneStyle":false},"diagonalUp":{"lineStyle":"None",
"lineWidth":0.0,"shadow":false,"space":0.0,"hasNoneStyle":false}}},"cells":[
{"blocks":[{"paragraphFormat":{"styleName":"Normal"},"inlines":[{"editRangeId
":"1373703080","columnFirst":0,"columnLast":0,"user":"manager@mycompany.com
"},"{"text":"Enter the
comments"},"{"editRangeId":"1373703080","editableRangeStart":{"editRangeId":"
1373703080","columnFirst":0,"columnLast":0,"user":"manager@mycompany.com"}}]
}],{"cellFormat":{"columnSpan":1,"rowSpan":1,"preferredWidth":467.5,"preferre
dWidthType":"Point","verticalAlignment":"Center","isSamePaddingAsTable":true
,"borders":{"left":{"lineStyle":"None","lineWidth":0.0,"shadow":false,"space
":0.0,"hasNoneStyle":false},"right":{"lineStyle":"None","lineWidth":0.0,"sha
dow":false,"space":0.0,"hasNoneStyle":false},"top":{"lineStyle":"None","line
Width":0.0,"shadow":false,"space":0.0,"hasNoneStyle":false},"bottom":{"lineS
tyle":"None","lineWidth":0.0,"shadow":false,"space":0.0,"hasNoneStyle":false
},"vertical":{"lineStyle":"None","lineWidth":0.0,"shadow":false,"space":0.0,
"hasNoneStyle":false},"horizontal":{"lineStyle":"None","lineWidth":0.0,"shad
ow":false,"space":0.0,"hasNoneStyle":false},"diagonalDown":{"lineStyle":"Non
e","lineWidth":0.0,"shadow":false,"space":0.0,"hasNoneStyle":false},"diagona
lUp":{"lineStyle":"None","lineWidth":0.0,"shadow":false,"space":0.0,"hasNone
Style":false}}}}}],{"title":null,"description":null,"tableFormat":{"allowAut
oFit":true,"leftIndent":0.0,"tableAlignment":"Left","preferredWidthType":"Au
to","borders":{"left":{"lineStyle":"Single","lineWidth":0.5,"shadow":false,"
space":0.0,"hasNoneStyle":false},"right":{"lineStyle":"Single","lineWidth":0
.5,"shadow":false,"space":0.0,"hasNoneStyle":false},"top":{"lineStyle":"Sing
le","lineWidth":0.5,"shadow":false,"space":0.0,"hasNoneStyle":false},"bottom
":{"lineStyle":"Single","lineWidth":0.5,"shadow":false,"space":0.0,"hasNoneS
tyle":false},"vertical":{"lineStyle":"Single","lineWidth":0.5,"shadow":false
,"space":0.0,"hasNoneStyle":false},"horizontal":{"lineStyle":"Single","lineW
idth":0.5,"shadow":false,"space":0.0,"hasNoneStyle":false},"diagonalDown":{"
lineStyle":"None","lineWidth":0.0,"shadow":false,"space":0.0,"hasNoneStyle":
false},"diagonalUp":{"lineStyle":"None","lineWidth":0.0,"shadow":false,"spac
e":0.0,"hasNoneStyle":false}}},"bidi":false}},{"paragraphFormat":{"styleName
":"Normal"},"inlines":[]},"headersFooters":{"header":{"blocks":[{"paragraphF
ormat":{"styleName":"Header"},"inlines":[{"text":"Employee's Details
"}]}]}},{"sectionFormat":{"headerDistance":36.0,"footerDistance":36.0,"pageWi
dth":612.0,"pageHeight":792.0,"leftMargin":72.0,"rightMargin":72.0,"topMargi
n":72.0,"bottomMargin":72.0,"differentFirstPage":false,"differentOddAndEvenP
ages":false,"bidi":false}}],"characterFormat":{"fontSize":11.0,"fontFamily":
"Calibri","fontSizeBidi":11.0,"fontFamilyBidi":"Calibri"},"paragraphFormat":

```



```

{"afterSpacing":8.0,"lineSpacing":1.0791666507720947,"lineSpacingType":"Multiple"},
"background":{"color":"#FFFFFFF"},
"styles":[{"type":"Paragraph","name":"Normal","next":"Normal"},
{"type":"Character","name":"Default Paragraph Font"},
{"type":"Paragraph","name":"List Paragraph","basedOn":"Normal",
"paragraphFormat":{"leftIndent":36.0,"contextualSpacing":true}},
{"type":"Paragraph","name":"Header","basedOn":"Normal","next":"Normal",
"link":"Header Char","paragraphFormat":{"afterSpacing":0.0,
"lineSpacing":1.0,"lineSpacingType":"Multiple",
"tabs":[{"tabJustification":"Center","position":234.0,
"tabLeader":"None","deletePosition":0.0},
{"tabJustification":"Right","position":468.0,
"tabLeader":"None","deletePosition":0.0}]}]},
{"type":"Character","name":"Header Char","basedOn":"Default Paragraph Font"},
{"type":"Paragraph","name":"Footer Char","basedOn":"Normal",
"link":"Footer Char","paragraphFormat":{"afterSpacing":0.0,
"lineSpacing":1.0,"lineSpacingType":"Multiple",
"tabs":[{"tabJustification":"Center","position":234.0,
"tabLeader":"None","deletePosition":0.0},
{"tabJustification":"Right","position":468.0,
"tabLeader":"None","deletePosition":0.0}]}]},
{"type":"Character","name":"Footer Char","basedOn":"Default Paragraph Font"}],
"defaultTabWidth":36.0,"formatting":false,"protectionType":"ReadOnly",
"enforcement":true,"hashValue":"TQGuJuLceQCe234Ygx4q6NFgHPRMfilhjFTojyKzbQOkwk+ckEM9CjNidkiUhsR/e/7sfMxO4sbPcg/DBzztMg==",
"saltValue":"FXbkr1RtDIIIZfwLM71dMg=="}`;

var editor =
this.$refs.doceditcontainer.ej2Instances.documentEditor;
//Open the default document in Document Editor.
editor.open(sfdt);
this.$refs.doceditcontainer.ej2Instances.serviceUrl =
'https://ej2services.syncfusion.com/production/web-services/api/documenteditor/';
},
provide: {
DocumentEditorContainer: [Toolbar]
}
}
</script>
<style>
@import "../../../node_modules/@syncfusion/ej2-vue-documenteditor/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/document-editor/getting-started-cs6" %}

See Also

- [How to protect the document in form filling mode](#)
- [How to protect the document in comments only mode](#)
- [How to protect the document in track changes only mode](#)

### Spell check in Vue Document editor component

Document editor supports performing spell checking for any input text. You can perform spell checking for the text in Document Editor and it will provide suggestions for the mis-spelled words through dialog and in context menu. Document editor's spell checker is compatible with [hunspell dictionary files](#).



```
<template>
<div id="app">
<ejs-documenteditor ref="documenteditor" id="container_1" height="370px" style="width: 100%;"
:enableSpellCheck='true'></ejs-documenteditor>
</div>
</template>
<script>
import Vue from 'vue'
import { DocumentEditorPlugin } from '@syncfusion/ej2-vue-documenteditor';
Vue.use(DocumentEditorPlugin);
export default {
data: function() {
return {
};
},
mounted: function() {
//Set language id to map dictionary.
this.$refs.documenteditor.ej2Instances.spellChecker.languageID = 1033;
this.$refs.documenteditor.ej2Instances.spellChecker.removeUnderline = false;
this.$refs.documenteditor.ej2Instances.spellChecker.allowSpellCheckAndSuggestion = true;
}
}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-vue-documenteditor/styles/material.css";
</style>
`
```

## Features

- Supports context menu suggestions.
- Provides built-in options to Ignore, Ignore All, Change, Change All for error words in spell checker dialog.

## Enable SpellCheck

To enable spell check in Document Editor, set [enableSpellCheck](#) property as `true` and then configure [SpellCheckSettings](#).

### Disable SpellCheck

To disable spell check in Document Editor, set [enableSpellCheck](#) property as `false` or remove [enableSpellCheck](#) property initialization code. The default value of this property is `false`.

### Spell check settings

#### *Remove Underline*

By default, mis-spelled words are marked with squiggly line. You can also disable this behavior by enabling the [removeUnderline](#) API and now, the squiggly lines will never be rendered for mis-spelled words.

```
`ts
```

```
this.$refs.documenteditor.ej2Instances.spellChecker.removeUnderline = false;
```

```
,
```

#### *AllowSpellCheckAndSuggestion*

By default, on performing spell check in Document Editor, both spelling and suggestions of the mis-spelled words will be retrieved, and this mis-spelled words can be corrected through context menu suggestions. You can modify this behavior using the [allowSpellCheckAndSuggestion](#) API, which will perform only spell check.

```
`ts
```

```
this.$refs.documenteditor.ej2Instances.spellChecker.allowSpellCheckAndSuggestion = false;
```

```
,
```

#### *LanguageID*

Document Editor provides multi-language spell check support. You can add as many languages (dictionaries) in the server-side and to use that language for spell checking in Document Editor, it must be matched with [languageID](#) you pass in the Document Editor.

```
`ts
```

```
this.$refs.documenteditor.ej2Instances.spellChecker.languageID = 1033 //LCID of "en-us";
```

```
,
```

#### *EnableOptimizedSpellCheck*

Document editor provides option to spellcheck page by page when loading the documents. The default value of this property is `false`, so when opening the document spellcheck web API will be called for each word in the document. To optimize the frequency of spellcheck web API calls, you can enable this property.

The following code example illustrates how to enable optimized spell checking.

```
`ts
```

```
this.container.documentEditor.spellChecker.enableOptimizedSpellCheck = true;
```

```
,
```

#### *Spell check dictionary cache*

Starting from `v20.1.0.xx`, we have optimized the performance and memory usage of spell checker by adding a static method to initialize the dictionaries with specified cache count.

By default, the spell checker holds only one language dictionary in memory. If you want to hold multiple dictionaries in memory, you need to set the cache limit by using `InitializeDictionaries` method as in the below example.

```
`c#
List<DictionaryData> spellDictCollection = new List<DictionaryData>();
string personalDictPath = string.Empty;
int cacheCount = 2;
// Initialize dictionaries
SpellChecker.InitializeDictionaries(spellDictCollection, personalDictPath, cacheCount);
`
```

If dictionaries are initialized using `InitializeDictionaries` method, then we should use default constructor of the `SpellChecker` to check spelling and get suggestion as in the below example code, it will prevent reinitialization of already loaded dictionaries.

```
`c#
public string SpellCheck([FromBody] SpellCheckJsonData spellChecker)
{
    try
    {
        SpellChecker spellCheck = new SpellChecker();
        spellCheck.GetSuggestions(spellChecker.LanguageID, spellChecker.TexttoCheck,
            spellChecker.CheckSpelling, spellChecker.CheckSuggestion, spellChecker.AddWord);
        return Newtonsoft.Json.JsonConvert.SerializeObject(spellCheck);
    }
    catch
    {
        return "{\"SpellCollection\":[],\"HasSpellingError\":false,\"Suggestions\":null}";
    }
}
`
```

Previously on every `SpellChecker.GetSuggestion()` method call, the `.aff` and dictionary data will be parsed to generate suggestion for miss spelled word. But, starting from v20.1.0.xx, the `.aff` and dictionary data will be parsed only for the first time alone while calling `SpellChecker.GetSuggestion()` method.

### Add new root word and possible words to dictionary

If you find any root word is missing in the dictionary file, then you can add that new root word and the rule to form the possible words to dictionary file using `AddNewWord` API in the server-side Spell check library.

Note:

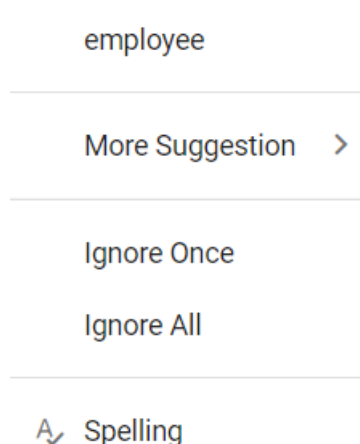
1. The rules are framed automatically using the root word, the possible words and affix file.
2. If you pass null for the parameters `affPath` and `possibleWords`, then it will add a single root word to dictionary.
3. This API is included starting from `v20.2.0.xx`.

The following code example demonstrates how to add a new root word to the dictionary along with the rule to form the possible words.

```
`c#
SpellChecker spellChecker = new SpellChecker();
// Adds the specified new root word to the dictionary along with the rule to form the possible words.
spellChecker.AddNewWord("en.dic","en.aff", "construct", new string[] { "constructs", "reconstruct",
"constructed", "constructive" });
`
```

### Context menu

Right click on error word to open the context menu with spell check options. Please see below screenshot for your reference.



### Suggestions

Context menu shows the suggestions for mis-spelled words. By clicking on the required word from suggestion, the error word gets replaced automatically.

### Add To Dictionary

Using this option, you can add the current word to the dictionary. So that the spell checker does not consider that word as error in future.

### [Ignore Once and Ignore All](#)

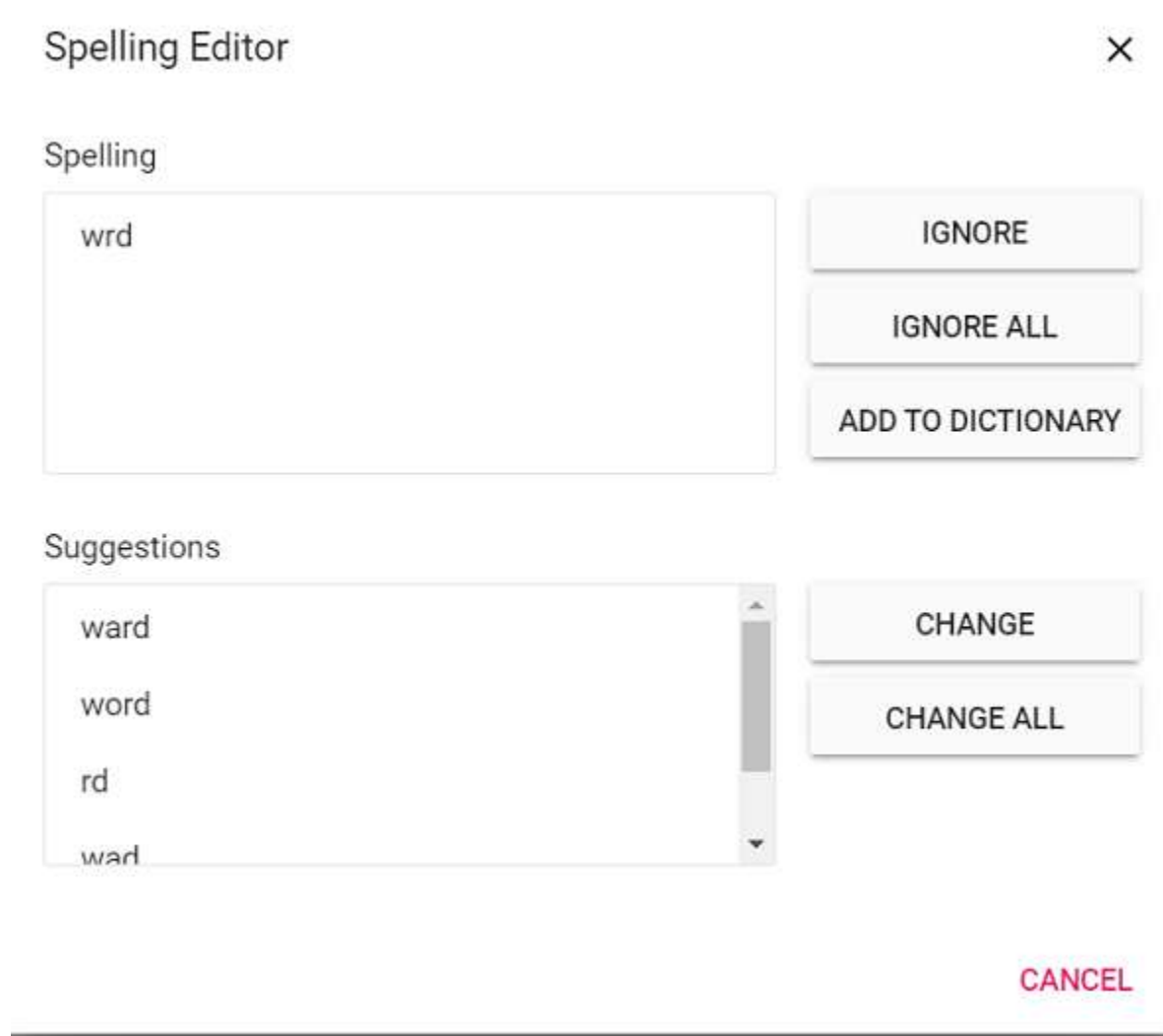
If you do not wish to add the word to dictionary and do not want to show error, use Ignore Once or Ignore All options.

Ignore: ignore only the current occurrence of a word from error.

Ignore All: ignore all occurrence of a word from error in the entire document.

### [Spelling](#)

Using this option, you can open spell check dialog. Please see below screenshot for your reference.



- Refer to the [Spell checker](#) link for configuring spell checker in server-side.

### [Global local in Vue Document editor component](#)

#### Localization

The Localization library allows you to localize default text content of the Document Editor. The document editor component has static text on some features (like find & replace, context-menu,

dialogs) that can be changed to other cultures (Arabic, Deutsch, French, etc.) by defining the locale value and translation object. Please refer the sample link [RTL](#)

Note: Please refer the [Locale](#).

### Document Editor

The following list of properties and its values are used in the document editor.

Locale keywords |Text

New | New

Open | Open

Undo | Undo

Redo | Redo

Image | Image

Table | Table

Link | Link

Bookmark | Bookmark

Table of Contents | Table of Contents

HEADING - - - - 1 | HEADING - - - - 1

HEADING - - - - 2 | HEADING - - - - 2

HEADING - - - - 3 | HEADING - - - - 3

Header | Header

Footer | Footer

Page Setup | Page Setup

Page Number | Page Number

Break | Break

Find | Find

Local Clipboard | Local Clipboard

Restrict Editing | Restrict Editing

Upload from computer | Upload from computer

By URL | By URL

Page Break | Page Break

Section Break | Section Break

Header And Footer | Header & Footer

Options | Options

Levels | Levels

Different First Page | Different First Page

Different header and footer for odd and even pages | Different header and footer for odd and even pages.

Different Odd And Even Pages | Different Odd & Even Pages

Different header and footer for first page | Different header and footer for first page.

Position | Position

Header from Top | Header from Top

Footer from Bottom | Footer from Bottom

Distance from top of the page to top of the header | Distance from top of the page to top of the header.

Distance from bottom of the page to bottom of the footer | Distance from bottom of the page to bottom of the footer.

Aspect ratio | Aspect ratio

W | W

H | H

Width | Width

Height | Height

Text | Text

Paragraph | Paragraph

Fill | Fill

Fill color | Fill color

Border Style | Border Style

Outside borders | Outside borders

All borders | All borders

Inside borders | Inside borders

Left border | Left border

Inside vertical border | Inside vertical border

Right border | Right border

Top border | Top border

Inside horizontal border | Inside horizontal border

Bottom border | Bottom border

Border color | Border color

Border width | Border width

Cell | Cell

Merge cells | Merge cells

Insert Or Delete | Insert / Delete

Insert columns to the left | Insert columns to the left

Insert columns to the right | Insert columns to the right

Insert rows above | Insert rows above

Insert rows below | Insert rows below

Delete rows | Delete rows

Delete columns | Delete columns

Cell Margin | Cell Margin

Top | Top

Bottom | Bottom

Left | Left

Right | Right

Align Text | Align Text

Align top | Align top

Align bottom | Align bottom

Align center | Align center

Number of heading or outline levels to be shown in table of contents | Number of heading or outline levels to be shown in table of contents.

Show page numbers | Show page numbers

Show page numbers in table of contents | Show page numbers in table of contents.

Right align page numbers | Right align page numbers

Right align page numbers in table of contents | Right align page numbers in table of contents.

Use hyperlinks | Use hyperlinks

Use hyperlinks instead of page numbers | Use hyperlinks instead of page numbers.

Font | Font

Font Size | Font Size

Font color | Font color

Text highlight color | Text highlight color

Clear all formatting | Clear all formatting

Bold Tooltip | Bold (Ctrl+B)

Italic Tooltip | Italic (Ctrl+I)

Underline Tooltip | Underline (Ctrl+U)

Strikethrough | Strikethrough

Superscript Tooltip | Superscript (Ctrl+Shift++)



Subscript Tooltip | Subscript (Ctrl+=)

Align left Tooltip | Align left (Ctrl+L)

Center Tooltip | Center (Ctrl+E)

Align right Tooltip | Align right (Ctrl+R)

Justify Tooltip | Justify (Ctrl+J)

Decrease indent | Decrease indent

Increase indent | Increase indent

Line spacing | Line spacing

Bullets | Bullets

Numbering | Numbering

Styles | Styles

Manage Styles | Manage Styles

Page | Page

of | of

Fit one page | Fit one page

Spell Check | Spell Check

Underline errors | Underline errors

Fit page width | Fit page width

Update | Update

Cancel | Cancel

Insert | Insert

No Border | No Border

Create a new document | Create a new document.

Open a document | Open a document.

Undo Tooltip | Undo the last operation (Ctrl+Z).

Redo Tooltip | Redo the last operation (Ctrl+Y).

Insert inline picture from a file | Insert inline picture from a file.

Insert a table into the document | Insert a table into the document

Create Hyperlink | Create a link in your document for quick access to web pages and files (Ctrl+K).

Insert a bookmark in a specific place in this document | Insert a bookmark in a specific place in this document.

Provide an overview of your document by adding a table of contents | Provide an overview of your document by adding a table of contents.

Add or edit the header | Add or edit the header.

Add or edit the footer | Add or edit the footer.

Open the page setup dialog | Open the page setup dialog.

Add page numbers | Add page numbers.

Find Text | Find text in the document (Ctrl+F).

Toggle between the internal clipboard and system clipboard | Toggle between the internal clipboard and system clipboard.</br>Access to system clipboard through script is denied due to browsers security policy. Instead, </br> 1. You can enable internal clipboard to cut, copy and paste within the component.</br> 2. You can use the keyboard shortcuts (Ctrl+X, Ctrl+C and Ctrl+V) to cut, copy and paste with system clipboard.

Current Page Number | The current page number in the document. Click or tap to navigate specific page.

Read only | Read only

Protections | Protections

Error in establishing connection with web server | Error in establishing connection with web server

Single | Single

Double | Double

New comment | New comment

Comments | Comments

Print layout | Print layout

Web layout | Web layout

Text Form | Text Form

Check Box | Check Box

DropDown | Drop-Down

Update Fields | Update Fields

Update cross reference fields | Update cross reference fields

Hide properties pane | Hide properties pane

Show properties pane | Show properties pane

[Color Picker](#)

The following list of properties and its values are used in the color picker.

Locale keywords |Text

Apply | Apply

Cancel | Cancel

ModeSwitcher | Switch Mode

## Notes in Vue Document editor component

DocumentEditorContainer component provides support for inserting footnotes and endnotes through the in-built toolbar. Refer to the following screenshot.



The Footnotes and endnotes are both ways of adding extra bits of information to your writing outside of the main text. You can use footnotes and endnotes to add side comments to your work or to place other publications like books, articles, or websites.

### Insert footnotes

Document Editor exposes an API to insert footnotes at cursor position programmatically or can be inserted to the end of selected text.

,

```
<template>
<div id="app">
<div>
<button v-on:click='Insertfootnote' >Insert footnote</button>
</div>

<ejs-documenteditor id="container6" height="370px" style="width: 100%;" :serviceUrl='serviceUrl'
:isReadOnly='false' :enablePrint='true' :enableSfdtExport='true' :enableSelection='true'
:enableContextMenu='true' :enableSearch='true' :enableOptionsPane='true' :enableWordExport='true'
:enableTextExport='true' :enableEditor='true' :enableImageResizer='true' :enableEditorHistory='true'
:enableHyperlinkDialog='true' :enableTableDialog='true' :enableBookmarkDialog='true'
:enableTableOfContentsDialog='true' :enablePageSetupDialog='true' :enableStyleDialog='true'
:enableListDialog='true' :enableParagraphDialog='true' :enableFontDialog='true'
:enableTablePropertiesDialog='true' :enableBordersAndShadingDialog='true'
:enableTableOptionsDialog='true'></ejs-documenteditor>

</div>
</template>
<script>
import Vue from 'vue'

import { DocumentEditorPlugin, DocumentEditorComponent, Print, SfdtExport, WordExport, TextExport,
Selection, Search, Editor, ImageResizer, EditorHistory, ContextMenu, OptionsPane, HyperlinkDialog,
TableDialog, BookmarkDialog, TableOfContentsDialog, PageSetupDialog, StyleDialog, ListDialog,
ParagraphDialog, BulletsAndNumberingDialog, FontDialog, TablePropertiesDialog,
BordersAndShadingDialog, TableOptionsDialog, CellOptionsDialog, StylesDialog } from '@syncfusion/ej2-
vue-documenteditor';

Vue.use(DocumentEditorPlugin);
```

```

export default {
  data() {
    return {
      serviceUrl: 'https://ej2services.syncfusion.com/production/web-services/api/documenteditor/'
    };
  },
  provide: {
    DocumentEditor: [Print, SfdtExport, WordExport, TextExport, Selection, Search, Editor, ImageResizer,
      EditorHistory, ContextMenu, OptionsPane, HyperlinkDialog, TableDialog, BookmarkDialog,
      TableOfContentsDialog, PageSetupDialog, StyleDialog, ListDialog, ParagraphDialog,
      BulletsAndNumberingDialog, FontDialog, TablePropertiesDialog, BordersAndShadingDialog,
      TableOptionsDialog, CellOptionsDialog, StylesDialog]
  },
  methods: {
    Insertfootnote: function() {
      this.$refs.documenteditor.editor.insertFootnote();
    }
  }
}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-vue-documenteditor/styles/material.css";
</style>

```

### Insert endnotes

Document Editor exposes an API to insert endnotes at cursor position programmatically or can be inserted to the end of selected text.

```

<template>
<div id="app">
<div>
<button v-on:click='Insertendnote' >Insert endnote</button>
</div>

<ejs-documenteditor id="container6" style="width: 100%;height: 100%;" :serviceUrl='serviceUrl'
:isReadOnly='false' :enablePrint='true' :enableSfdtExport='true' :enableSelection='true'
:enableContextMenu='true' :enableSearch='true' :enableOptionsPane='true' :enableWordExport='true'

```

```
:enableTextExport='true' :enableEditor='true' :enableImageResizer='true' :enableEditorHistory='true'
:enableHyperlinkDialog='true' :enableTableDialog='true' :enableBookmarkDialog='true'
:enableTableOfContentsDialog='true' :enablePageSetupDialog='true' :enableStyleDialog='true'
:enableListDialog='true' :enableParagraphDialog='true' :enableFontDialog='true'
:enableTablePropertiesDialog='true' :enableBordersAndShadingDialog='true'
:enableTableOptionsDialog='true'></ejs-documenteditor>

</div>

</template>

<script>

import Vue from 'vue'

import { DocumentEditorPlugin, DocumentEditorComponent, Print, SfdtExport, WordExport, TextExport,
Selection, Search, Editor, ImageResizer, EditorHistory, ContextMenu, OptionsPane, HyperlinkDialog,
TableDialog, BookmarkDialog, TableOfContentsDialog, PageSetupDialog, StyleDialog, ListDialog,
ParagraphDialog, BulletsAndNumberingDialog, FontDialog, TablePropertiesDialog,
BordersAndShadingDialog, TableOptionsDialog, CellOptionsDialog, StylesDialog } from '@syncfusion/ej2-
vue-documenteditor';

Vue.use(DocumentEditorPlugin);

export default {
  data() {
    return {
      serviceUrl:'https://ej2services.syncfusion.com/production/web-services/api/documenteditor/'
    };
  },
  provide: {
    DocumentEditor: [Print, SfdtExport, WordExport, TextExport, Selection, Search, Editor, ImageResizer,
    EditorHistory, ContextMenu, OptionsPane, HyperlinkDialog, TableDialog, BookmarkDialog,
    TableOfContentsDialog, PageSetupDialog, StyleDialog, ListDialog, ParagraphDialog,
    BulletsAndNumberingDialog, FontDialog, TablePropertiesDialog, BordersAndShadingDialog,
    TableOptionsDialog, CellOptionsDialog, StylesDialog]
  },
  methods: {
    Insertendnote: function() {
      this.$refs.documenteditor.editor.insertEndnote();
    }
  }
}

</script>
```

```
<style>
```

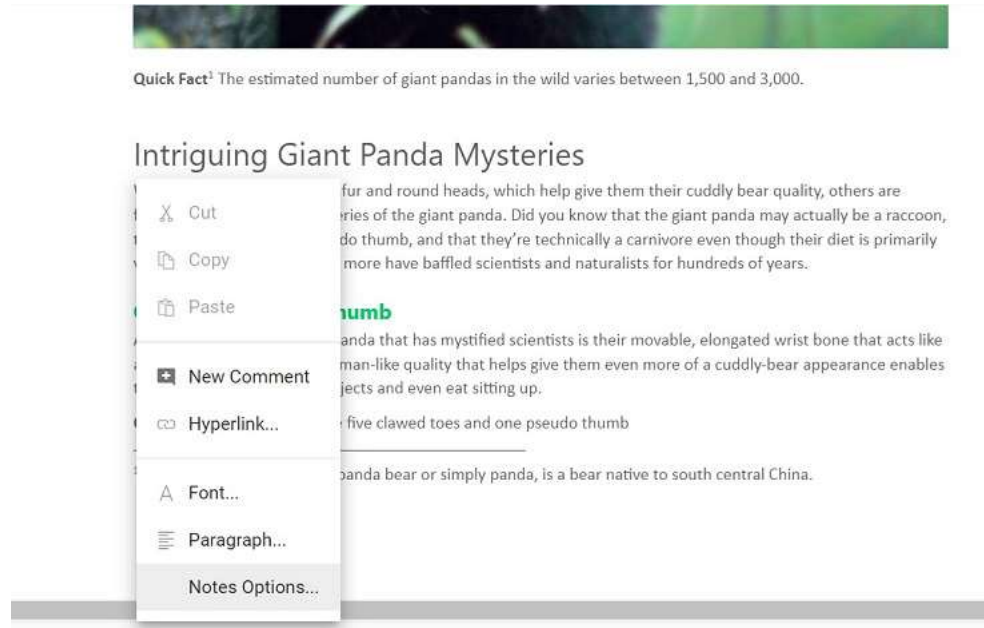
```
@import "../node_modules/@syncfusion/ej2-vue-documenteditor/styles/material.css";
```

```
</style>
```

```
,
```

### Update or edit footnotes and endnotes

You can update or edit the footnotes and endnotes using the built-in context menu shown up by right-clicking it. The footnote endnote dialog box popup and you can customize the number format and start at. Refer to the following screenshot.



### How To

Override the keyboard shortcuts in Vue Document editor component

Document Editor triggers the [keyDown](#) event every time when any key is entered and provides an instance of [DocumentEditorKeyDownEventArgs](#). You can use the [isHandled](#) property to override the keyboard shortcut behavior.

#### Preventing default keyboard shortcut

The following code shows how to prevent the **CTRL + C** keyboard shortcut for copying selected content in Document Editor.

#### APP.VUE

```
<template>
  <div style="height:330px">
    <ejs-documenteditor ref="documenteditor" :height="height"
    style="width: 100%;display:block" :isReadOnly='false'
    :enableSelection='true' :enableSfdtExport='true' :enableEditor='true' v-
    bind:keyDown='onKeyDown'></ejs-documenteditor>
  </div>
</template>
<script>
  import Vue from 'vue'
```

```
import { DocumentEditorPlugin, Selection, Editor, SfdtExport } from
'@syncfusion/ej2-vue-documenteditor';
Vue.use(DocumentEditorPlugin);
export default {
  data: function() {
    return {
      height: '370px'
    };
  },
  provide: {
    DocumentEditor : [SfdtExport, Selection, Editor]
  }
  methods: {
    onKeyDown: function(args) {
      let keyCode: number = args.event.which ||
args.event.keyCode;
      let isCtrlKey: boolean = (args.event.ctrlKey ||
args.event.metaKey) ? true : ((keyCode === 17) ? true : false);
      //67 is the character code for 'C'
      if (isCtrlKey && keyCode === 67) {
        //To prevent copy operation set isHandled to true
        args.isHandled = true;
      }
    }
  }
}
</script>
<style>
  @import "../node_modules/@syncfusion/ej2-vue-
documenteditor/styles/material.css";
</style>
```

{% previewsample "page.domainurl/code-snippet/document-editor/export-cs6" %}

### Override or define the keyboard shortcut

Override or define a new keyboard shortcut behavior instead of preventing the keyboard shortcut.

For example, **Ctrl + S** keyboard shortcut saves the document in SFDT format by default, and there is no behavior for **Ctrl + Alt + S**. The following code demonstrates how to override the **Ctrl + S** shortcut to save a document in DOCX format and define **Ctrl + Alt + S** to save the document in SFDT format.

### APP.VUE

```
<template>
  <div>
    <ejs-documenteditor ref="documenteditor" :height="height"
style="width: 100%;display:block" :isReadOnly='false'
:enableSelection='true' :enableSfdtExport='true' :enableWordExport=true
:enableEditor='true' v-bind:keyDown='onKeyDown'></ejs-documenteditor>
  </div>
</template>
<script>
  import Vue from 'vue'
  import { DocumentEditorPlugin, Selection, Editor,
DocumentEditorKeyDownEventArgs, SfdtExport, WordExport } from
'@syncfusion/ej2-vue-documenteditor';
```

```

Vue.use(DocumentEditorPlugin);
export default {
  data: function() {
    return {
      height: '370px'
    };
  },
  provide: {
    DocumentEditor : [SfdtExport, Selection, Editor, WordExport]
  },
  methods: {
    onKeyDown: function(args) {
      let keyCode: number = args.event.which ||
args.event.keyCode;
      let isCtrlKey: boolean = (args.event.ctrlKey ||
args.event.metaKey) ? true : ((keyCode === 17) ? true : false);
      let isAltKey: boolean = args.event.altKey ?
args.event.altKey : ((keyCode === 18) ? true : false);
      // 83 is the character code for 'S'
      if (isCtrlKey && !isAltKey && keyCode === 83) {
        //To prevent default save operation, set the isHandled
property to true
        args.isHandled = true;
        this.$refs.documenteditor.save('sample', 'Docx');
        args.event.preventDefault();
      } else if (isCtrlKey && isAltKey && keyCode === 83) {
        this.$refs.documenteditor.save('sample', 'Sfdt');
      }
    }
  }
}
</script>
<style>
  @import "../node_modules/@syncfusion/ej2-vue-
documenteditor/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/document-editor/export-cs7" %}

Customize context menu in Vue Document editor component

[How to customize context menu in Document Editor](#)

Document Editor allows you to add custom option in context menu. It can be achieved by using the [addCustomMenu\(\)](#) method and custom action is defined using the [customContextMenuSelect](#)

Add Custom Option

The following code shows how to add custom option in context menu.

,

<template>

<div id="app">

<ejs-documenteditorcontainer ref='container' :serviceUrl='serviceUrl' v-on:created="onCreated"  
height="590px" id='container' :enableToolBar='true'></ejs-documenteditorcontainer>



```
</div>
</template>
<script>
import Vue from 'vue';
import {
  DocumentEditorContainerPlugin,
  DocumentEditorContainerComponent,
  Toolbar,
} from '@syncfusion/ej2-vue-documenteditor';
Vue.use(DocumentEditorContainerPlugin);
export default {
  data() {
    return {
      serviceUrl:
        'https://ej2services.syncfusion.com/production/web-services/api/documenteditor/',
    };
  },
  provide: {
    //Inject require modules.
    DocumentEditorContainer: [Toolbar],
  },
  methods: {
    onCreated: function () {
      // creating Custom Options
      let menuItems = [
        {
          text: 'Search In Google',
          id: 'searchingoogle',
          iconCss: 'e-icons e-de-ctnr-find',
        },
      ];
      // adding Custom Options
      this.$refs.container.ej2Instances.documentEditor.contextMenu.addCustomMenu(
```

```

menuItems,
false
);
// custom Options Select Event
this.$refs.container.ej2Instances.documentEditor.customContextMenuSelect =
(args) => {
// custom Options Functionality
let id =
this.$refs.container.ej2Instances.documentEditor.element.id;
switch (args.id) {
case id + 'searchingoogle':
// To get the selected content as plain text
let searchContent =
this.$refs.container.ej2Instances.documentEditor.selection
.text;
if (
!this.$refs.container.ej2Instances.documentEditor.selection
.isEmpty &&
/\S/.test(searchContent)
) {
window.open('http://google.com/search?q=' + searchContent);
}
break;
}
};
},
},
};
</script>
`

```

#### [Customize custom option in context menu](#)

Document Editor allows you to customize the added custom option and also to hide/show default context menu.

### Hide default context menu items

Using [addCustomMenu\(\)](#) method, you can hide the default context menu. By setting second parameter as true.

The following code shows how to hide default context menu and add custom option in context menu.

```
<template>
<div id="app">
<ejs-documenteditorcontainer ref='container' :serviceUrl='serviceUrl' v-on:created="onCreated"
height="590px" id='container' :enableToolbar='true'></ejs-documenteditorcontainer>
</div>
</template>
<script>
import Vue from 'vue';
import {
DocumentEditorContainerPlugin,
DocumentEditorContainerComponent,
Toolbar,
} from '@syncfusion/ej2-vue-documenteditor';
Vue.use(DocumentEditorContainerPlugin);
export default {
data() {
return {
serviceUrl:
'https://ej2services.syncfusion.com/production/web-services/api/documenteditor/',
};
},
provide: {
//Inject require modules.
DocumentEditorContainer: [Toolbar],
},
methods: {
onCreated: function () {
// creating Custom Options
let menuItems = [
```

```
{
text: 'Search In Google',
id: 'searchingoogle',
iconCss: 'e-icons e-de-ctnr-find',
},
];
// adding Custom Options
this.$refs.container.ej2Instances.documentEditor.contextMenu.addCustomMenu( menuItems,true);
}
}
};
</script>
`
```

Customize added context menu items

The following code shows how to hide/show added custom option in context menu using the [customContextMenuBeforeOpen](#).

```
`
<template>
<div id="app">
<ejs-documenteditorcontainer ref='container' :serviceUrl='serviceUrl' v-on:created="onCreated"
height="590px" id='container' :enableToolbar='true'></ejs-documenteditorcontainer>
</div>
</template>
<script>
import Vue from 'vue';
import {
DocumentEditorContainerPlugin,
DocumentEditorContainerComponent,
Toolbar,
} from '@syncfusion/ej2-vue-documenteditor';
Vue.use(DocumentEditorContainerPlugin);
export default {
data() {
return {
```

```
serviceUrl:
'https://ej2services.syncfusion.com/production/web-services/api/documenteditor/',
};
},
provide: {
//Inject require modules.
DocumentEditorContainer: [Toolbar],
},
methods: {
onCreated: function () {
// creating Custom Options
let menuItems = [
{
text: 'Search In Google',
id: 'searchingoogle',
iconCss: 'e-icons e-de-ctnr-find',
},
];
// adding Custom Options
this.$refs.container.ej2Instances.documentEditor.contextMenu.addCustomMenu( menuItems,false);
// custom Options Select Event
this.$refs.container.ej2Instances.documentEditor.customContextMenuSelect =
(args) => {
// custom Options Functionality
let id =
this.$refs.container.ej2Instances.documentEditor.element.id;
switch (args.id) {
case id + 'searchingoogle':
// To get the selected content as plain text
let searchContent =
this.$refs.container.ej2Instances.documentEditor.selection
.text;
if (
```

```

!this.$refs.container.ej2Instances.documentEditor.selection
.isEmpty &&
/\S/.test(searchContent)
){
window.open('http://google.com/search?q=' + searchContent);
}
break;
}
};
// custom options hide/show functionality
this.$refs.container.ej2Instances.documentEditor.customContextMenuBeforeOpen = (args) => {
let search = document.getElementById(args.ids[0]);
search.style.display = 'none';
let searchContent = this.$refs.container.ej2Instances.documentEditor.selection.text;
if (!this.$refs.container.ej2Instances.documentEditor.selection.isEmpty && /\S/.test(searchContent)) {
search.style.display = 'block';
}
};
},
},
};
</script>
`

```

The following is the output of custom context menu with customization.

### **APP.VUE**

```

<template>
  <div id="app">
    <ejs-documenteditorcontainer ref='container' :serviceUrl='serviceUrl'
v-on:created="onCreated" height="590px" id='container'
:enableToolbar='true'></ejs-documenteditorcontainer>
  </div>
</template>
<script>
import Vue from 'vue';
import {
  DocumentEditorContainerPlugin,
  DocumentEditorContainerComponent,
  Toolbar,
} from '@syncfusion/ej2-vue-documenteditor';

```

```

Vue.use(DocumentEditorContainerPlugin);
export default {
  data() {
    return {
      serviceUrl:
        'https://ej2services.syncfusion.com/production/web-
services/api/documenteditor/',
    };
  },
  provide: {
    //Inject require modules.
    DocumentEditorContainer: [Toolbar],
  },
  methods: {
    onCreate: function () {
      // creating Custom Options
      let menuItems = [
        {
          text: 'Search In Google',
          id: 'search_in_google',
          iconCss: 'e-icons e-de-ctnr-find',
        },
      ],
      // adding Custom Options

      this.$refs.container.ej2Instances.documentEditor.contextMenu.addCustomMenu(
        menuItems, false);
      // custom Options Select Event

      this.$refs.container.ej2Instances.documentEditor.customContextMenuSelect =
        (args) => {
          // custom Options Functionality
          let id =
            this.$refs.container.ej2Instances.documentEditor.element.id;
          switch (args.id) {
            case id + 'search_in_google':
              // To get the selected content as plain text
              let searchContent =
                this.$refs.container.ej2Instances.documentEditor.selection
                  .text;
              if (
                !this.$refs.container.ej2Instances.documentEditor.selection
                  .isEmpty &&
                /\S/.test(searchContent)
              ) {
                window.open('http://google.com/search?q=' +
searchContent);
              }
              break;
            }
          };
          // custom options hide/show functionality

          this.$refs.container.ej2Instances.documentEditor.customContextMenuBeforeOpen
            = (args) => {
              let search = document.getElementById(args.ids[0]);

```

```

        search.style.display = 'none';
        let searchContent =
this.$refs.container.ej2Instances.documentEditor.selection.text;
        if
(!this.$refs.container.ej2Instances.documentEditor.selection.isEmpty &&
/\S/.test(searchContent)) {
            search.style.display = 'block';
        }
    };
},
},
};
</script>
<style>
    @import "../node_modules/@syncfusion/ej2-vue-
documenteditor/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/document-editor/customize-context-menu-cs1" %}

## Customize tool bar in Vue Document editor component

### *How to customize existing toolbar in DocumentEditorContainer*

Document Editor Container allows you to customize(add, show, hide, enable, and disable) existing items in a toolbar.

- Add - New items can be defined by [CustomToolbarItemModel](#) and with existing items in [toolbarItems](#) property. Newly added item click action can be defined in [toolbarclick](#).
- Show, Hide - Existing items can be shown or hidden using the [toolbarItems](#) property. Pre-defined toolbar items are available with [ToolbarItem](#).
- Enable, Disable - Toolbar items can be enabled or disable using [enableItems](#)

,

<template>

<div id="app">

<ejs-documenteditorcontainer ref="container" :toolbarItems='items' v-  
bind:toolbarClick='onToolbarClick' :enableToolbar='true'> </ejs-documenteditorcontainer>

</div>

</template>

<script>

import Vue from 'vue';

import { DocumentEditorContainerPlugin, DocumentEditorContainerComponent, Toolbar } from  
'@syncfusion/ej2-vue-documenteditor';

Vue.use(DocumentEditorContainerPlugin);

export default {

data(){



```

return {
  items: [
    {
      prefixIcon: "e-de-ctnr-lock",
      tooltipText: "Disable Image",
      text: "Disable Image",
      id: "Custom"
    },
    'Undo','Redo','Separator','Image','Table','Hyperlink','Bookmark','Comments','TableOfContents','Separator',
    'Header','Footer','PageSetup','PageNumber','Break',
    'Separator','Find','Separator','LocalClipboard','RestrictEditing']
  ],
  provide: {
    DocumentEditorContainer: [Toolbar]
  },
  methods: {
    onToolbarClick:function(args) {
      switch(args.item.id) {
        case 'Custom':
          //Disable image toolbar item.
          this.$refs.container.ej2Instances.toolbar.enableItems(4, false);
          break;
        }
      }
    }
  }
}
</script>

```

Note: Default value of `toolbarItems` is ['New', 'Open', 'Separator', 'Undo', 'Redo', 'Separator', 'Image', 'Table', 'Hyperlink', 'Bookmark', 'TableOfContents', 'Separator', 'Header', 'Footer', 'PageSetup', 'PageNumber', 'Break', 'InsertFootnote', 'InsertEndnote', 'Separator', 'Find', 'Separator', 'Comments', 'TrackChanges', 'Separator', 'LocalClipboard', 'RestrictEditing', 'Separator', 'FormFields', 'UpdateFields'].

Change document view in Vue Document editor component

*How to change the document view in DocumentEditor component*

Document Editor allows you to change the view to web layout and print using the [layoutType](#) property with the supported [LayoutType](#).

```
<ejs-documenteditor :layoutType='Continuous' id='container'></ejs-documenteditor>
```

Note: Default value of [layoutType](#) in Document Editor component is [Pages](#).

*How to change the document view in DocumentEditorContainer component*

Document Editor Container component allows you to change the view to web layout and print using the [layoutType](#) property with the supported [LayoutType](#).

```
<ejs-documenteditorcontainer :layoutType='Continuous' id='container'></ejs-documenteditorcontainer>
```

Note: Default value of [layoutType](#) in Document Editor Container component is [Pages](#).

Open default document in Vue Document editor component

In this article, we are going to see how to open a default document when Document Editor & Document Editor Container is initialized.

*Opening a default document in DocumentEditor*

By default, Document Editor will open blank document. You can use [open](#) API in Document Editor to open the sfdt content.

Document editor have [created](#) event which gets triggered once Document editor control created. So, if you want to open the document by default, you can use [open](#) and [created](#) API.

The following example illustrates how to open the default SFDT content once Document editor control gets created.

### **APP.VUE**

```
<template>
  <ejs-documenteditor ref="documenteditor" :height="height"
  :serviceUrl='serviceUrl'></ejs-documenteditor>
</template>
<script>
  import Vue from 'vue';
  import { DocumentEditorPlugin, DocumentEditorComponent } from
  '@syncfusion/ej2-vue-documenteditor';
  Vue.use(DocumentEditorPlugin);
  export default {
    data() {
      return {
        serviceUrl: 'https://ej2services.syncfusion.com/production/web-
        services/api/documenteditor/',
        height: '370px'
      }
    }
  }
</script>
```

```

    },
    provide: {
      DocumentEditor: []
    },
    mounted: function() {
      // load your default document here
      let sfdt = `{ "sections": [{ "sectionFormat": { "pageWidth": 612,
"pageHeight": 792, "leftMargin": 72, "rightMargin": 72, "topMargin": 72,
"bottomMargin": 72, "differentFirstPage": false, "differentOddAndEvenPages":
false, "headerDistance": 36, "footerDistance": 36, "bidi": false },
"blocks": [{ "paragraphFormat": { "afterSpacing": 30, "styleName": "Heading
1", "listFormat": { } }, "characterFormat": { }, "inlines": [{
"characterFormat": { }, "text": "Adventure Works Cycles" } ] },
"headersFooters": { "header": { "blocks": [{ "paragraphFormat": {
"listFormat": { } }, "characterFormat": { }, "inlines": [ ] } ] }, "footer": {
"blocks": [{ "paragraphFormat": { "listFormat": { } }, "characterFormat": { },
"inlines": [ ] } ] } } }, "characterFormat": { "bold": false, "italic":
false, "fontSize": 11, "fontFamily": "Calibri", "underline": "None",
"strikethrough": "None", "baselineAlignment": "Normal", "highlightColor":
"NoColor", "fontColor": "empty", "fontSizeBidi": 11, "fontFamilyBidi":
"Calibri", "allCaps": false }, "paragraphFormat": { "leftIndent": 0,
"rightIndent": 0, "firstLineIndent": 0, "textAlignment": "Left",
"beforeSpacing": 0, "afterSpacing": 0, "lineSpacing": 1.0791666507720947,
"lineSpacingType": "Multiple", "listFormat": { }, "bidi": false },
"defaultTabWidth": 36, "trackChanges": false, "enforcement": false,
"hashValue": "", "saltValue": "", "formatting": false, "protectionType":
"NoProtection", "dontUseHTMLParagraphAutoSpacing": false,
"formFieldShading": true, "styles": [{ "name": "Normal", "type":
"Paragraph", "paragraphFormat": { "lineSpacing": 1.149999976158142,
"lineSpacingType": "Multiple", "listFormat": { } }, "characterFormat": {
"fontFamily": "Calibri" }, "next": "Normal" }, { "name": "Default Paragraph
Font", "type": "Character", "characterFormat": { } }, { "name": "Heading 1
Char", "type": "Character", "characterFormat": { "fontSize": 16,
"fontFamily": "Calibri Light", "fontColor": "#2F5496" }, "basedOn": "Default
Paragraph Font" }, { "name": "Heading 1", "type": "Paragraph",
"paragraphFormat": { "beforeSpacing": 12, "afterSpacing": 0, "outlineLevel":
"Level1", "listFormat": { } }, "characterFormat": { "fontSize": 16,
"fontFamily": "Calibri Light", "fontColor": "#2F5496" }, "basedOn":
"Normal", "link": "Heading 1 Char", "next": "Normal" }, { "name": "Heading 2
Char", "type": "Character", "characterFormat": { "fontSize": 13,
"fontFamily": "Calibri Light", "fontColor": "#2F5496" }, "basedOn": "Default
Paragraph Font" }, { "name": "Heading 2", "type": "Paragraph",
"paragraphFormat": { "beforeSpacing": 2, "afterSpacing": 6, "outlineLevel":
"Level2", "listFormat": { } }, "characterFormat": { "fontSize": 13,
"fontFamily": "Calibri Light", "fontColor": "#2F5496" }, "basedOn":
"Normal", "link": "Heading 2 Char", "next": "Normal" }, { "name": "Heading
3", "type": "Paragraph", "paragraphFormat": { "leftIndent": 0,
"rightIndent": 0, "firstLineIndent": 0, "textAlignment": "Left",
"beforeSpacing": 2, "afterSpacing": 0, "lineSpacing": 1.0791666507720947,
"lineSpacingType": "Multiple", "outlineLevel": "Level3", "listFormat": { } },
"characterFormat": { "fontSize": 12, "fontFamily": "Calibri Light",
"fontColor": "#1F3763" }, "basedOn": "Normal", "link": "Heading 3 Char",
"next": "Normal" }, { "name": "Heading 3 Char", "type": "Character",
"characterFormat": { "fontSize": 12, "fontFamily": "Calibri Light",
"fontColor": "#1F3763" }, "basedOn": "Default Paragraph Font" }, { "name":
"Heading 4", "type": "Paragraph", "paragraphFormat": { "leftIndent": 0,
"rightIndent": 0, "firstLineIndent": 0, "textAlignment": "Left",

```

```
"beforeSpacing": 2, "afterSpacing": 0, "lineSpacing": 1.0791666507720947,
"lineSpacingType": "Multiple", "outlineLevel": "Level4", "listFormat": { } },
"characterFormat": { "italic": true, "fontFamily": "Calibri Light",
"fontColor": "#2F5496" }, "basedOn": "Normal", "link": "Heading 4 Char",
"next": "Normal" }, { "name": "Heading 4 Char", "type": "Character",
"characterFormat": { "italic": true, "fontFamily": "Calibri Light",
"fontColor": "#2F5496" }, "basedOn": "Default Paragraph Font" }, { "name":
"Heading 5", "type": "Paragraph", "paragraphFormat": { "leftIndent": 0,
"rightIndent": 0, "firstLineIndent": 0, "textAlignment": "Left",
"beforeSpacing": 2, "afterSpacing": 0, "lineSpacing": 1.0791666507720947,
"lineSpacingType": "Multiple", "outlineLevel": "Level5", "listFormat": { } },
"characterFormat": { "fontFamily": "Calibri Light", "fontColor": "#2F5496"
}, "basedOn": "Normal", "link": "Heading 5 Char", "next": "Normal" }, {
"name": "Heading 5 Char", "type": "Character", "characterFormat": {
"fontFamily": "Calibri Light", "fontColor": "#2F5496" }, "basedOn": "Default
Paragraph Font" }, { "name": "Heading 6", "type": "Paragraph",
"paragraphFormat": { "leftIndent": 0, "rightIndent": 0, "firstLineIndent":
0, "textAlignment": "Left", "beforeSpacing": 2, "afterSpacing": 0,
"lineSpacing": 1.0791666507720947, "lineSpacingType": "Multiple",
"outlineLevel": "Level6", "listFormat": { } }, "characterFormat": {
"fontFamily": "Calibri Light", "fontColor": "#1F3763" }, "basedOn":
"Normal", "link": "Heading 6 Char", "next": "Normal" }, { "name": "Heading 6
Char", "type": "Character", "characterFormat": { "fontFamily": "Calibri
Light", "fontColor": "#1F3763" }, "basedOn": "Default Paragraph Font" }},
"lists": [], "abstractLists": [], "comments": [], "revisions": [],
"customXml": [ ] }`;
    // open the default document.
    this.$refs.documenteditor.open(sfdd);
  }
}
</script>
<style>
  @import '../node_modules/@syncfusion/ej2-base/styles/material.css';
  @import '../node_modules/@syncfusion/ej2-buttons/styles/material.css';
  @import '../node_modules/@syncfusion/ej2-inputs/styles/material.css';
  @import '../node_modules/@syncfusion/ej2-popups/styles/material.css';
  @import '../node_modules/@syncfusion/ej2-lists/styles/material.css';
  @import '../node_modules/@syncfusion/ej2-
navigations/styles/material.css';
  @import '../node_modules/@syncfusion/ej2-
splitbuttons/styles/material.css';
  @import '../node_modules/@syncfusion/ej2-dropdowns/styles/material.css';
  @import "../node_modules/@syncfusion/ej2-vue-
documenteditor/styles/material.css";
</style>
```

{% previewsample "page.domainurl/code-snippet/document-editor/open-default-document-cs1" %}

*Opening a default document in DocumentEditorContainer*

#### APP.VUE

```
<template>
  <ejs-documenteditorcontainer ref="container" :serviceUrl='serviceUrl'
:height='height' :enableToolBar='true' :created= "onCreate()"> </ejs-
documenteditorcontainer>
</template>
```

```

<script>
  import Vue from 'vue';
  import { DocumentEditorContainerPlugin,
DocumentEditorContainerComponent, Toolbar } from '@syncfusion/ej2-vue-
documenteditor';
  Vue.use(DocumentEditorContainerPlugin);
  export default {
    data() {
      return {
        serviceUrl: 'https://ej2services.syncfusion.com/production/web-
services/api/documenteditor/',
        height: '590px'
      }
    },
    provide: {
      DocumentEditorContainer: [Toolbar]
    },
    methods: {
      onCreate: function() {
      }
    },
    mounted: function() {
      // load your default document here
      let sfdt = `{ "sections": [{ "sectionFormat": { "pageWidth": 612,
"pageHeight": 792, "leftMargin": 72, "rightMargin": 72, "topMargin": 72,
"bottomMargin": 72, "differentFirstPage": false, "differentOddAndEvenPages":
false, "headerDistance": 36, "footerDistance": 36, "bidi": false },
"blocks": [{ "paragraphFormat": { "afterSpacing": 30, "styleName": "Heading
1", "listFormat": { } }, "characterFormat": { }, "inlines": [{
"characterFormat": { }, "text": "Adventure Works Cycles" } ] },
"headersFooters": { "header": { "blocks": [{ "paragraphFormat": {
"listFormat": { } }, "characterFormat": { }, "inlines": [ ] } ] }, "footer": {
"blocks": [{ "paragraphFormat": { "listFormat": { } }, "characterFormat": { },
"inlines": [ ] } ] } } ], "characterFormat": { "bold": false, "italic":
false, "fontSize": 11, "fontFamily": "Calibri", "underline": "None",
"strikethrough": "None", "baselineAlignment": "Normal", "highlightColor":
"NoColor", "fontColor": "empty", "fontSizeBidi": 11, "fontFamilyBidi":
"Calibri", "allCaps": false }, "paragraphFormat": { "leftIndent": 0,
"rightIndent": 0, "firstLineIndent": 0, "textAlignment": "Left",
"beforeSpacing": 0, "afterSpacing": 0, "lineSpacing": 1.0791666507720947,
"lineSpacingType": "Multiple", "listFormat": { }, "bidi": false },
"defaultTabWidth": 36, "trackChanges": false, "enforcement": false,
"hashValue": "", "saltValue": "", "formatting": false, "protectionType":
"NoProtection", "dontUseHTMLParagraphAutoSpacing": false,
"formFieldShading": true, "styles": [{ "name": "Normal", "type":
"Paragraph", "paragraphFormat": { "lineSpacing": 1.149999976158142,
"lineSpacingType": "Multiple", "listFormat": { } }, "characterFormat": {
"fontFamily": "Calibri" }, "next": "Normal" }, { "name": "Default Paragraph
Font", "type": "Character", "characterFormat": { } }, { "name": "Heading 1
Char", "type": "Character", "characterFormat": { "fontSize": 16,
"fontFamily": "Calibri Light", "fontColor": "#2F5496" }, "basedOn": "Default
Paragraph Font" }, { "name": "Heading 1", "type": "Paragraph",
"paragraphFormat": { "beforeSpacing": 12, "afterSpacing": 0, "outlineLevel":
"Level1", "listFormat": { } }, "characterFormat": { "fontSize": 16,
"fontFamily": "Calibri Light", "fontColor": "#2F5496" }, "basedOn":
"Normal", "link": "Heading 1 Char", "next": "Normal" }, { "name": "Heading 2
Char", "type": "Character", "characterFormat": { "fontSize": 13,

```

```

"fontFamily": "Calibri Light", "fontColor": "#2F5496" }, "basedOn": "Default Paragraph Font" }, { "name": "Heading 2", "type": "Paragraph", "paragraphFormat": { "beforeSpacing": 2, "afterSpacing": 6, "outlineLevel": "Level2", "listFormat": {} }, "characterFormat": { "fontSize": 13, "fontFamily": "Calibri Light", "fontColor": "#2F5496" }, "basedOn": "Normal", "link": "Heading 2 Char", "next": "Normal" }, { "name": "Heading 3", "type": "Paragraph", "paragraphFormat": { "leftIndent": 0, "rightIndent": 0, "firstLineIndent": 0, "textAlignment": "Left", "beforeSpacing": 2, "afterSpacing": 0, "lineSpacing": 1.0791666507720947, "lineSpacingType": "Multiple", "outlineLevel": "Level3", "listFormat": {} }, "characterFormat": { "fontSize": 12, "fontFamily": "Calibri Light", "fontColor": "#1F3763" }, "basedOn": "Normal", "link": "Heading 3 Char", "next": "Normal" }, { "name": "Heading 3 Char", "type": "Character", "characterFormat": { "fontSize": 12, "fontFamily": "Calibri Light", "fontColor": "#1F3763" }, "basedOn": "Default Paragraph Font" }, { "name": "Heading 4", "type": "Paragraph", "paragraphFormat": { "leftIndent": 0, "rightIndent": 0, "firstLineIndent": 0, "textAlignment": "Left", "beforeSpacing": 2, "afterSpacing": 0, "lineSpacing": 1.0791666507720947, "lineSpacingType": "Multiple", "outlineLevel": "Level4", "listFormat": {} }, "characterFormat": { "italic": true, "fontFamily": "Calibri Light", "fontColor": "#2F5496" }, "basedOn": "Normal", "link": "Heading 4 Char", "next": "Normal" }, { "name": "Heading 4 Char", "type": "Character", "characterFormat": { "italic": true, "fontFamily": "Calibri Light", "fontColor": "#2F5496" }, "basedOn": "Default Paragraph Font" }, { "name": "Heading 5", "type": "Paragraph", "paragraphFormat": { "leftIndent": 0, "rightIndent": 0, "firstLineIndent": 0, "textAlignment": "Left", "beforeSpacing": 2, "afterSpacing": 0, "lineSpacing": 1.0791666507720947, "lineSpacingType": "Multiple", "outlineLevel": "Level5", "listFormat": {} }, "characterFormat": { "fontFamily": "Calibri Light", "fontColor": "#2F5496" }, "basedOn": "Normal", "link": "Heading 5 Char", "next": "Normal" }, { "name": "Heading 5 Char", "type": "Character", "characterFormat": { "fontFamily": "Calibri Light", "fontColor": "#2F5496" }, "basedOn": "Default Paragraph Font" }, { "name": "Heading 6", "type": "Paragraph", "paragraphFormat": { "leftIndent": 0, "rightIndent": 0, "firstLineIndent": 0, "textAlignment": "Left", "beforeSpacing": 2, "afterSpacing": 0, "lineSpacing": 1.0791666507720947, "lineSpacingType": "Multiple", "outlineLevel": "Level6", "listFormat": {} }, "characterFormat": { "fontFamily": "Calibri Light", "fontColor": "#1F3763" }, "basedOn": "Normal", "link": "Heading 6 Char", "next": "Normal" }, { "name": "Heading 6 Char", "type": "Character", "characterFormat": { "fontFamily": "Calibri Light", "fontColor": "#1F3763" }, "basedOn": "Default Paragraph Font" }], "lists": [], "abstractLists": [], "comments": [], "revisions": [], "customXml": [] }`;

    // open the default document.
    this.$refs.container.ej2Instances.calibri.documentEditor.open(sfdd);
  }
}
</script>
<style>
  @import '../node_modules/@syncfusion/ej2-base/styles/material.css';
  @import '../node_modules/@syncfusion/ej2-buttons/styles/material.css';
  @import '../node_modules/@syncfusion/ej2-inputs/styles/material.css';
  @import '../node_modules/@syncfusion/ej2-popups/styles/material.css';
  @import '../node_modules/@syncfusion/ej2-lists/styles/material.css';
  @import '../node_modules/@syncfusion/ej2-navigations/styles/material.css';

```

```

    @import '../node_modules/@syncfusion/ej2-splitbuttons/styles/material.css';
    @import '../node_modules/@syncfusion/ej2-dropdowns/styles/material.css';
    @import "../node_modules/@syncfusion/ej2-vue-documenteditor/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/document-editor/open-default-document-cs2" %}

Read only by default in Vue Document editor component

In this article, we are going to see how to open a document in read only mode by default in Document Editor & Document Editor Container.

*How to open a document in read only mode by default in DocumentEditor*

### APP.VUE

```

<template>
  <div id="app">
    <ejs-documenteditor id="container6" ref="documenteditor"
      :created="onCreated" :documentChange="onDocumentChange" style="width: 100%; "
      :serviceUrl='serviceUrl' :isReadOnly='false' :enablePrint='true'
      :enableSfdtExport='true' :enableSelection='true' :enableContextMenu='true'
      :enableSearch='true' :enableOptionsPane='true' :enableWordExport='true'
      :enableTextExport='true' :enableEditor='true' :enableImageResizer='true'
      :enableEditorHistory='true' :enableHyperlinkDialog='true'
      :enableTableDialog='true' :enableBookmarkDialog='true'
      :enableTableOfContentsDialog='true' :enablePageSetupDialog='true'
      :enableStyleDialog='true' :enableListDialog='true'
      :enableParagraphDialog='true' :enableFontDialog='true'
      :enableTablePropertiesDialog='true' :enableBordersAndShadingDialog='true'
      :enableTableOptionsDialog='true' height='400px'></ejs-documenteditor>
    </div>
  </template>
<script>
  import Vue from 'vue';
  import { DocumentEditorPlugin, DocumentEditorComponent, Print,
    SfdtExport, WordExport, TextExport, Selection, Search, Editor, ImageResizer,
    EditorHistory, ContextMenu, OptionsPane, HyperlinkDialog, TableDialog,
    BookmarkDialog, TableOfContentsDialog, PageSetupDialog, StyleDialog,
    ListDialog, ParagraphDialog, BulletsAndNumberingDialog, FontDialog,
    TablePropertiesDialog, BordersAndShadingDialog, TableOptionsDialog,
    CellOptionsDialog, StylesDialog } from '@syncfusion/ej2-vue-documenteditor';
  Vue.use(DocumentEditorPlugin);
  export default {
    data() {
      return
    },
    provide: {
      DocumentEditor: [Print, SfdtExport, WordExport, TextExport,
        Selection, Search, Editor, ImageResizer, EditorHistory, ContextMenu,
        OptionsPane, HyperlinkDialog, TableDialog, BookmarkDialog,
        TableOfContentsDialog, PageSetupDialog, StyleDialog, ListDialog,
        ParagraphDialog, BulletsAndNumberingDialog, FontDialog,

```

```

TablePropertiesDialog, BordersAndShadingDialog, TableOptionsDialog,
CellOptionsDialog, StylesDialog]
},
methods: {
    onCreate : function() {
        let sfdt = `{ "sections": [{ "sectionFormat": { "pageWidth": 612,
"pageHeight": 792, "leftMargin": 72, "rightMargin": 72, "topMargin": 72,
"bottomMargin": 72, "differentFirstPage": false, "differentOddAndEvenPages":
false, "headerDistance": 36, "footerDistance": 36, "bidi": false },
"blocks": [{ "paragraphFormat": { "afterSpacing": 30, "styleName": "Heading
1", "listFormat": {} }, "characterFormat": {}, "inlines": [{
"characterFormat": {}, "text": "Adventure Works Cycles" } ] },
"headersFooters": { "header": { "blocks": [{ "paragraphFormat": {
"listFormat": {} }, "characterFormat": {}, "inlines": [ ] } ] }, "footer": {
"blocks": [{ "paragraphFormat": { "listFormat": {} }, "characterFormat": {},
"inlines": [ ] } ] } } ], "characterFormat": { "bold": false, "italic":
false, "fontSize": 11, "fontFamily": "Calibri", "underline": "None",
"strikethrough": "None", "baselineAlignment": "Normal", "highlightColor":
"NoColor", "fontColor": "empty", "fontSizeBidi": 11, "fontFamilyBidi":
"Calibri", "allCaps": false }, "paragraphFormat": { "leftIndent": 0,
"rightIndent": 0, "firstLineIndent": 0, "textAlignment": "Left",
"beforeSpacing": 0, "afterSpacing": 0, "lineSpacing": 1.0791666507720947,
"lineSpacingType": "Multiple", "listFormat": {}, "bidi": false },
"defaultTabWidth": 36, "trackChanges": false, "enforcement": false,
"hashValue": "", "saltValue": "", "formatting": false, "protectionType":
"NoProtection", "dontUseHTMLParagraphAutoSpacing": false,
"formFieldShading": true, "styles": [{ "name": "Normal", "type":
"Paragraph", "paragraphFormat": { "lineSpacing": 1.149999976158142,
"lineSpacingType": "Multiple", "listFormat": {} }, "characterFormat": {
"fontFamily": "Calibri" }, "next": "Normal" }, { "name": "Default Paragraph
Font", "type": "Character", "characterFormat": {} }, { "name": "Heading 1
Char", "type": "Character", "characterFormat": { "fontSize": 16,
"fontFamily": "Calibri Light", "fontColor": "#2F5496" }, "basedOn": "Default
Paragraph Font" }, { "name": "Heading 1", "type": "Paragraph",
"paragraphFormat": { "beforeSpacing": 12, "afterSpacing": 0, "outlineLevel":
"Level1", "listFormat": {} }, "characterFormat": { "fontSize": 16,
"fontFamily": "Calibri Light", "fontColor": "#2F5496" }, "basedOn":
"Normal", "link": "Heading 1 Char", "next": "Normal" }, { "name": "Heading 2
Char", "type": "Character", "characterFormat": { "fontSize": 13,
"fontFamily": "Calibri Light", "fontColor": "#2F5496" }, "basedOn": "Default
Paragraph Font" }, { "name": "Heading 2", "type": "Paragraph",
"paragraphFormat": { "beforeSpacing": 2, "afterSpacing": 6, "outlineLevel":
"Level2", "listFormat": {} }, "characterFormat": { "fontSize": 13,
"fontFamily": "Calibri Light", "fontColor": "#2F5496" }, "basedOn":
"Normal", "link": "Heading 2 Char", "next": "Normal" }, { "name": "Heading
3", "type": "Paragraph", "paragraphFormat": { "leftIndent": 0,
"rightIndent": 0, "firstLineIndent": 0, "textAlignment": "Left",
"beforeSpacing": 2, "afterSpacing": 0, "lineSpacing": 1.0791666507720947,
"lineSpacingType": "Multiple", "outlineLevel": "Level3", "listFormat": {} },
"characterFormat": { "fontSize": 12, "fontFamily": "Calibri Light",
"fontColor": "#1F3763" }, "basedOn": "Normal", "link": "Heading 3 Char",
"next": "Normal" }, { "name": "Heading 3 Char", "type": "Character",
"characterFormat": { "fontSize": 12, "fontFamily": "Calibri Light",
"fontColor": "#1F3763" }, "basedOn": "Default Paragraph Font" }, { "name":
"Heading 4", "type": "Paragraph", "paragraphFormat": { "leftIndent": 0,
"rightIndent": 0, "firstLineIndent": 0, "textAlignment": "Left",
"beforeSpacing": 2, "afterSpacing": 0, "lineSpacing": 1.0791666507720947,

```



```

"lineSpacingType": "Multiple", "outlineLevel": "Level4", "listFormat": {} },
"characterFormat": { "italic": true, "fontFamily": "Calibri Light",
"fontColor": "#2F5496" }, "basedOn": "Normal", "link": "Heading 4 Char",
"next": "Normal" }, { "name": "Heading 4 Char", "type": "Character",
"characterFormat": { "italic": true, "fontFamily": "Calibri Light",
"fontColor": "#2F5496" }, "basedOn": "Default Paragraph Font" }, { "name":
"Heading 5", "type": "Paragraph", "paragraphFormat": { "leftIndent": 0,
"rightIndent": 0, "firstLineIndent": 0, "textAlignment": "Left",
"beforeSpacing": 2, "afterSpacing": 0, "lineSpacing": 1.0791666507720947,
"lineSpacingType": "Multiple", "outlineLevel": "Level5", "listFormat": {} },
"characterFormat": { "fontFamily": "Calibri Light", "fontColor": "#2F5496"
}, "basedOn": "Normal", "link": "Heading 5 Char", "next": "Normal" }, {
"name": "Heading 5 Char", "type": "Character", "characterFormat": {
"fontFamily": "Calibri Light", "fontColor": "#2F5496" }, "basedOn": "Default
Paragraph Font" }, { "name": "Heading 6", "type": "Paragraph",
"paragraphFormat": { "leftIndent": 0, "rightIndent": 0, "firstLineIndent":
0, "textAlignment": "Left", "beforeSpacing": 2, "afterSpacing": 0,
"lineSpacing": 1.0791666507720947, "lineSpacingType": "Multiple",
"outlineLevel": "Level6", "listFormat": {} }, "characterFormat": {
"fontFamily": "Calibri Light", "fontColor": "#1F3763" }, "basedOn":
"Normal", "link": "Heading 6 Char", "next": "Normal" }, { "name": "Heading 6
Char", "type": "Character", "characterFormat": { "fontFamily": "Calibri
Light", "fontColor": "#1F3763" }, "basedOn": "Default Paragraph Font" }],
"lists": [], "abstractLists": [], "comments": [], "revisions": [],
"customXml": [] }`;
    //Open default document in Document Editor.
    this.$refs.documenteditor.open(sfdt);
  },
  onDocumentChange : function() {
    //Enable read only mode.
    this.$refs.documenteditor.isReadOnly = true;
  }
}
}
</script>
<style>
  @import '../node_modules/@syncfusion/ej2-base/styles/material.css';
  @import '../node_modules/@syncfusion/ej2-buttons/styles/material.css';
  @import '../node_modules/@syncfusion/ej2-inputs/styles/material.css';
  @import '../node_modules/@syncfusion/ej2-popups/styles/material.css';
  @import '../node_modules/@syncfusion/ej2-lists/styles/material.css';
  @import '../node_modules/@syncfusion/ej2-
navigations/styles/material.css';
  @import '../node_modules/@syncfusion/ej2-
splitbuttons/styles/material.css';
  @import '../node_modules/@syncfusion/ej2-dropdowns/styles/material.css';
  @import '../node_modules/@syncfusion/ej2-vue-
documenteditor/styles/material.css';
</style>

```

{% previewsample "page.domainurl/code-snippet/document-editor/getting-started-cs4" %}

[How to open a document in ready only mode by default in DocumentEditorContainer](#)

## APP.VUE

```
<template>
```

```

<div id="app">
  <ejs-documenteditorcontainer :serviceUrl='serviceUrl'
:enableToolbar='true' v-bind:documentChange="onDocumentChange" v-
bind:created="onCreated" ref="documentEditorContainer" height='590px'>
</ejs-documenteditorcontainer>
</div>
</template>
<script>
  import Vue from 'vue';
  import { DocumentEditorContainerPlugin,
DocumentEditorContainerComponent,Toolbar } from '@syncfusion/ej2-vue-
documenteditor';
  Vue.use(DocumentEditorContainerPlugin);
  export default {
    data(){
      return {
serviceUrl:'https://ej2services.syncfusion.com/production/web-
services/api/documenteditor/'
    },
    provide: {
      //Inject require modules.
      DocumentEditorContainer: [Toolbar]
    },
    methods: {
      onCreated : function() {
        let sfdt = `{ "sections": [{ "sectionFormat": { "pageWidth": 612,
"pageHeight": 792, "leftMargin": 72, "rightMargin": 72, "topMargin": 72,
"bottomMargin": 72, "differentFirstPage": false, "differentOddAndEvenPages":
false, "headerDistance": 36, "footerDistance": 36, "bidi": false },
"blocks": [{ "paragraphFormat": { "afterSpacing": 30, "styleName": "Heading
1", "listFormat": { } }, "characterFormat": { }, "inlines": [{
"characterFormat": { }, "text": "Adventure Works Cycles" } ] },
"headersFooters": { "header": { "blocks": [{ "paragraphFormat": {
"listFormat": { } }, "characterFormat": { }, "inlines": [ ] } ] }, "footer": {
"blocks": [{ "paragraphFormat": { "listFormat": { } }, "characterFormat": { },
"inlines": [ ] } ] } } }, "characterFormat": { "bold": false, "italic":
false, "fontSize": 11, "fontFamily": "Calibri", "underline": "None",
"strikethrough": "None", "baselineAlignment": "Normal", "highlightColor":
"NoColor", "fontColor": "empty", "fontSizeBidi": 11, "fontFamilyBidi":
"Calibri", "allCaps": false }, "paragraphFormat": { "leftIndent": 0,
"rightIndent": 0, "firstLineIndent": 0, "textAlignment": "Left",
"beforeSpacing": 0, "afterSpacing": 0, "lineSpacing": 1.0791666507720947,
"lineSpacingType": "Multiple", "listFormat": { }, "bidi": false },
"defaultTabWidth": 36, "trackChanges": false, "enforcement": false,
"hashValue": "", "saltValue": "", "formatting": false, "protectionType":
"NoProtection", "dontUseHTMLParagraphAutoSpacing": false,
"formFieldShading": true, "styles": [{ "name": "Normal", "type":
"Paragraph", "paragraphFormat": { "lineSpacing": 1.149999976158142,
"lineSpacingType": "Multiple", "listFormat": { }, "characterFormat": {
"fontFamily": "Calibri" }, "next": "Normal" }, { "name": "Default Paragraph
Font", "type": "Character", "characterFormat": { } }, { "name": "Heading 1
Char", "type": "Character", "characterFormat": { "fontSize": 16,
"fontFamily": "Calibri Light", "fontColor": "#2F5496" }, "basedOn": "Default
Paragraph Font" }, { "name": "Heading 1", "type": "Paragraph",
"paragraphFormat": { "beforeSpacing": 12, "afterSpacing": 0, "outlineLevel":
"Levell1", "listFormat": { } }, "characterFormat": { "fontSize": 16,
"fontFamily": "Calibri Light", "fontColor": "#2F5496" }, "basedOn":

```

```

"Normal", "link": "Heading 1 Char", "next": "Normal" }, { "name": "Heading 2
Char", "type": "Character", "characterFormat": { "fontSize": 13,
"fontFamily": "Calibri Light", "fontColor": "#2F5496" }, "basedOn": "Default
Paragraph Font" }, { "name": "Heading 2", "type": "Paragraph",
"paragraphFormat": { "beforeSpacing": 2, "afterSpacing": 6, "outlineLevel":
"Level2", "listFormat": {} }, "characterFormat": { "fontSize": 13,
"fontFamily": "Calibri Light", "fontColor": "#2F5496" }, "basedOn":
"Normal", "link": "Heading 2 Char", "next": "Normal" }, { "name": "Heading
3", "type": "Paragraph", "paragraphFormat": { "leftIndent": 0,
"rightIndent": 0, "firstLineIndent": 0, "textAlignment": "Left",
"beforeSpacing": 2, "afterSpacing": 0, "lineSpacing": 1.0791666507720947,
"lineSpacingType": "Multiple", "outlineLevel": "Level3", "listFormat": {} },
"characterFormat": { "fontSize": 12, "fontFamily": "Calibri Light",
"fontColor": "#1F3763" }, "basedOn": "Normal", "link": "Heading 3 Char",
"next": "Normal" }, { "name": "Heading 3 Char", "type": "Character",
"characterFormat": { "fontSize": 12, "fontFamily": "Calibri Light",
"fontColor": "#1F3763" }, "basedOn": "Default Paragraph Font" }, { "name":
"Heading 4", "type": "Paragraph", "paragraphFormat": { "leftIndent": 0,
"rightIndent": 0, "firstLineIndent": 0, "textAlignment": "Left",
"beforeSpacing": 2, "afterSpacing": 0, "lineSpacing": 1.0791666507720947,
"lineSpacingType": "Multiple", "outlineLevel": "Level4", "listFormat": {} },
"characterFormat": { "italic": true, "fontFamily": "Calibri Light",
"fontColor": "#2F5496" }, "basedOn": "Normal", "link": "Heading 4 Char",
"next": "Normal" }, { "name": "Heading 4 Char", "type": "Character",
"characterFormat": { "italic": true, "fontFamily": "Calibri Light",
"fontColor": "#2F5496" }, "basedOn": "Default Paragraph Font" }, { "name":
"Heading 5", "type": "Paragraph", "paragraphFormat": { "leftIndent": 0,
"rightIndent": 0, "firstLineIndent": 0, "textAlignment": "Left",
"beforeSpacing": 2, "afterSpacing": 0, "lineSpacing": 1.0791666507720947,
"lineSpacingType": "Multiple", "outlineLevel": "Level5", "listFormat": {} },
"characterFormat": { "fontFamily": "Calibri Light", "fontColor": "#2F5496"
}, "basedOn": "Normal", "link": "Heading 5 Char", "next": "Normal" }, {
"name": "Heading 5 Char", "type": "Character", "characterFormat": {
"fontFamily": "Calibri Light", "fontColor": "#2F5496" }, "basedOn": "Default
Paragraph Font" }, { "name": "Heading 6", "type": "Paragraph",
"paragraphFormat": { "leftIndent": 0, "rightIndent": 0, "firstLineIndent":
0, "textAlignment": "Left", "beforeSpacing": 2, "afterSpacing": 0,
"lineSpacing": 1.0791666507720947, "lineSpacingType": "Multiple",
"outlineLevel": "Level6", "listFormat": {} }, "characterFormat": {
"fontFamily": "Calibri Light", "fontColor": "#1F3763" }, "basedOn":
"Normal", "link": "Heading 6 Char", "next": "Normal" }, { "name": "Heading 6
Char", "type": "Character", "characterFormat": { "fontFamily": "Calibri
Light", "fontColor": "#1F3763" }, "basedOn": "Default Paragraph Font" }],
"lists": [], "abstractLists": [], "comments": [], "revisions": [],
"customXml": [] }`;

```

*//Open default document in Document Editor.*

```

this.$refs.documentEditorContainer.ej2Instances.documentEditor.open(sfdd);
},
onDocumentChange : function() {
    //Enable read only mode.
    this.$refs.documentEditorContainer.restrictEditing = true;
}
},
}
</script>
<style>

```

```

    @import '../node_modules/@syncfusion/ej2-base/styles/material.css';
    @import '../node_modules/@syncfusion/ej2-buttons/styles/material.css';
    @import '../node_modules/@syncfusion/ej2-inputs/styles/material.css';
    @import '../node_modules/@syncfusion/ej2-popups/styles/material.css';
    @import '../node_modules/@syncfusion/ej2-lists/styles/material.css';
    @import '../node_modules/@syncfusion/ej2-navigations/styles/material.css';
    @import '../node_modules/@syncfusion/ej2-splitbuttons/styles/material.css';
    @import '../node_modules/@syncfusion/ej2-dropdowns/styles/material.css';
    @import "../node_modules/@syncfusion/ej2-vue-documenteditor/styles/material.css";
  </style>

```

{% previewsample "page.domainurl/code-snippet/document-editor/getting-started-cs5" %}

Note: You can use the `restrictEditing` in Document Editor Container and `isReadOnly` in Document Editor based on your requirement to change component to read only mode.

Open document by address in Vue Document editor component

[How to open a document from URL in DocumentEditor](#)

In this article, we are going to see how to open a document from URL in DocumentEditor

please refer below example for client-side code

,

```

<template>
<div id="app">
<button id='import' v-on:click="onClick">Import</button>
<ejs-documenteditorcontainer ref='container' :serviceUrl='serviceUrl' height="590px" id='container'
:enableToolbar='true'></ejs-documenteditorcontainer>
</div>
</template>
<script>
import Vue from 'vue';
import { DocumentEditorContainerPlugin, DocumentEditorContainerComponent, Toolbar } from
'@syncfusion/ej2-vue-documenteditor';
Vue.use(DocumentEditorContainerPlugin);
export default {
data() {
return { serviceUrl:'https://ej2services.syncfusion.com/production/web-services/api/documenteditor/'
};
},
provide: {

```

```
//Inject require modules.
DocumentEditorContainer: [Toolbar]
},
methods: {
onClick:function() {
let obj = this;
let http: XMLHttpRequest = new XMLHttpRequest();
//add your url in which you want to open document inside the ""
let content = { fileUrl: "" };
let baseUrl: string = "/api/documenteditor/ImportFileURL";
http.open("POST", baseUrl, true);
http.setRequestHeader("Content-Type", "application/json;charset=UTF-8");
http.onreadystatechange = () => {
if (http.readyState === 4) {
if (http.status === 200 || http.status === 304) {
//open the SFDt text in Document Editor
this.$refs.container.ej2Instances.documentEditor.open(http.responseText);
}
}
};
http.send(JSON.stringify(content));
}
}
};
</script>
`
```

please refer below example for server-side code

```
`c#
[AcceptVerbs("Post")]
public string ImportFileURL([FromBody]FileInfo param)
{
try {
using(WebClient client = new WebClient())
```

```
{
MemoryStream stream = new MemoryStream(client.DownloadData(param.fileUrl));
WordDocument document = WordDocument.Load(stream, FormatType.Docx);
string json = Newtonsoft.Json.JsonConvert.SerializeObject(document);
document.Dispose();
stream.Dispose();
return json;
}
}
catch (Exception) {
return "";
}
}

public class FileUrlInfo {
public string fileUrl { get; set; }
public string Content { get; set; }
}
、
```

### Deploy document editor component for mobile in Vue Document editor component

#### *Document editor component for Mobile*

At present, Document editor component is not responsive for mobile, and we haven't ensured the editing functionalities in mobile browsers. Whereas it works properly as a document viewer in mobile browsers.

Hence, it is recommended to switch the Document editor component as read-only in mobile browsers. Also, invoke [fitPage](#) method with [FitPageWidth](#) parameter in document change event, such as to display one full page by adjusting the zoom factor.

The following example code illustrates how to deploy Document Editor component for Mobile.

```
、
<template>
<div id="app">
<ejs-documenteditorcontainer ref="container" height="590px" :serviceUrl='serviceUrl'
:enableToolbar='true' :documentChange='onDocumentChange'> </ejs-documenteditorcontainer>
</div>
</template>
<script>
```

```
import Vue from 'vue';
import { DocumentEditorContainerPlugin, DocumentEditorContainerComponent, Toolbar } from
 '@syncfusion/ej2-vue-documenteditor';
Vue.use(DocumentEditorContainerPlugin);
export default {
  data(){
    return { serviceUrl:'https://ej2services.syncfusion.com/production/web-services/api/documenteditor/' }
  },
  provide: {
    //Inject require modules.
    DocumentEditorContainer: [Toolbar]
  },
  methods:{
    onDocumentChange: function (args) {
      //To detect the device
      let isMobileDevice = /Android|Windows Phone|webOS/i.test(navigator.userAgent);
      if (isMobileDevice) {
        this.$refs.container.ej2Instances.restrictEditing = true;
        setTimeout(() => {
          this.$refs.container.ej2Instances.documentEditor.fitPage("FitPageWidth");
        }, 50);
      }
      else {
        this.$refs.container.ej2Instances.restrictEditing = false;
      }
    }
  }
}
</script>
<style>
@import '../node_modules/@syncfusion/ej2-base/styles/material.css';
@import '../node_modules/@syncfusion/ej2-buttons/styles/material.css';
@import '../node_modules/@syncfusion/ej2-inputs/styles/material.css';
```

```

@import '../node_modules/@syncfusion/ej2-popups/styles/material.css';
@import '../node_modules/@syncfusion/ej2-lists/styles/material.css';
@import '../node_modules/@syncfusion/ej2-navigations/styles/material.css';
@import '../node_modules/@syncfusion/ej2-splitbuttons/styles/material.css';
@import '../node_modules/@syncfusion/ej2-dropdowns/styles/material.css';
@import "../node_modules/@syncfusion/ej2-vue-documenteditor/styles/material.css";
</style>
`

```

You can download the complete working example from this [GitHub location](#)

Note: You can use the [restrictEditing](#) in DocumentEditorContainer and [isReadOnly](#) in DocumentEditor based on your requirement to change component to read only mode.

#### Disable optimized text measuring in Vue Document editor component

Starting from v19.3.0.x, the accuracy of text size measurements in Document editor is improved such as to match Microsoft Word pagination for most Word documents. This improvement is included as default behavior along with an optional API [enableOptimizedTextMeasuring](#) in Document editor settings.

If you want the Document editor component to retain the document pagination (display page-by-page) behavior like v19.2.0.x and older versions. Then you can disable this optimized text measuring improvement, by setting `false` to [enableOptimizedTextMeasuring](#) property of Vue Document Editor component.

#### Disable optimized text measuring in `DocumentEditorContainer` instance

The following example code illustrates how to disable optimized text measuring improvement in `DocumentEditorContainer` instance.

```

`
<template>
<div id="app">
<ejs-documenteditorcontainer ref='documenteditor' :serviceUrl='serviceUrl' height="590px"
id='container' :enableToolbar='true' :documentEditorSettings='settings'></ejs-
documenteditorcontainer>
</div>
</template>
<script>
import Vue from 'vue';
import { DocumentEditorContainerPlugin, DocumentEditorContainerComponent, Toolbar } from
'@syncfusion/ej2-vue-documenteditor';
Vue.use(DocumentEditorContainerPlugin);
export default {
data() {

```



```

return { serviceUrl:'https://ej2services.syncfusion.com/production/web-services/api/documenteditor/',
settings:{ enableOptimizedTextMeasuring : false} };
},
provide: {
//Inject require modules.
DocumentEditorContainer: [Toolbar]
}
}
</script>
`

```

#### *Disable optimized text measuring in `DocumentEditor` instance*

The following example code illustrates how to disable optimized text measuring improvement in **DocumentEditor** instance.

```

<template>
<div id="app">
<ejs-documenteditor :serviceUrl='serviceUrl' :isReadOnly='false' :enablePrint='true'
:enableSfdtExport='true' :enableSelection='true' :enableContextMenu='true' :enableSearch='true'
:enableOptionsPane='true' :enableWordExport='true' :enableTextExport='true' :enableEditor='true'
:enableImageResizer='true' :enableEditorHistory='true' :enableHyperlinkDialog='true'
:enableTableDialog='true' :enableBookmarkDialog='true' :enableTableOfContentsDialog='true'
:enablePageSetupDialog='true' :enableStyleDialog='true' :enableListDialog='true'
:enableParagraphDialog='true' :enableFontDialog='true' :enableTablePropertiesDialog='true'
:enableBordersAndShadingDialog='true' :enableTableOptionsDialog='true' height="370px"
:documentEditorSettings='settings'> </ejs-documenteditor>
</div>
</template>
<script>
import Vue from 'vue';

import { DocumentEditorPlugin, DocumentEditorComponent, Print, SfdtExport, WordExport, TextExport,
Selection, Search, Editor, ImageResizer, EditorHistory, ContextMenu, OptionsPane, HyperlinkDialog,
TableDialog, BookmarkDialog, TableOfContentsDialog, PageSetupDialog, StyleDialog, ListDialog,
ParagraphDialog, BulletsAndNumberingDialog, FontDialog, TablePropertiesDialog,
BordersAndShadingDialog, TableOptionsDialog, CellOptionsDialog, StylesDialog } from '@syncfusion/ej2-
vue-documenteditor';

Vue.use(DocumentEditorPlugin);

export default {
data () {

```

```

return { serviceUrl:'https://ej2services.syncfusion.com/production/web-services/api/documenteditor/',
settings:{ enableOptimizedTextMeasuring : false} };
},
provide: {
//Inject require modules.
DocumentEditor: [Print, SfdtExport, WordExport, TextExport, Selection, Search, Editor, ImageResizer,
EditorHistory, ContextMenu, OptionsPane, HyperlinkDialog, TableDialog, BookmarkDialog,
TableOfContentsDialog, PageSetupDialog, StyleDialog, ListDialog, ParagraphDialog,
BulletsAndNumberingDialog, FontDialog, TablePropertiesDialog, BordersAndShadingDialog,
TableOptionsDialog, CellOptionsDialog, StylesDialog]
}
}
</script>
`

```

### Get the selected content in Vue Document editor component

You can get the selected content from the Vue Document Editor component as plain text and SFDT (rich text).

#### Get the selected content as plain text

You can use [text](#) API to get the selected content as plain text from Vue Document Editor component.

The following example code illustrates how to add search in google option in context menu for the selected text.

```

`
<template>
<div id="app">
<ejs-documenteditorcontainer ref='container' :serviceUrl='serviceUrl' v-on:created="onCreated"
height="590px" id='container' :enableToolbar='true'></ejs-documenteditorcontainer>
</div>
</template>
<script>
import Vue from 'vue';

import { DocumentEditorContainerPlugin, DocumentEditorContainerComponent,Toolbar} from
'@syncfusion/ej2-vue-documenteditor';

Vue.use(DocumentEditorContainerPlugin);

export default {
data() {
return {

```

```
serviceUrl:
'https://ej2services.syncfusion.com/production/web-services/api/documenteditor/',
};
},
provide: {
//Inject require modules.
DocumentEditorContainer: [Toolbar],
},
methods: {
handleCustomMenuSelect: function (args) {
var documentEditor = this.$refs.container.ej2Instances.documentEditor;
// custom Options Functionality
let id = documentEditor.element.id;
switch (args.id) {
case id + 'searchingoogle':
var searchContent = documentEditor.selection.text;
if (!documentEditor.selection.isEmpty && /\S/.test(searchContent)) {
window.open('http://google.com/search?q=' + searchContent);
}
break;
}
},
onCreated: function () {
var obj = this.$refs.container.ej2Instances.documentEditor;
var menuItems = [
{
text: 'Search In Google',
id: 'searchingoogle',
iconCss: 'e-icons e-de-ctnr-find',
},
];
// adding Custom Options
obj.contextMenu.addCustomMenu(menuItems, false);
```

```
// custom Options Select Event
obj.customContextMenuSelect = (args) => {
  this.handleCustomMenuSelect(args);
};
},
}
};
</script>
`
```

You can add the following custom options using this API,

- Save or export the selected text as text file.
- Search the selected text in Google or other search engines.
- Show synonyms for the selected word in context menu and replace with selected synonym using the setter method of same API.

*Get the selected content as SFDT (rich text)*

You can use [sfdt](#) API to get the selected content as plain text from Vue Document Editor component.

The following example code illustrates how to get the content of a bookmark and export it as SFDT.

```
<template>
<div id="app">
  <ejs-documenteditorcontainer ref='container' :serviceUrl='serviceUrl' v-on:created="onCreated"
  height="590px" id='container' :enableToolbar='true'></ejs-documenteditorcontainer>
</div>
</template>
<script>
import Vue from 'vue';

import { DocumentEditorContainerPlugin, DocumentEditorContainerComponent, Toolbar } from
'@syncfusion/ej2-vue-documenteditor';

Vue.use(DocumentEditorContainerPlugin);

export default {
  data() {
    return {
      serviceUrl:
        'https://ej2services.syncfusion.com/production/web-services/api/documenteditor/',
    }
  }
}
```

```
};
},
provide: {
  //Inject require modules.
  DocumentEditorContainer: [Toolbar],
},
methods: {
  onCreated: function () {
    var obj = this.$refs.container.ej2Instances.documentEditor;
    // To insert text in cursor position
    obj.editor.insertText('Document editor');
    // To select all the content in document
    obj.selection.selectAll();
    // Insert bookmark to selected content
    obj.editor.insertBookmark('Bookmark1');
    //Select the bookmark
    obj.selection.selectBookmark('Bookmark1');
    // To get the selected content as sfdt
    let selectedContent = obj.selection.sfdt;
    // Insert the sfdt content in cursor position using paste API
    obj.editor.paste(selectedContent);
  }
}
};
</script>
```

You can add the following custom options using this API,

- Save or export the selected content as SFDT file.
- Get the content of a bookmark in Word document as SFDT by selecting a bookmark using [selectbookmark](#) API.
- Create template content that can be inserted to multiple documents in cursor position using [paste](#) API.

[Set default format in document editor in Vue Document editor component](#)

You can set the default character format, paragraph format and section format in Document editor.

*Set the default character format*

You can use [setDefaultCharacterFormat](#) method to set the default character format. For example, Document editor default font size is 11 and you can change it as any valid value.

The following example code illustrates how to change the default font size in Document editor.

```
<template>
<div id="app">
<ejs-documenteditorcontainer ref='container' :serviceUrl='serviceUrl' v-on:created="onCreated"
height="590px" id='container' :enableToolbar='true'></ejs-documenteditorcontainer>
</div>
</template>
<script>
import Vue from 'vue';

import { DocumentEditorContainerPlugin, DocumentEditorContainerComponent, Toolbar } from
'@syncfusion/ej2-vue-documenteditor';

Vue.use(DocumentEditorContainerPlugin);

export default {
  data() {
    return {
      serviceUrl:
        'https://ej2services.syncfusion.com/production/web-services/api/documenteditor/',
    };
  },
  provide: {
    //Inject require modules.
    DocumentEditorContainer: [Toolbar],
  },
  methods: {
    onCreated: function () {
      this.$refs.container.ej2Instances.documentEditor.setDefaultCharacterFormat({fontSize: 20});
    },
  },
};
</script>
```

Similarly, you can change the required [CharacterFormatProperties](#) default value.

The following example code illustrates how to change other character format default value in Document editor.

```
<template>
<div id="app">
<ejs-documenteditorcontainer ref='container' :serviceUrl='serviceUrl' v-on:created="onCreated"
height="590px" id='container' :enableToolbar='true'></ejs-documenteditorcontainer>
</div>
</template>
<script>
import Vue from 'vue';

import { DocumentEditorContainerPlugin, DocumentEditorContainerComponent, Toolbar } from
'@syncfusion/ej2-vue-documenteditor';

Vue.use(DocumentEditorContainerPlugin);

export default {
data() {
return {
serviceUrl:
'https://ej2services.syncfusion.com/production/web-services/api/documenteditor/',
};
},
provide: {
//Inject require modules.
DocumentEditorContainer: [Toolbar],
},
methods: {
onCreated: function () {
var defaultCharacterFormat = {
bold: false,
italic: false,
baselineAlignment: 'Normal',
underline: 'None',
```

```

fontColor: '#000000',
fontFamily: 'Algerian',
fontSize: 12,
};
this.$refs.container.ej2Instances.documentEditor.setDefaultCharacterFormat(
defaultCharacterFormat
);
},
},
};
</script>
`

```

#### *Set the default paragraph format*

You can use [setDefaultParagraphFormat](#) API to set the default paragraph format. You can change the required [ParagraphFormatProperties](#) default value.

The following example code illustrates how to change the paragraph format(before spacing, line spacing etc.,) default value in Document editor.

```

<template>
<div id="app">
<ejs-documenteditorcontainer ref='container' :serviceUrl='serviceUrl' v-on:created="onCreated"
height="590px" id='container' :enableToolbar='true'></ejs-documenteditorcontainer>
</div>
</template>
<script>
import Vue from 'vue';

import { DocumentEditorContainerPlugin, DocumentEditorContainerComponent,Toolbar} from
'@syncfusion/ej2-vue-documenteditor';

Vue.use(DocumentEditorContainerPlugin);

export default {
data() {
return {
serviceUrl:
'https://ej2services.syncfusion.com/production/web-services/api/documenteditor/',
};
}
}

```



```

},
provide: {
  //Inject require modules.
  DocumentEditorContainer: [Toolbar],
},
methods: {
  onCreated: function () {
    let defaultParagraphFormat = {
      beforeSpacing: 8,
      lineSpacing: 1.5,
      leftIndent: 24,
      textAlignment: "Center"
    };
    this.$refs.container.ej2Instances.documentEditor.setDefaultParagraphFormat(defaultParagraphFormat)
  };
},
};
</script>

```

#### *Set the default section format*

You can use [setDefaultSectionFormat](#) API to set the default section format. You can change the required [SectionFormatProperties](#) default value.

The following example code illustrates how to change the section format(header and footer distance, page width and height, etc.,) default value in Document editor.

```

<template>
<div id="app">
  <ejs-documenteditorcontainer ref='container' :serviceUrl='serviceUrl' v-on:created="onCreated"
  height="590px" id='container' :enableToolbar='true'></ejs-documenteditorcontainer>
</div>
</template>
<script>
import Vue from 'vue';

```

```

import { DocumentEditorContainerPlugin, DocumentEditorContainerComponent, Toolbar } from
 '@syncfusion/ej2-vue-documenteditor';
Vue.use(DocumentEditorContainerPlugin);
export default {
  data() {
    return {
      serviceUrl:
        'https://ej2services.syncfusion.com/production/web-services/api/documenteditor/',
    };
  },
  provide: {
    //Inject require modules.
    DocumentEditorContainer: [Toolbar],
  },
  methods: {
    onCreated: function () {
      let defaultSectionFormat = {
        pageWidth: 500,
        pageHeight: 800,
        headerDistance: 56,
        footerDistance: 48,
        leftMargin: 12,
        rightMargin: 12,
        topMargin: 0,
        bottomMargin: 0
      };
      this.$refs.container.ej2Instances.documentEditor.setDefaultSectionFormat(defaultSectionFormat);
    },
  },
};
</script>

```

Show [hide spinner](#) in Vue Document editor component

Using [spinner](#) component, you can show/hide spinner while opening document in DocumentEditor .

Example code snippet to show/hide spinner

```
// showSpinner() will make the spinner visible
showSpinner(document.getElementById('container'));

// hideSpinner() method used hide spinner
hideSpinner(document.getElementById('container'));
```

Refer to the following example.

### APP.VUE

```
<template>
  <div id="app">
    <button id='import' v-on:click="onClick">Load Document</button>
    <ejs-documenteditorcontainer ref='container' :serviceUrl='serviceUrl'
    v-on:created="onCreated" height="590px" id='container'
    :enableToolbar='true'></ejs-documenteditorcontainer>
  </div>
</template>
<script>
  import Vue from 'vue';
  import {
    DocumentEditorContainerPlugin,
    DocumentEditorContainerComponent,
    Toolbar,
  } from '@syncfusion/ej2-vue-documenteditor';
  import { showSpinner, hideSpinner } from '@syncfusion/ej2-popups';
  Vue.use(DocumentEditorContainerPlugin);
  export default {
    data() {
      return {
        serviceUrl:
          'https://ej2services.syncfusion.com/production/web-
          services/api/documenteditor/',
      };
    },
    provide: {
      //Inject require modules.
      DocumentEditorContainer: [Toolbar],
    },
    methods: {
      onClick:function() {
        // load your default document here
        let data=
          '{"sections":[{"sectionFormat":{"pageWidth":612,"pageHeight":792,"leftMargin":72,"rightMargin":72,"topMargin":72,"bottomMargin":72,"differentFirstPage":false,"differentOddAndEvenPages":false,"headerDistance":36,"footerDistance":36,"bidi":false},"blocks":[{"paragraphFormat":{"afterSpacing":30,"styleName":"Heading1","listFormat":{}},"characterFormat":{}},"inlines":[{"characterFormat":{}},"text":"Adventure Works Cycles"]]}],"headersFooters":{"header":{"blocks":[{"paragraphFormat":{"listF
```

```

ormat":{}}, "characterFormat":{}, "inlines":[]]]}, "footer":{"blocks":[{"paragr
aphFormat":{"listFormat":{}}, "characterFormat":{}, "inlines":[]]]}}, "charac
terFormat":{"bold":false, "italic":false, "fontSize":11, "fontFamily":"Calibri"
, "underline":"None", "strikethrough":"None", "baselineAlignment":"Normal", "hig
hlightColor":"NoColor", "fontColor":"empty", "fontSizeBidi":11, "fontFamilyBidi
":"Calibri", "allCaps":false}, "paragraphFormat":{"leftIndent":0, "rightIndent"
:0, "firstLineIndent":0, "textAlignment":"Left", "beforeSpacing":0, "afterSpacin
g":0, "lineSpacing":1.0791666507720947, "lineSpacingType":"Multiple", "listForm
at":{}}, "bidi":false}, "defaultTabWidth":36, "trackChanges":false, "enforcement"
:false, "hashValue":"","saltValue":"","formatting":false, "protectionType":"No
Protection", "dontUseHTMLParagraphAutoSpacing":false, "formFieldShading":true,
"styles":[{"name":"Normal", "type":"Paragraph", "paragraphFormat":{"lineSpacin
g":1.149999976158142, "lineSpacingType":"Multiple", "listFormat":{}}, "characte
rFormat":{"fontFamily":"Calibri"}, "next":"Normal"}, {"name":"Default
Paragraph Font", "type":"Character", "characterFormat":{}}, {"name":"Heading 1
Char", "type":"Character", "characterFormat":{"fontSize":16, "fontFamily":"Cali
bri Light", "fontColor":"#2F5496"}, "basedOn":"Default Paragraph
Font"}, {"name":"Heading
1", "type":"Paragraph", "paragraphFormat":{"beforeSpacing":12, "afterSpacing":0
, "outlineLevel":"Level1", "listFormat":{}}, "characterFormat":{"fontSize":16, "
fontFamily":"Calibri
Light", "fontColor":"#2F5496"}, "basedOn":"Normal", "link":"Heading 1
Char", "next":"Normal"}, {"name":"Heading 2
Char", "type":"Character", "characterFormat":{"fontSize":13, "fontFamily":"Cali
bri Light", "fontColor":"#2F5496"}, "basedOn":"Default Paragraph
Font"}, {"name":"Heading
2", "type":"Paragraph", "paragraphFormat":{"beforeSpacing":2, "afterSpacing":6,
"outlineLevel":"Level2", "listFormat":{}}, "characterFormat":{"fontSize":13, "f
ontFamily":"Calibri
Light", "fontColor":"#2F5496"}, "basedOn":"Normal", "link":"Heading 2
Char", "next":"Normal"}, {"name":"Heading
3", "type":"Paragraph", "paragraphFormat":{"leftIndent":0, "rightIndent":0, "fir
stLineIndent":0, "textAlignment":"Left", "beforeSpacing":2, "afterSpacing":0, "l
ineSpacing":1.0791666507720947, "lineSpacingType":"Multiple", "outlineLevel":"
Level3", "listFormat":{}}, "characterFormat":{"fontSize":12, "fontFamily":"Cali
bri Light", "fontColor":"#1F3763"}, "basedOn":"Normal", "link":"Heading 3
Char", "next":"Normal"}, {"name":"Heading 3
Char", "type":"Character", "characterFormat":{"fontSize":12, "fontFamily":"Cali
bri Light", "fontColor":"#1F3763"}, "basedOn":"Default Paragraph
Font"}, {"name":"Heading
4", "type":"Paragraph", "paragraphFormat":{"leftIndent":0, "rightIndent":0, "fir
stLineIndent":0, "textAlignment":"Left", "beforeSpacing":2, "afterSpacing":0, "l
ineSpacing":1.0791666507720947, "lineSpacingType":"Multiple", "outlineLevel":"
Level4", "listFormat":{}}, "characterFormat":{"italic":true, "fontFamily":"Cali
bri Light", "fontColor":"#2F5496"}, "basedOn":"Normal", "link":"Heading 4
Char", "next":"Normal"}, {"name":"Heading 4
Char", "type":"Character", "characterFormat":{"italic":true, "fontFamily":"Cali
bri Light", "fontColor":"#2F5496"}, "basedOn":"Default Paragraph
Font"}, {"name":"Heading
5", "type":"Paragraph", "paragraphFormat":{"leftIndent":0, "rightIndent":0, "fir
stLineIndent":0, "textAlignment":"Left", "beforeSpacing":2, "afterSpacing":0, "l
ineSpacing":1.0791666507720947, "lineSpacingType":"Multiple", "outlineLevel":"
Level5", "listFormat":{}}, "characterFormat":{"fontFamily":"Calibri
Light", "fontColor":"#2F5496"}, "basedOn":"Normal", "link":"Heading 5
Char", "next":"Normal"}, {"name":"Heading 5
Char", "type":"Character", "characterFormat":{"fontFamily":"Calibri
Light", "fontColor":"#2F5496"}, "basedOn":"Default Paragraph

```

```

Font"}, {"name": "Heading
6", "type": "Paragraph", "paragraphFormat": {"leftIndent": 0, "rightIndent": 0, "fir
stLineIndent": 0, "textAlignment": "Left", "beforeSpacing": 2, "afterSpacing": 0, "l
ineSpacing": 1.0791666507720947, "lineSpacingType": "Multiple", "outlineLevel": "
Level6", "listFormat": {}}, "characterFormat": {"fontFamily": "Calibri
Light", "fontColor": "#1F3763"}, "basedOn": "Normal", "link": "Heading 6
Char", "next": "Normal"}, {"name": "Heading 6
Char", "type": "Character", "characterFormat": {"fontFamily": "Calibri
Light", "fontColor": "#1F3763"}, "basedOn": "Default Paragraph
Font"}], "lists": [], "abstractLists": [], "comments": [], "revisions": [], "customXm
l": []}';

    // showSpinner() will make the spinner visible
    showSpinner(document.getElementById('container'));
    // Open the default document
    this.$refs.container.ej2Instances.documentEditor.open(data);
    setInterval(function() {
        // hideSpinner() method used hide spinner
        hideSpinner(document.getElementById('container'));
    }, 5000);
},
onCreated: function () {
    createSpinner({
        // Specify the target for the spinner to show
        target: document.getElementById('container')
    });
}
},
};
</script>
<style>
    @import '../node_modules/@syncfusion/ej2-base/styles/material.css';
    @import '../node_modules/@syncfusion/ej2-buttons/styles/material.css';
    @import '../node_modules/@syncfusion/ej2-inputs/styles/material.css';
    @import '../node_modules/@syncfusion/ej2-popups/styles/material.css';
    @import '../node_modules/@syncfusion/ej2-lists/styles/material.css';
    @import '../node_modules/@syncfusion/ej2-
navigations/styles/material.css';
    @import '../node_modules/@syncfusion/ej2-
splitbuttons/styles/material.css';
    @import '../node_modules/@syncfusion/ej2-
dropdowns/styles/material.css';
    @import "../node_modules/@syncfusion/ej2-vue-
documenteditor/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/document-editor/spinner-cs1" %}

Note: In above example, we have used setInterval to hide spinner, just for demo purpose.

### [Resize document editor in Vue Document editor component](#)

In this article, we are going to see how to change height and width of Documenteditor.

#### [Change height of Document Editor](#)

DocumentEditorContainer initially render with default height. You can change height of documenteditor using [height](#) property, the value which is in pixel.

The following example code illustrates how to change height of Document Editor.

,

```
<ejs-documenteditorcontainer ref='container' :serviceUrl='serviceUrl' height="590px" id='container'
:enableToolbar='true'></ejs-documenteditorcontainer>
```

,

Similarly, you can use [height](#) property for DocumentEditor also.

#### *Change width of Document Editor*

DocumentEditorContainer initially render with default width. You can change width of documenteditor using [width](#) property, the value which is in percent.

The following example code illustrates how to change width of Document Editor.

,

```
<ejs-documenteditorcontainer ref='container' :serviceUrl='serviceUrl' v-on:created="onCreated"
width="100%" id='container' :enableToolbar='true'></ejs-documenteditorcontainer>
```

,

Similarly, you can use [width](#) property for DocumentEditor also.

#### *Resize Document Editor*

Using [resize](#) method, you change height and width of Document editor.

The following example code illustrates how to fit Document Editor to browser window size.

,

```
<template>
<div id="app">

<ejs-documenteditorcontainer ref='container' :serviceUrl='serviceUrl' v-on:created="onCreated"
height="590px" id='container' :enableToolbar='true'></ejs-documenteditorcontainer>

</div>
</template>
<script>
import Vue from 'vue';

import { DocumentEditorContainerPlugin, DocumentEditorContainerComponent,Toolbar} from
'@syncfusion/ej2-vue-documenteditor';

Vue.use(DocumentEditorContainerPlugin);

export default {
data() {
return {
serviceUrl:
'https://ej2services.syncfusion.com/production/web-services/api/documenteditor/',
```

```

};
},
provide: {
  //Inject require modules.
  DocumentEditorContainer: [Toolbar],
},
methods: {
  onWindowResize: function () {
    //Resizes the document editor component to fit full browser window automatically whenever the
    browser resized.
    this.updateDocumentEditorSize();
  },
  updateDocumentEditorSize: function () {
    //Resizes the document editor component to fit full browser window.
    var windowWidth = window.innerWidth;
    var windowHeight = window.innerHeight;
    this.$refs.container.ej2Instances.resize(windowWidth, windowHeight);
  },
  onCreated: function () {
    setInterval(() => {
      this.updateDocumentEditorSize();
    }, 100);
    //Adds event listener for browser window resize event.
    window.addEventListener('resize', this.onWindowResize.bind(this));
  },
},
};
</script>
`

```

### [Export document as pdf in Vue Document editor component](#)

In this article, we are going to see how to export the document as Pdf format. You can export the document as Pdf in following ways:

*Export the document as pdf in client-side*

Use [pdf export component](#) in application level to export the document as pdf using [exportasimage](#) API. Here, all pages will be converted to image and inserted as pdf pages(works like print as PDF). There is one limitation we can't search the text because we are exporting the pdf as image.

Note: You can install the pdf export packages from this [link](#).

The following example code illustrates how to export the document as pdf in client-side.

```
,
<template>
<div id="app">
<button id='export' v-on:click="onClick">Export</button>
<ejs-documenteditorcontainer ref='container' :serviceUrl='serviceUrl' height="590px" id='container'
:enableToolbar='true'></ejs-documenteditorcontainer>
</div>
</template>
<script>
import Vue from 'vue';
import {
PdfBitmap,
PdfDocument,
PdfPageOrientation,
PdfPageSettings,
PdfSection,
SizeF
} from '@syncfusion/ej2-pdf-export';
import { DocumentEditorContainerPlugin, DocumentEditorContainerComponent, Toolbar } from
'@syncfusion/ej2-vue-documenteditor';
Vue.use(DocumentEditorContainerPlugin);
export default {
data() {
return { serviceUrl:'https://ej2services.syncfusion.com/production/web-services/api/documenteditor/'
};
},
provide: {
//Inject require modules.
DocumentEditorContainer: [Toolbar]
```



```
},
methods: {
  onClick:function() {
    let obj = this;
    let pdfdocument = new PdfDocument();
    let count = obj.$refs.container.ej2Instances.documentEditor.pageCount;
    obj.$refs.container.ej2Instances.documentEditor.documentEditorSettings.printDevicePixelRatio = 2;
    let loadedPage = 0;
    for (let i = 1; i <= count; i++) {
      setTimeout(() => {
        let format = 'image/jpeg';
        // Getting pages as image
        let image = obj.$refs.container.ej2Instances.documentEditor.exportAsImage(i, format);
        image.onload = function () {
          let imageHeight = parseInt(
            image.style.height.toString().replace('px', '')
          );
          let imageWidth = parseInt(
            image.style.width.toString().replace('px', '')
          );
          let section = pdfdocument.sections.add();
          let settings = new PdfPageSettings(0);
          if (imageWidth > imageHeight) {
            settings.orientation = PdfPageOrientation.Landscape;
          }
          settings.size = new SizeF(imageWidth, imageHeight);
          (section).setPageSettings(settings);
          let page = section.pages.add();
          let graphics = page.graphics;
          let imageStr = image.src.replace('data:image/jpeg;base64,', '');
          let pdfImage = new PdfBitmap(imageStr);
          graphics.drawImage(pdfImage, 0, 0, imageWidth, imageHeight);
          loadedPage++;
        }
      }, 100);
    }
  }
}
```

```

if (loadedPage == count) {
  // Exporting the document as pdf
  pdfdocument.save(
    (obj.$refs.container.ej2Instances.documentEditor.documentName === ''
    ? 'sample'
    : obj.$refs.container.ej2Instances.documentEditor.documentName) + '.pdf'
  );
}
};
}, 500);
}
}
}
};
</script>
`

```

#### Export document as pdf in server-side using Syncfusion DocIO

With the help of [Synfusion DocIO](#), you can export the document as Pdf in server-side. Here, you can search the text.

The following way illustrates how to convert the document as Pdf:

- Using [serialize](#) API, convert the document as Sfdt and send it to server-side.

The following example code illustrates how to convert the document to sfdt and pass it to server-side.

```

`
<template>
<div id="app">
<button id='export' v-on:click="onClick">Export</button>
<ejs-documenteditorcontainer ref='container' :serviceUrl='serviceUrl' height="590px" id='container'
:enableToolbar='true'></ejs-documenteditorcontainer>
</div>
</template>
<script>
import Vue from 'vue';

import { DocumentEditorContainerPlugin, DocumentEditorContainerComponent, Toolbar } from
'@syncfusion/ej2-vue-documenteditor';

```

```

Vue.use(DocumentEditorContainerPlugin);
export default {
  data() {
    return { serviceUrl:'https://ej2services.syncfusion.com/production/web-services/api/documenteditor/'
  };
},
  provide: {
    //Inject require modules.
    DocumentEditorContainer: [Toolbar]
  },
  methods: {
    onClick:function() {
      let obj = this;
      let http = new XMLHttpRequest();
      // Replace your running web service Url here
      http.open('POST', 'http://localhost:62869/api/documenteditor/ExportPdf');
      http.setRequestHeader('Content-Type', 'application/json;charset=UTF-8');
      http.responseType = 'json';
      //Serialize document content as SFDT.
      let sfdt = { content: obj.$refs.container.ej2Instances.documentEditor.serialize() };
      //Send the sfdt content to server side.
      http.send(JSON.stringify(sfdt));
    }
  }
};
</script>

```

- Using Save API in server-side, you can convert the sfdt to stream.
- Finally, convert the stream to Pdf using [Syncfusion.DocIORenderer.Net.Core](#) library.

The following example code illustrates how to process the sfdt in server-side.

```

`c#
[AcceptVerbs("Post")]
[HttpPost]

```

```
[EnableCors("AllowAllOrigins")]
[Route("ExportPdf")]
public void ExportPdf([FromBody]SaveParameter data)
{
    // Converts the sfdt to stream
    Stream document = WordDocument.Save(data.content, FormatType.Docx);
    Syncfusion.DocIO.DLS.WordDocument doc = new Syncfusion.DocIO.DLS.WordDocument(document,
    Syncfusion.DocIO.FormatType.Docx);
    //Instantiation of DocIORenderer for Word to PDF conversion
    DocIORenderer render = new DocIORenderer();
    //Converts Word document into PDF document
    PdfDocument pdfDocument = render.ConvertToPDF(doc);
    // Saves the document to server machine file system, you can customize here to save into databases or
    file servers based on requirement.
    FileStream fileStream = new FileStream("sample.pdf", FileMode.OpenOrCreate, FileAccess.ReadWrite);
    //Saves the PDF file
    pdfDocument.Save(fileStream);
    pdfDocument.Close();
    fileStream.Close();
    document.Close();
}
```

Get the complete working sample in this [link](#).

Customize font family drop down in Vue Document editor component

Document editor provides an options to customize the font family drop down list values using [fontfamilies](#) in Document editor settings. Fonts which are added in fontFamilies of [documentEditorSettings](#) will be displayed on font drop down list of text properties pane and font dialog.

Similarly, you can use [documentEditorSettings](#) property for DocumentEditor also.

The following example code illustrates how to change the font families in Document editor container.

```
<template>
<div id="app">
<ejs-documenteditorcontainer ref='documenteditor' :serviceUrl='serviceUrl'
:documentEditorSettings='settings' height="590px" id='container' :enableToolbar='true'></ejs-
documenteditorcontainer>
```

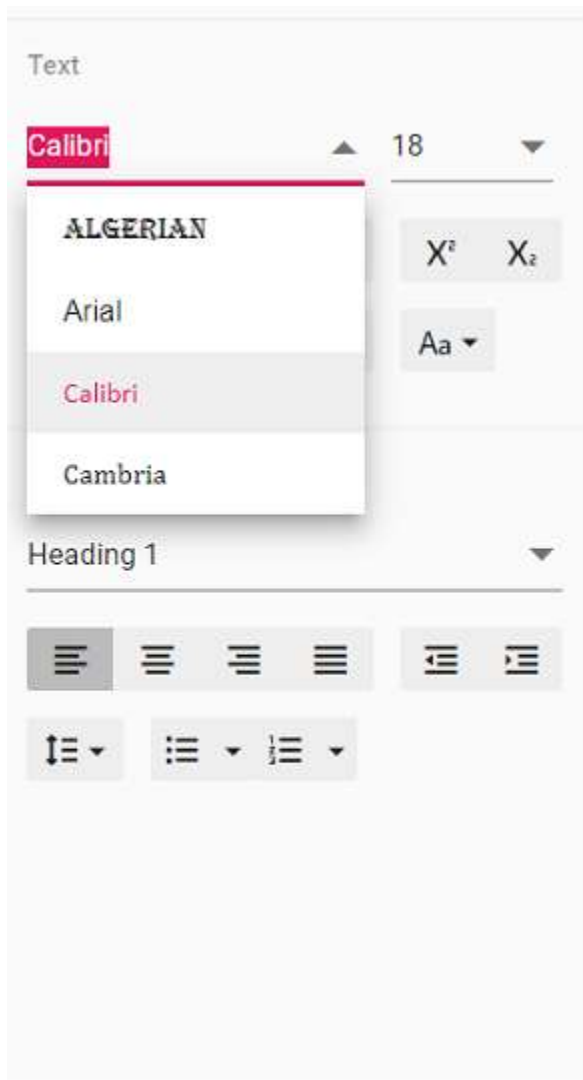
```
</div>
</template>
<script>
import Vue from 'vue';

import { DocumentEditorContainerPlugin, DocumentEditorContainerComponent, Toolbar } from
 '@syncfusion/ej2-vue-documenteditor';

Vue.use(DocumentEditorContainerPlugin);

export default {
  data() {
    return { serviceUrl: 'https://ej2services.syncfusion.com/production/web-services/api/documenteditor/',
      // Add required font families to list it in font drop down
      settings: { fontFamilies: ['Algerian', 'Arial', 'Calibri', 'Cambria'] } },
  },
  provide: {
    //Inject require modules.
    DocumentEditorContainer: [Toolbar]
  }
}
</script>
`
```

Output will be like below:



#### [Auto save document in document editor in Vue Document editor component](#)

In this article, we are going to see how to autosave the document in AWS S3. You can automatically save the edited content in regular intervals of time. It helps reduce the risk of data loss by saving an open document automatically at customized intervals.

The following example illustrates how to auto save the document in AWS S3.

- In the client-side, using content change event, we can automatically save the edited content in regular intervals of time. Based on `contentChanged` boolean, the document send as Docx format to server-side using [saveAsBlob](#) method.

<template>

<div id="app">

```
<ejs-documenteditorcontainer ref='container' :serviceUrl='serviceUrl' v-on:created="onCreated" v-
on:contentChange="contentChangeEvent" height="590px" id='container' :enableToolbar='true'></ejs-
documenteditorcontainer>

</div>

</template>

<script>

import Vue from 'vue';

import { DocumentEditorContainerPlugin, DocumentEditorContainerComponent, Toolbar } from
 '@syncfusion/ej2-vue-documenteditor';

Vue.use(DocumentEditorContainerPlugin);

export default {
  data() {
    return { serviceUrl:'https://ej2services.syncfusion.com/production/web-services/api/documenteditor/',
      contentChanged:false};
  },
  provide: {
    //Inject require modules.
    DocumentEditorContainer: [Toolbar]
  },
  methods: {
    contentChangeEvent: function () {
      this.contentChanged = true;
    },
    onCreated: function () {
      setInterval(() => {
        if (this.contentChanged) {
          //You can save the document as below
          this.$refs.container.ej2Instances.documentEditor
            .saveAsBlob('Docx')
            .then((blob) => {
              console.log('Saved sucessfully');
              let exportedDocument = blob;
              //Now, save the document where ever you want.
              let formData = new FormData();
```

```

formData.append('fileName', 'sample.docx');
formData.append('data', exportedDocument);
/ tslint:disable /
var req = new XMLHttpRequest();
// Replace your running Url here
req.open(
'POST',
'http://localhost:62869/api/documenteditor/SaveToS3',
true
);
req.onreadystatechange = () => {
if (req.readyState === 4) {
if (req.status === 200 || req.status === 304) {
console.log('Saved sucessfully');
}
}
};
req.send(formData);
});
this.contentChanged = false;
}
}, 1000);
},
},
};
</script>
`

```

- In server-side, configure the access key and secret key in `web.config` file and register profile in `startup.cs`.

In `web.config`, add key like below format:

```

`c#
<appSettings>

```



```
<add key="AWSProfileName" value="sync_development" />
<add key="AWSAccessKey" value="" />
<add key="AWSSecretKey" value="" />
</appSettings>
```

In `startup.cs`, register profile in below format:

```
`c#
Amazon.Util.ProfileManager.RegisterProfile("sync_development", "", "");
```

- In server-side, Receives the stream content from client-side and process it to save the document in aws s3. Add Web API in controller file like below to save the document in aws s3.

```
`c#
[AcceptVerbs("Post")]
[HttpPost]
[EnableCors("AllowAllOrigins")]
[Route("SaveToS3")]
public string SaveToS3()
{
    IFormFile file = HttpContext.Request.Form.Files[0];
    Stream stream = new MemoryStream();
    file.CopyTo(stream);
    UploadFileStreamToS3(stream, "documenteditor", "", "GettingStarted.docx");
    stream.Close();
    return "Sucess";
}

public bool UploadFileStreamToS3(System.IO.Stream localFilePath, string bucketName, string
subDirectoryInBucket, string fileNameInS3)
{
    AWSCredentials credentials = new StoredProfileAWSCredentials("sync_development");
    IAmazonS3 client = new AmazonS3Client(credentials, Amazon.RegionEndpoint.USEast1);
    TransferUtility utility = new TransferUtility(client);
    TransferUtilityUploadRequest request = new TransferUtilityUploadRequest();
    if (subDirectoryInBucket == "" || subDirectoryInBucket == null)
```

```

{
request.BucketName = bucketName; //no subdirectory just bucket name
}
else
{ // subdirectory and bucket name
request.BucketName = bucketName + @"/" + subDirectoryInBucket;
}
request.Key = fileNameInS3; //file name up in S3
request.InputStream = localFilePath;
utility.Upload(request); //commencing the transfer
return true; //indicate that the file was sent
}
`

```

Get the complete working sample in this [link](#).

Retrieve the bookmark content as text in Vue Document editor component

You can get the bookmark or whole document content from the Vue Document Editor component as plain text and SFDT (rich text).

*Get the bookmark content as plain text*

You can [selectBookmark](#) API to navigate to the bookmark and use [text](#) API to get the bookmark content as plain text from Vue Document Editor component.

The following example code illustrates how to get the bookmark content as plain text.

```

<template>
<div id="app">
<ejs-documenteditorcontainer
ref="container"
:serviceUrl="serviceUrl"
height="590px"
id="container"
:enableToolbar="true"
v-on:created="onCreated.bind(this)"
</ejs-documenteditorcontainer>
</div>
</template>

```

```

<script>
import Vue from 'vue';
import {
  DocumentEditorContainerPlugin,
  DocumentEditorContainerComponent,
  Toolbar,
} from '@syncfusion/ej2-vue-documenteditor';
Vue.use(DocumentEditorContainerPlugin);
export default {
  data() {
    return {
      serviceUrl:
        'https://ej2services.syncfusion.com/production/web-services/api/documenteditor/',
    };
  },
  provide: {
    //Inject require modules.
    DocumentEditorContainer: [Toolbar],
  },
  methods: {
    onCreated: function () {
      // To insert text in cursor position
      this.$refs.container.ej2Instances.documentEditor.editor.insertText('Document editor');
      // To select all the content in document
      this.$refs.container.ej2Instances.documentEditor.selection.selectAll();
      // Insert bookmark to selected content
      this.$refs.container.ej2Instances.documentEditor.editor.insertBookmark('Bookmark1');
      // Provide your bookmark name to navigate to specific bookmark
      this.$refs.container.ej2Instances.documentEditor.selection.selectBookmark('Bookmark1');
      // To get the selected content as text
      let selectedContent = this.$refs.container.ej2Instances.documentEditor.selection.text;
    }
  }
}

```

```
};
</script>
`
```

To get the bookmark content as SFDT (rich text), please check this [link](#)

*Get the whole document content as text*

You can use [text](#) API to get the whole document content as plain text from Vue Document Editor component.

The following example code illustrates how to get the whole document content as plain text.

```
`
<template>
<div id="app">
<ejs-documenteditorcontainer
ref="container"
:serviceUrl="serviceUrl"
height="590px"
id="container"
:enableToolbar="true"
v-on:created="onCreated.bind(this)"
</ejs-documenteditorcontainer>
</div>
</template>
<script>
import Vue from 'vue';
import {
DocumentEditorContainerPlugin,
DocumentEditorContainerComponent,
Toolbar,
} from '@syncfusion/ej2-vue-documenteditor';
Vue.use(DocumentEditorContainerPlugin);
export default {
data() {
return {
serviceUrl:
'https://ej2services.syncfusion.com/production/web-services/api/documenteditor/',

```

```

};
},
provide: {
  //Inject require modules.
  DocumentEditorContainer: [Toolbar],
},
methods: {
  onCreated: function () {
    // To insert text in cursor position
    this.$refs.container.ej2Instances.documentEditor.editor.insertText('Document editor');
    // To select all the content in document
    this.$refs.container.ej2Instances.documentEditor.selection.selectAll();
    // To get the content as text
    let selectedContent = this.$refs.container.ej2Instances.documentEditor.selection.text;
  }
}
};
</script>

```

#### *Get the whole document content as SFDT(rich text)*

You can use [serialize](#) API to get the whole document content as SFDT string from Vue Document Editor component.

The following example code illustrates how to get the whole document content as SFDT.

```

<template>
<div id="app">
<ejs-documenteditorcontainer
  ref="container"
  :serviceUrl="serviceUrl"
  height="590px"
  id="container"
  :enableToolbar="true"
  v-on:created="onCreated.bind(this)"
</ejs-documenteditorcontainer>

```

```
</div>
</template>
<script>
import Vue from 'vue';
import {
  DocumentEditorContainerPlugin,
  DocumentEditorContainerComponent,
  Toolbar,
} from '@syncfusion/ej2-vue-documenteditor';
Vue.use(DocumentEditorContainerPlugin);
export default {
  data() {
    return {
      serviceUrl:
        'https://ej2services.syncfusion.com/production/web-services/api/documenteditor/',
    };
  },
  provide: {
    //Inject require modules.
    DocumentEditorContainer: [Toolbar],
  },
  methods: {
    onCreated: function () {
      // To insert text in cursor position
      this.$refs.container.ej2Instances.documentEditor.editor.insertText('Document editor');
      // To get the content as SFDT
      let selectedContent: string = this.$refs.container.ej2Instances.documentEditor.serialize();
    }
  }
};
</script>
,
```

*Get the header content as text*

You can use [goToHeader](#) API to navigate the selection to the header and then use [text](#) API to get the content as plain text.

The following example code illustrates how to get the header content as plain text.

```
,
<template>
<div id="app">
<ejs-documenteditorcontainer
ref="container"
:serviceUrl="serviceUrl"
height="590px"
id="container"
:enableToolbar="true"
v-on:created="onCreated.bind(this)"
</ejs-documenteditorcontainer>
</div>
</template>
<script>
import Vue from 'vue';
import {
DocumentEditorContainerPlugin,
DocumentEditorContainerComponent,
Toolbar,
} from '@syncfusion/ej2-vue-documenteditor';
Vue.use(DocumentEditorContainerPlugin);
export default {
data() {
return {
serviceUrl:
'https://ej2services.syncfusion.com/production/web-services/api/documenteditor/',
};
},
provide: {
//Inject require modules.
```

```

DocumentEditorContainer: [Toolbar],
},
methods: {
  onCreated: function () {
    // To navigate the selection to header
    this.$refs.container.ej2Instances.documentEditor.selection.goToHeader();
    // To insert text in cursor position
    this.$refs.container.ej2Instances.documentEditor.editor.insertText('Document editor');
    // To select all the content in document
    this.$refs.container.ej2Instances.documentEditor.selection.selectAll();
    // To get the selected content as text
    let selectedContent = this.$refs.container.ej2Instances.documentEditor.selection.text;
  }
}
};
</script>
`

```

Similarly, you can use [goToFooter](#) API to navigate the selection to the footer and then use [text](#) API to get the content as plain text.

#### Get current word in Vue Document editor component

You can get the current word or paragraph content from the Vue Document Editor component as plain text and SFDT (rich text).

##### Select and get the word in current cursor position

You can use [selectCurrentWord](#) API in selection module to select the current word at cursor position and use [text](#) API to get the selected content as plain text from Vue Document Editor component.

The following example code illustrates how to select and get the current word as plain text.

```

<template>
<div id="app">
<ejs-documenteditorcontainer
ref="container"
:serviceUrl="serviceUrl"
height="590px"
id="container"
:enableToolbar="true"

```



```

v-on:created="onCreated.bind(this)"
</ejs-documenteditorcontainer>
</div>
</template>
<script>
import Vue from 'vue';
import {
  DocumentEditorContainerPlugin,
  DocumentEditorContainerComponent,
  Toolbar,
} from '@syncfusion/ej2-vue-documenteditor';
Vue.use(DocumentEditorContainerPlugin);
export default {
  data() {
    return {
      serviceUrl:
        'https://ej2services.syncfusion.com/production/web-services/api/documenteditor/',
    };
  },
  provide: {
    //Inject require modules.
    DocumentEditorContainer: [Toolbar],
  },
  methods: {
    onCreated: function () {
      // To insert text in cursor position
      this.$refs.container.ej2Instances.documentEditor.editor.insertText(
        'Document editor'
      );
      // Move selection to previous character
      this.$refs.container.ej2Instances.documentEditor.selection.moveToPreviousCharacter();
      // To select the current word in document
      this.$refs.container.ej2Instances.documentEditor.selection.selectCurrentWord();
    }
  }
}

```

```
// To get the selected content as text
var selectedContent =
this.$refs.container.ej2Instances.documentEditor.selection.text;
}
}
};
</script>
`
```

To get the bookmark content as SFDT (rich text), please check this [link](#)

*Select and get the paragraph in current cursor position*

You can use [selectParagraph](#) API in selection module to select the current paragraph at cursor position and use [text](#) API or [sfdt](#) API to get the selected content as plain text or SFDT from Vue Document Editor component.

The following example code illustrates how to select and get the current paragraph as SFDT.

```
<template>
<div id="app">
<ejs-documenteditorcontainer
ref="container"
:serviceUrl="serviceUrl"
height="590px"
id="container"
:enableToolbar="true"
v-on:created="onCreated.bind(this)"
</ejs-documenteditorcontainer>
</div>
</template>
<script>
import Vue from 'vue';
import {
DocumentEditorContainerPlugin,
DocumentEditorContainerComponent,
Toolbar,
} from '@syncfusion/ej2-vue-documenteditor';
```

```

Vue.use(DocumentEditorContainerPlugin);
export default {
  data() {
    return {
      serviceUrl:
        'https://ej2services.syncfusion.com/production/web-services/api/documenteditor/',
    };
  },
  provide: {
    //Inject require modules.
    DocumentEditorContainer: [Toolbar],
  },
  methods: {
    onCreated: function () {
      // To insert text in cursor position
      this.$refs.container.ej2Instances.documentEditor.editor.insertText('Document editor');
      // To select the current paragraph in document
      this.$refs.container.ej2Instances.documentEditor.selection.selectParagraph();
      // To get the selected content as SFDT
      let selectedContent = this.$refs.container.ej2Instances.documentEditor.selection.sfDT;
    }
  }
};
</script>

```

### Insert page number and navigate to page in Vue Document editor component

You can insert page number and navigate to specific page in Vue Document Editor component by following ways.

#### Insert page number

You can use [insertPageNumber](#) API in editor module to insert the page number in current cursor position. By default, Page number will insert in Arabic number style. You can change it, by providing the number style in parameter.

Note: Currently, Documenteditor have options to insert page number at current cursor position.

The following example code illustrates how to insert page number in header.

,

```
<template>
<div id="app">
<ejs-documenteditorcontainer
ref="container"
:serviceUrl="serviceUrl"
height="590px"
id="container"
:enableToolbar="true"
v-on:created="onCreated.bind(this)"
</ejs-documenteditorcontainer>
</div>
</template>
<script>
import Vue from 'vue';
import {
  DocumentEditorContainerPlugin,
  DocumentEditorContainerComponent,
  Toolbar,
} from '@syncfusion/ej2-vue-documenteditor';
Vue.use(DocumentEditorContainerPlugin);
export default {
  data() {
    return {
      serviceUrl:
        'https://ej2services.syncfusion.com/production/web-services/api/documenteditor/',
    };
  },
  provide: {
    //Inject require modules.
    DocumentEditorContainer: [Toolbar],
  },
  methods: {
    onCreated: function () {
```

```
// To insert text in cursor position
this.$refs.container.ej2Instances.documentEditor.editor.insertText('Document editor');

// To move the selection to header
this.$refs.container.ej2Instances.documentEditor.selection.goToHeader();

// Insert page number in the current cursor position
this.$refs.container.ej2Instances.documentEditor.editor.insertPageNumber();
}
}
};
</script>
,
```

Also, you use [insertField](#) API in Editor module to insert the Page number in current position

,

```
//Current page number
this.$refs.container.ej2Instances.documentEditor.editor.insertField('PAGE * MERGEFORMAT', '1');
,
```

#### *Get page count*

You can use [pageCount](#) API to gets the total number of pages in Document.

The following example code illustrates how to get the number of pages in Document.

,

```
<template>
<div id="app">
<ejs-documenteditorcontainer
ref="container"
:serviceUrl="serviceUrl"
height="590px"
id="container"
:enableToolbar="true"
v-on:created="onCreated.bind(this)"
</ejs-documenteditorcontainer>
</div>
</template>
<script>
```

```
import Vue from 'vue';
import {
  DocumentEditorContainerPlugin,
  DocumentEditorContainerComponent,
  Toolbar,
} from '@syncfusion/ej2-vue-documenteditor';
Vue.use(DocumentEditorContainerPlugin);
export default {
  data() {
    return {
      serviceUrl:
        'https://ej2services.syncfusion.com/production/web-services/api/documenteditor/',
    };
  },
  provide: {
    //Inject require modules.
    DocumentEditorContainer: [Toolbar],
  },
  methods: {
    onCreated: function () {
      // To insert text in cursor position
      this.$refs.container.ej2Instances.documentEditor.editor.insertText('Document editor');
      // To get the total number of pages
      let pageCount = this.$refs.container.ej2Instances.documentEditor.pageCount;
    }
  }
};
</script>
```

#### *Navigate to specific page*

You can use [goToPage](#) API in Selection module to move selection to the start of the specified page number.

The following example code illustrates how to move selection to specific page.

```
<template>
<div id="app">
<ejs-documenteditorcontainer
ref="container"
:serviceUrl="serviceUrl"
height="590px"
id="container"
:enableToolbar="true"
v-on:created="onCreated.bind(this)"
</ejs-documenteditorcontainer>
</div>
</template>
<script>
import Vue from 'vue';
import {
  DocumentEditorContainerPlugin,
  DocumentEditorContainerComponent,
  Toolbar,
} from '@syncfusion/ej2-vue-documenteditor';
Vue.use(DocumentEditorContainerPlugin);
export default {
  data() {
    return {
      serviceUrl:
        'https://ej2services.syncfusion.com/production/web-services/api/documenteditor/',
    };
  },
  provide: {
    //Inject require modules.
    DocumentEditorContainer: [Toolbar],
  },
  methods: {
    onCreated: function () {
```

```
// To move selection to page number 2
this.$refs.container.ej2Instances.documentEditor.selection.goToPage(2);
}
}
};
</script>
`
```

Move selection to specific position in Vue Document editor component

Using [select](#) API in selection module, You can set cursor position to anywhere in the document.

*Selects content based on start and end hierarchical index*

Hierarchical index will be in below format.

**sectionIndex;blockIndex;offset**

The following code snippet illustrate how to select using hierarchical index.

```
// Selection will occur between provided start and end offset
this.$refs.container.ej2Instances..documentEditor.editor.insertText("Welcome");
// The below code will select the letters "We" from inserted text "Welcome"
this.$refs.container.ej2Instances..documentEditor.selection.select("0;0;0", "0;0;2");
`
```

The following table illustrates about Hierarchical index in detail.

Element	Hierarchical Format	Explanation
---------	---------------------	-------------

----- ----- ----		
------------------	--	--

Move selection to Paragraph	sectionIndex;blockIndex;offset	 <b>Ex:</b> 0;0;0	It moves the cursor to the start of paragraph.
-----------------------------	--------------------------------	----------------------	------------------------------------------------

Move selection to Table	sectionIndex;tableIndex;rowIndex;cellIndex;blockIndex;offset	 <b>Ex:</b> 0;0;0;1;0	It moves the cursor to the second paragraph which is inside first row and cell of table.
-------------------------	--------------------------------------------------------------	--------------------------	------------------------------------------------------------------------------------------

Move selection to header	pageIndex;H;sectionIndex;blockIndex;offset	 <b>Ex:</b> 1;H;0;0;0	It moves cursor to the header in second page.
--------------------------	--------------------------------------------	--------------------------	-----------------------------------------------

Move selection to Footer	pageIndex;F;sectionIndex;blockIndex;offset	 <b>Ex:</b> 1;F;0;0;0	It moves cursor to the footer in second page.
--------------------------	--------------------------------------------	--------------------------	-----------------------------------------------

*Get the selection start and end hierarchical index*

Using [startOffset](#), you can get start hierarchical index which denotes the start index of current selection.

Similarly, using [endOffset](#), you can get end hierarchical index which denotes the end index of current selection.

The following code snippet illustrate how to get the selection start and end offset on selection changes in document.



```
,  
  
<template>  
<div id="app">  
<ejs-documenteditorcontainer  
  ref="container"  
  :serviceUrl="serviceUrl"  
  height="590px"  
  id="container"  
  :enableToolbar="true"  
  v-on:selectionChange="selectionChanged.bind(this)"  
</ejs-documenteditorcontainer>  
</div>  
</template>  
<script>  
import Vue from 'vue';  
import {  
  DocumentEditorContainerPlugin,  
  DocumentEditorContainerComponent,  
  Toolbar,  
} from '@syncfusion/ej2-vue-documenteditor';  
Vue.use(DocumentEditorContainerPlugin);  
export default {  
  data() {  
    return {  
      serviceUrl:  
        'https://ej2services.syncfusion.com/production/web-services/api/documenteditor/',  
    };  
  },  
  provide: {  
    //Inject require modules.  
    DocumentEditorContainer: [Toolbar],  
  },  
  methods: {
```

```

selectionChanged: function () {
//Get the start index of current selection
let startOffset =
this.$refs.container.ej2Instances.documentEditor.selection.startOffset;
//Get the end index of current selection
let endOffset = this.$refs.container.ej2Instances.documentEditor.selection.endOffset;
}
}
};
</script>
`

```

Document editor have [selectionChange](#) event which is triggered whenever the selection changes in Document.

*Selects the content based on left and top position*

Here, you can specify the [selection settings](#) to select the content based on left and top position.

x denotes the left position and y denotes the top position and extend denotes whether to extend or update selection.

Please check below code sample for reference.

```

this.$refs.container.ej2Instances.documentEditor.selection.select({ x: 188.4814208984375 , y:
662.00005, extend: true });
`

```

[Disable header and footer edit in document editor in Vue Document editor component](#)

*Disable header and footer edit in DocumentEditorContainer instance*

You can use [restrictEditing](#) property to disable header and footer editing based on selection context type.

RestrictEditing allows you to restrict the document modification and makes the Document read only mode. So, by using this property, and if selection inside header or footer, you can set this property as true.

The following example code illustrates how to header and footer edit in `DocumentEditorContainer` instance.

```

<template>
<div id="app">
<ejs-documenteditorcontainer
ref="container"

```

```

:serviceUrl="serviceUrl"
height="590px"
id="container"
:enableToolbar="true"
v-on:selectionChange="selectionChanged.bind(this)"
</ejs-documenteditorcontainer>
</div>
</template>
<script>
import Vue from 'vue';
import {
  DocumentEditorContainerPlugin,
  DocumentEditorContainerComponent,
  Toolbar,
} from '@syncfusion/ej2-vue-documenteditor';
Vue.use(DocumentEditorContainerPlugin);
export default {
  data() {
    return {
      serviceUrl:
        'https://ej2services.syncfusion.com/production/web-services/api/documenteditor/',
    };
  },
  provide: {
    //Inject require modules.
    DocumentEditorContainer: [Toolbar],
  },
  methods: {
    selectionChanged: function () {
      // Check whether selection is in header
      if (this.$refs.container.ej2Instances.documentEditor.selection.contextType.indexOf('Header') > -1 ||
        // Check whether selection is in Footer
        this.$refs.container.ej2Instances.documentEditor.selection.contextType.indexOf('Footer') > -1 ) {

```

```
// Change the document to read only mode
this.$refs.container.ej2Instances.restrictEditing = true;
} else {
// Change the document to editable mode
this.$refs.container.ej2Instances.restrictEditing = false;
}
}
};
</script>
```

Otherwise, you can disable clicking inside Header or Footer by using [closeHeaderFooter](#) API in selection module.

The following example code illustrates how to close header and footer when selection is inside header or footer in `DocumentEditorContainer` instance.

```
<template>
<div id="app">
<ejs-documenteditorcontainer
ref="container"
:serviceUrl="serviceUrl"
height="590px"
id="container"
:enableToolbar="true"
v-on:selectionChange="selectionChanged.bind(this)"
</ejs-documenteditorcontainer>
</div>
</template>
<script>
import Vue from 'vue';
import {
DocumentEditorContainerPlugin,
DocumentEditorContainerComponent,
Toolbar,
```

```

} from '@syncfusion/ej2-vue-documenteditor';
Vue.use(DocumentEditorContainerPlugin);
export default {
  data() {
    return {
      serviceUrl:
        'https://ej2services.syncfusion.com/production/web-services/api/documenteditor/',
    };
  },
  provide: {
    //Inject require modules.
    DocumentEditorContainer: [Toolbar],
  },
  methods: {
    selectionChanged: function () {
      // Check whether selection is in header
      if (this.$refs.container.ej2Instances.documentEditor.selection.contextType.indexOf('Header') > -1 ||
        // Check whether selection is in Footer
        this.$refs.container.ej2Instances.documentEditor.selection.contextType.indexOf('Footer') > -1 ) {
        // Close header Footer
        this.$refs.container.ej2Instances.documentEditor.selection.closeHeaderFooter();
      }
    }
  }
};
</script>

```

#### *Disable header and footer edit in DocumentEditor instance*

Like restrictEditing, you can use [isReadOnly](#) property in Document editor to disable header and footer edit.

The following example code illustrates how to header and footer edit in **DocumentEditor** instance.

```

<template>
<div id="app">

```

```

<ejs-documenteditor :serviceUrl='serviceUrl' ref='documentEditor' :isReadOnly='false'
:enablePrint='true' :enableSfdtExport='true' :enableSelection='true' :enableContextMenu='true'
:enableSearch='true' :enableOptionsPane='true' :enableWordExport='true' :enableTextExport='true'
:enableEditor='true' :enableImageResizer='true' :enableEditorHistory='true'
:enableHyperlinkDialog='true' :enableTableDialog='true' :enableBookmarkDialog='true'
:enableTableOfContentsDialog='true' :enablePageSetupDialog='true' :enableStyleDialog='true'
:enableListDialog='true' :enableParagraphDialog='true' :enableFontDialog='true'
:enableTablePropertiesDialog='true' :enableBordersAndShadingDialog='true'
:enableTableOptionsDialog='true' height="370px" v-on:selectionChange="selectionChanged.bind(this)">
</ejs-documenteditor>

</div>

</template>

<script>

import Vue from 'vue';

import { DocumentEditorPlugin, DocumentEditorComponent, Print, SfdtExport, WordExport, TextExport,
Selection, Search, Editor, ImageResizer, EditorHistory, ContextMenu, OptionsPane, HyperlinkDialog,
TableDialog, BookmarkDialog, TableOfContentsDialog, PageSetupDialog, StyleDialog, ListDialog,
ParagraphDialog, BulletsAndNumberingDialog, FontDialog, TablePropertiesDialog,
BordersAndShadingDialog, TableOptionsDialog, CellOptionsDialog, StylesDialog } from '@syncfusion/ej2-
vue-documenteditor';

Vue.use(DocumentEditorPlugin);

export default {

data () {

return {

serviceUrl:'https://ej2services.syncfusion.com/production/web-services/api/documenteditor/'

},

provide: {

//Inject require modules.

DocumentEditor: [Print, SfdtExport, WordExport, TextExport, Selection, Search, Editor, ImageResizer,
EditorHistory, ContextMenu, OptionsPane, HyperlinkDialog, TableDialog, BookmarkDialog,
TableOfContentsDialog, PageSetupDialog, StyleDialog, ListDialog, ParagraphDialog,
BulletsAndNumberingDialog, FontDialog, TablePropertiesDialog, BordersAndShadingDialog,
TableOptionsDialog, CellOptionsDialog, StylesDialog]

},

methods: {

selectionChanged: function () {

// Check whether selection is in header

if (this.$refs.documentEditor.ej2Instances.selection.contextType.indexOf('Header') > -1 ||

// Check whether selection is in Footer

```

```
this.$refs.documentEditor.ej2Instances.selection.contextType.indexOf('Footer') > -1) {
// Change the document to read only mode
this.$refs.documentEditor.ej2Instances.isReadOnly = true;
} else {
// Change the document to editable mode
this.$refs.documentEditor.ej2Instances.isReadOnly = false;
}
}
}
}
</script>
`
```

### Insert text in current position in Vue Document editor component

You can insert the text, paragraph and rich-text content in Vue Document Editor component.

#### *Insert text in current cursor position*

You can use [insertText](#) API in editor module to insert the text in current cursor position.

The following example code illustrates how to add the text in current selection.

```
`
// It will insert the provided text in current selection
this.$refs.container.ej2Instances.documentEditor.editor.insertText('Syncfusion');
`

<template>
<div id="app">
<div>
<button v-on:click='insertText'>Insert Text</button>
</div>
<ejs-documenteditorcontainer ref='container' height="590px" id='container'
:enableToolbar='true'></ejs-documenteditorcontainer>
</div>
</template>
<script>
import Vue from 'vue'
import { DocumentEditorContainerPlugin, Toolbar } from '@syncfusion/ej2-vue-documenteditor';
```

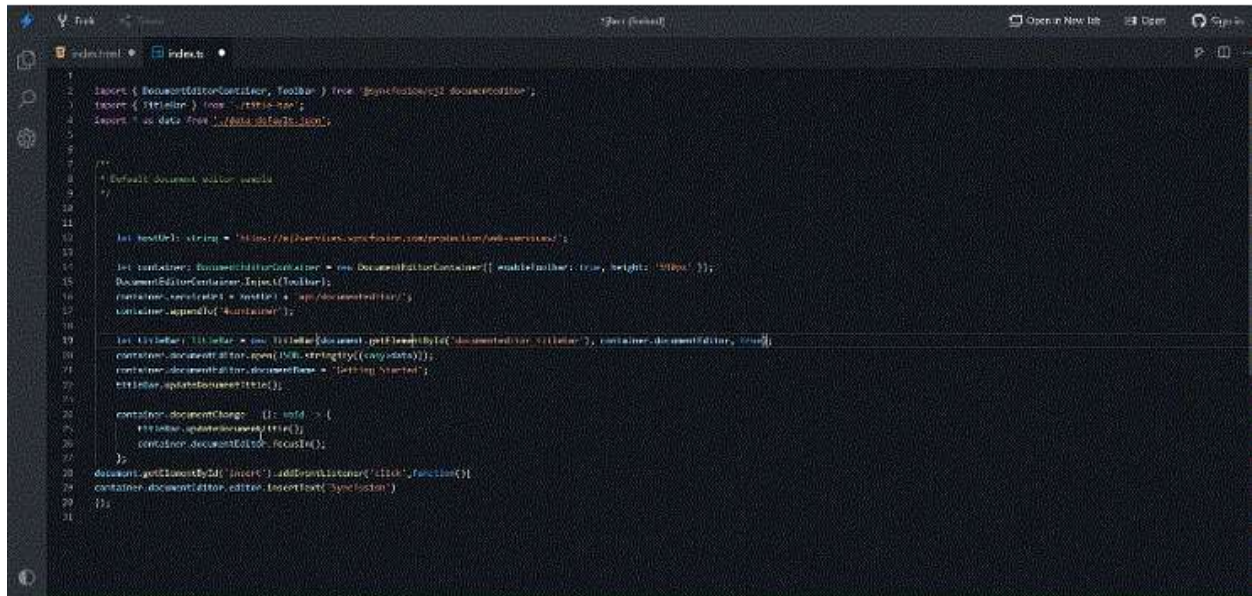
```

Vue.use(DocumentEditorContainerPlugin);
export default {
  data: function() {
    return {
    };
  },
  provide: {
    //Inject require modules.
    DocumentEditorContainer: [Toolbar],
  },
  methods: {
    insertText: function() {
      //It will insert the provided text in current selection
      this.$refs.container.ej2Instances.documentEditor.editor.insertText('Syncfusion');
    }
  }
}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-vue-documenteditor/styles/material.css";
</style>
`

```

Please check below gif which illustrates how to insert text in current cursor position on button click:





### Insert paragraph in current cursor position

To insert new paragraph at current selection, you can use [insertText](#) API with parameter as `\r\n` or `\n`.

The following example code illustrates how to add the new paragraph in current selection.

```

// It will add the new paragraph in current selection
this.$refs.container.ej2Instances.documentEditor.editor.insertText('\n');

```

### Insert the rich-text content

To insert the HTML content, you have to convert the HTML content to SFDT Format using [web service](#). Then use [paste](#) API to insert the sfdt at current cursor position.

Note: Html string should be wellformatted html. [DocIO](#) support only wellformatted XHTML.

The following example illustrates how to insert the HTML content at current cursor position.

- Send the HTML content to server side for SFDT conversion. Refer to the following example to send the HTML content to server side and inserting it in current cursor position.

```

<template>
<div id="app">
<ejs-documenteditorcontainer
ref="container"

```

```

:serviceUrl="serviceUrl"
height="590px"
id="container"
:enableToolbar="true"
v-on:created="onCreated.bind(this)"
</ejs-documenteditorcontainer>
</div>
</template>
<script>
import Vue from 'vue';
import {
  DocumentEditorContainerPlugin,
  DocumentEditorContainerComponent,
  Toolbar,
} from '@syncfusion/ej2-vue-documenteditor';
Vue.use(DocumentEditorContainerPlugin);
export default {
  data() {
    return {
      serviceUrl:
        'https://ej2services.syncfusion.com/production/web-services/api/documenteditor/',
    };
  },
  provide: {
    //Inject require modules.
    DocumentEditorContainer: [Toolbar]
  },
  methods: {
    onCreated: function () {
      let proxy = this;
      let htmltags =
        "<?xml version='1.0' encoding='utf - 8'?><!DOCTYPE html PUBLIC '-//W3C//DTD XHTML 1.0 Strict//EN'http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd><html xmlns='http://www.w3.org/1999/xhtml' xml:lang='en' lang='en'><body><h1>The img element</h1><img

```

```
src='https://www.w3schools.com/images/lamp.jpg' alt='Lamp Image' width='500'
height='600'/></body></html>";
let http = new XMLHttpRequest();
http.open('POST', 'http://localhost:5000/api/documenteditor/LoadString');
http.setRequestHeader('Content-Type', 'application/json;charset=UTF-8');
http.responseType = 'json';
http.onreadystatechange = function () {
if (http.readyState === 4) {
if (http.status === 200 || http.status === 304) {
// Insert the sfdt content in cursor position using paste API
proxy.container.documentEditor.editor.paste(http.response);
} else {
alert('failed;');
}
}
};
let htmlContent = { content: htmltags };
http.send(JSON.stringify(htmlContent));
}
}
};
</script>
`
```

- Please refer the following code example for server-side web implementation for HTML conversion using DocumentEditor.

```
`c#
//API controller for the conversion.
[HttpPost]
public string LoadString([FromBody]InputParameter data)
{
// You can also load HTML file/string from server side.
Syncfusion.EJ2.DocumentEditor.WordDocument document =
Syncfusion.EJ2.DocumentEditor.WordDocument.LoadString(data.content, FormatType.Html); // Convert
the HTML to SFDt format.
```

```

string json = Newtonsoft.Json.JsonConvert.SerializeObject(document);
document.Dispose();
return json;
}

public class InputParameter
{
    public string content {get; set; }
}

```

Note: The above example illustrates inserting HTML content. Similarly, you can insert any rich-text content by converting any of the supported file formats (DOCX, DOC, WordML, HTML, RTF) to SFDT.

#### [Change the cursor color in document editor in Vue Document editor component](#)

Document Editor default cursor color is black. The user can change the color by overriding the css property using class name. The Document editor cursor css have a class named `e-de-blink-cursor`.

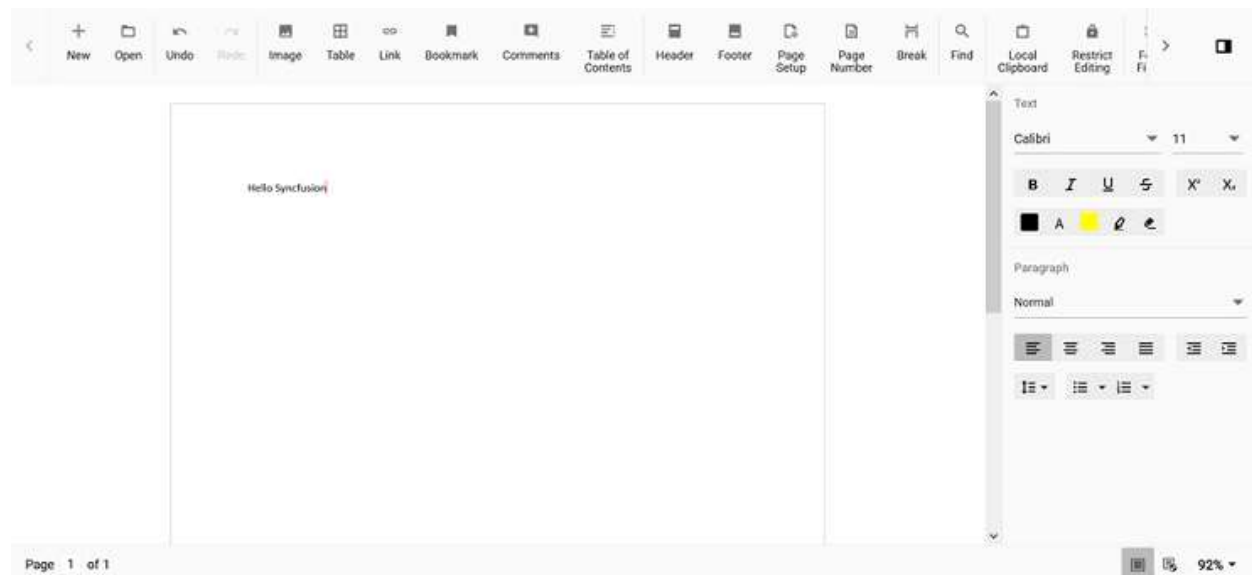
Please refer the below code snippet to change the cursor color to red.

```

.e-de-blink-cursor {
    border-left: 1px solid red!important;
}

```

Output will be like below:



### Hide tool bar and properties pane in Vue Document editor component

**Document editor container** provides the main document view area along with the built-in toolbar and properties pane.

**Document editor** provides just the main document view area. Here, the user can compose, view, and edit the Word documents. You may prefer to use this component when you want to design your own UI options for your application.

#### *Hide the properties pane*

By default, Document editor container has built-in properties pane which contains options for formatting text, table, image and header and footer. You can use [showPropertiesPane](#) API in [DocumentEditorContainer](#) to hide the properties pane.

The following example code illustrates how to hide the properties pane.

```
<template>
<div id="app">
<ejs-documenteditorcontainer
ref="container"
:serviceUrl="serviceUrl"
height="590px"
id="container"
:enableToolbar="true"
:showPropertiesPane='false'
</ejs-documenteditorcontainer>
</div>
</template>
<script>
import Vue from 'vue';
import {
DocumentEditorContainerPlugin,
DocumentEditorContainerComponent,
Toolbar,
} from '@syncfusion/ej2-vue-documenteditor';
Vue.use(DocumentEditorContainerPlugin);
export default {
data() {
return {
```

```

serviceUrl:
'https://ej2services.syncfusion.com/production/web-services/api/documenteditor/',
};
},
provide: {
//Inject require modules.
DocumentEditorContainer: [Toolbar],
}
};
</script>
`

```

Note: Positioning and customizing the properties pane in Document editor container is not possible. Instead, you can hide the exiting properties pane and create your own pane using public API's.

#### *Hide the toolbar*

You can use [enableToolbar](#) API in [DocumentEditorContainer](#) to hide the existing toolbar.

The following example code illustrates how to hide the existing toolbar.

```

`
<template>
<div id="app">
<ejs-documenteditorcontainer
ref="container"
:serviceUrl="serviceUrl"
height="590px"
id="container"
:enableToolbar="false"></ejs-documenteditorcontainer>
</div>
</template>
<script>
import Vue from 'vue';
import {
DocumentEditorContainerPlugin,
DocumentEditorContainerComponent,
Toolbar,
} from '@syncfusion/ej2-vue-documenteditor';

```

```

Vue.use(DocumentEditorContainerPlugin);
export default {
  data() {
    return {
      serviceUrl:
        'https://ej2services.syncfusion.com/production/web-services/api/documenteditor/',
    };
  },
  provide: {
    //Inject require modules.
    DocumentEditorContainer: [Toolbar],
  }
};
</script>
`

```

*See Also*

- [How to customize the toolbar](#)

Insert text or image in table programmatically in Vue Document editor component

Using Document editor API's, you can insert [text](#) or [image](#) in [table](#) programmatically based on your requirement.

Use [selection](#) API's to navigate between rows and cells.

The following example illustrates how to create 2\*2 table and then add text and image programmatically.

```

<template>
<div id="app">
<ejs-documenteditorcontainer
  ref="container"
  :serviceUrl="serviceUrl"
  height="590px"
  id="container"
  :enableToolbar="true"
  v-on:created="onCreated.bind(this)"

```

```

</ejs-documenteditorcontainer>
</div>
</template>
<script>
import Vue from 'vue';
import {
  DocumentEditorContainerPlugin,
  DocumentEditorContainerComponent,
  Toolbar,
} from '@syncfusion/ej2-vue-documenteditor';
Vue.use(DocumentEditorContainerPlugin);
export default {
  data() {
    return {
      serviceUrl:
        'https://ej2services.syncfusion.com/production/web-services/api/documenteditor/',
    };
  },
  provide: {
    //Inject require modules.
    DocumentEditorContainer: [Toolbar],
  },
  methods: {
    onCreated: function () {
      // To insert the table in cursor position
      this.$refs.container.ej2Instances.documentEditor.editor.insertTable(2, 2);
      // To insert the image at table first cell
      this.$refs.container.ej2Instances.documentEditor.editor.insertImage(
        'data:image/png;base64,iVBORw0KGgoAAAANSUhEUgAAAAUAAAACAYAAACNbybIAAAAHIEQVQI12P4
        //8/w38GIAXDIBKE0DHxgljNBAAO9TXL0Y4OHwAAAABJRU5ErkJggg=='
      );
      // To move the cursor to next cell
      this.moveCursorToNextCell();
    }
  }
};

```



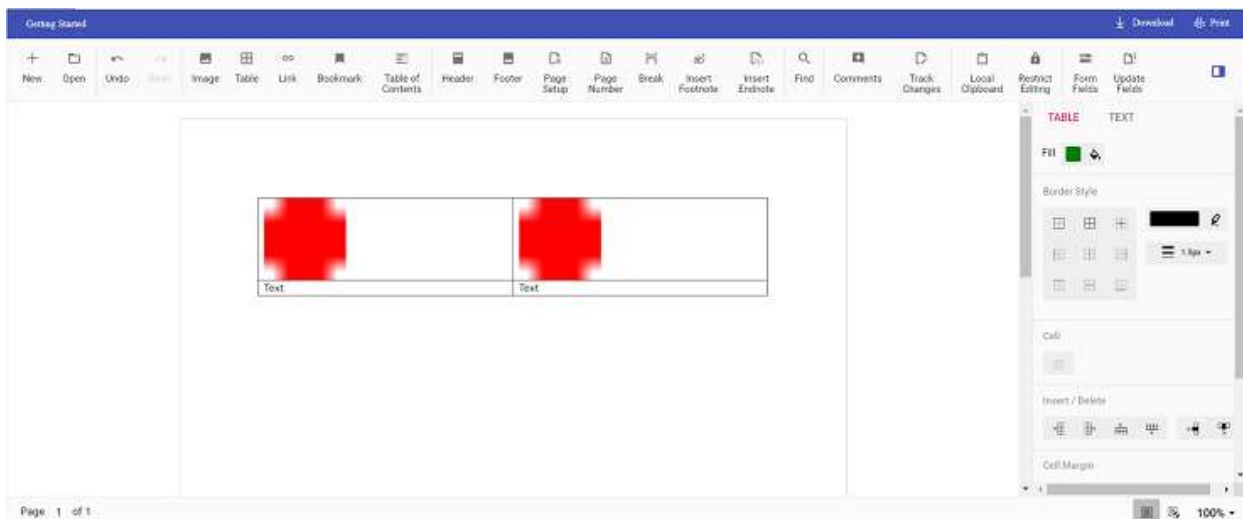
```
// To insert the image at table second cell
this.$refs.container.ej2Instances.documentEditor.editor.insertImage(
'data:image/png;base64,iVBORw0KGgoAAAANSUgAAAAUAAAAFCAYAAACNbybIAAAAHIEQVQI12P4
//8/w38GIAXDIBKE0DHxgljNBAAO9TXL0Y4OHwAAAABJRu5ErkJggg=='
);
// To move the cursor to next row
this.moveCursorToNextRow();
// To insert text in cursor position
this.$refs.container.ej2Instances.documentEditor.editor.insertText('Text');
// To move the cursor to next cell
this.moveCursorToNextCell();
// To insert text in cursor position
this.$refs.container.ej2Instances.documentEditor.editor.insertText('Text');
},
moveCursorToNextCell:function() {
// To get current selection start offset
var startOffset = this.$refs.container.ej2Instances.documentEditor.selection.startOffset;
// Increasing cell index to consider next cell
var cellIndex = parseInt(startOffset.substring(6, 7)) + 1;
// Changing start offset
startOffset =
startOffset.substring(0, 6) +
cellIndex.toString() +
startOffset.substring(7, startOffset.length);
// Navigating selection using select method
this.$refs.container.ej2Instances.documentEditor.selection.select(startOffset, startOffset);
},
moveCursorToNextRow:function() {
// To get current selection start offset
var startOffset = this.$refs.container.ej2Instances.documentEditor.selection.startOffset;
// Increasing row index to consider next row
var rowIndex = parseInt(startOffset.substring(4, 5)) + 1;
var cellIndex =
```

```

parseInt(startOffset.substring(6, 7)) != 0
? parseInt(startOffset.substring(6, 7)) - 1
: 0;
// Changing start offset
startOffset =
startOffset.substring(0, 4) +
rowIndex.toString() +
startOffset.substring(5, 6) +
cellIndex +
startOffset.substring(7, startOffset.length);
// Navigating selection using select method
this.$refs.container.ej2Instances.documentEditor.selection.select(startOffset, startOffset);
}
}
};
</script>
`

```

The output will be like below.



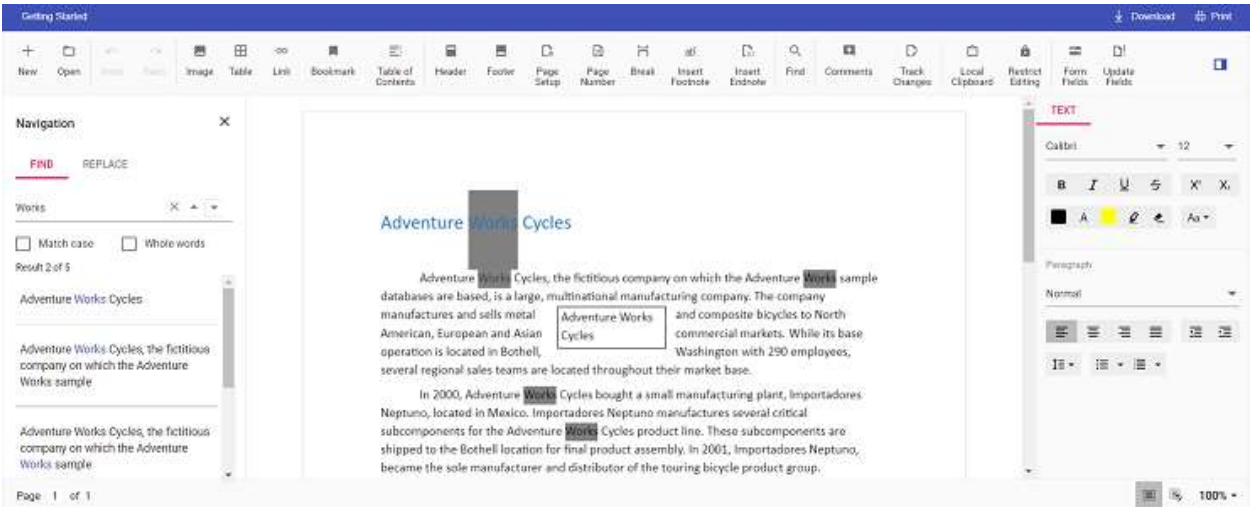
Change the default search highlight color in Vue Document editor component. Document editor provides an options to change the default search highlight color using [searchHighlightColor](#) in Document editor settings. The highlight color which is given in [documentEditorSettings](#) will be highlighted on the searched text. By default, search highlight color is yellow.

Similarly, you can use [documentEditorSettings](#) property for DocumentEditor also.

The following example code illustrates how to change the default search highlight color.

```
,  
<template>  
<div id="app">  
  <ejs-documenteditorcontainer ref='documenteditor' :serviceUrl='serviceUrl'  
    :documentEditorSettings='settings' height="590px" id='container' :enableToolbar='true'></ejs-  
    documenteditorcontainer>  
</div>  
</template>  
<script>  
import Vue from 'vue';  
  
import { DocumentEditorContainerPlugin, DocumentEditorContainerComponent, Toolbar } from  
'@syncfusion/ej2-vue-documenteditor';  
  
Vue.use(DocumentEditorContainerPlugin);  
  
export default {  
  data() {  
    return { serviceUrl: 'https://ej2services.syncfusion.com/production/web-services/api/documenteditor/',  
      // Add required color to change the default search highlight color  
      settings: { searchHighlightColor: 'Grey' } };  
  },  
  provide: {  
    //Inject require modules.  
    DocumentEditorContainer: [Toolbar]  
  }  
}  
</script>  
,
```

Output will be like below:



How to optimize the SFDT file

Starting from version v21.1.x, the SFDT file generated in Word Processor component is optimized by default to reduce the file size. All static keys are minified, and the final JSON string is compressed. This helps reduce the SFDT file size relative to a DOCX file and provides the following benefits,

- File transfer between client and server through the internet gets faster.
- The new optimized SFDT files require less storage space than the old SFDT files.

Hence, the optimized SFDT file can't be directly manipulated as JSON string.

This feature comes with a public API to switch between the old and new optimized SFDT format, allowing backward compatibility.

As a backward compatibility to create older format SFDT files, refer the following code changes,

Client/Server	Old Code	New Code from v21.1.x
Client-side		
Server-side C#	<code>WordDocument sfdtDocument = WordDocument.Load(stream, formatType);string sfdt = Newtonsoft.Json.JsonConvert.SerializeObject(sfdtDocument);</code>	<code>WordDocument sfdtDocument = WordDocument.Load(stream, formatType);sfdtDocument.OptimizeSfdt = false;string sfdt = Newtonsoft.Json.JsonConvert.SerializeObject(sfdtDocument);</code>
Server-side Java	<code>String sfdtDocument = WordProcessorHelper.load(stream, formatType);</code>	<code>String sfdtDocument = WordProcessorHelper.load(stream, formatType, false);</code>

To convert from older format SFDT from a new optimized SFDT file, refer the following code example,

Client/Server	Code example
---------------	--------------

Client-side	
Server-side C#	<pre>using(Syncfusion.DocIO.DLS.WordDocument docIODocument = WordDocument.Save(optimizedSfdt)) {sfdtDocument = WordDocument.Load(docIODocument);sfdtDocument.OptimizeSfdt = false;string oldSfdt = JsonSerializer.Serialize(sfdtDocument);}</pre>
Server-side Java	<pre>WordDocument docIODocument = WordProcessorHelper.save(optimizedSfdt);String oldSfdt = WordProcessorHelper.load(docIODocument, false);</pre>

### How to disable auto focus in Syncfusion Vue Document Editor component

Document Editor gets focused automatically when the page loads. If you want the Document editor not to be focused automatically it can be customized.

The following example illustrates to disable the auto focus in DocumentEditorContainer.

`typescript

```
<ejs-documenteditorcontainer ref="doceditcontainer" :enableAutoFocus='false' height='600px'></ejs-
documenteditorcontainer>
```

`

Note: Default value of [enableAutoFocus](#) property is `true`.

The following example illustrates to disable the auto focus in DocumentEditor.

`typescript

```
<ejs-documenteditor ref="doceditcontainer" :enableAutoFocus='false' height='600px'></ejs-
documenteditor>
```

`

Note: Default value of [enableAutoFocus](#) property is `true`.

### How to disable drag and drop in document editor in vue Document editor component

Document Editor provides support to drag and drop contents within the component and it can be customized(enable and disable) using [allowDragAndDrop](#) property in Document editor settings.

The following example illustrates to customize the drag and drop option.

`typescript

```
<template>
```

```
<ejs-documenteditorcontainer ref="doceditcontainer" :serviceUrl="hostUrl" :enableToolbar='true'
height='600px' :documentEditorSettings="settings"></ejs-documenteditorcontainer>
```

```
</template>
```

```
<script>
```

```
export default Vue.extend({
```

```
data: function() {
```

```
return {
```

```

hostUrl : 'https://services.syncfusion.com/vue/production/api/documenteditor/',
settings : { allowDragAndDrop: false };
}
}
})
</script>
`

```

Note: Default value of [allowDragAndDrop](#) property is `true`.

The following example illustrates to disable the drag and drop option in DocumentEditor.

```

`typescript
<template>
<ejs-documenteditor ref="docedit" height='600px' :documentEditorSettings="settings"></ejs-
documenteditor>
</template>
<script>
export default Vue.extend({
data: function() {
return {
hostUrl : 'https://services.syncfusion.com/vue/production/api/documenteditor/',
settings : { allowDragAndDrop: false };
}
}
})
</script>
`

```

Note: Default value of [allowDragAndDrop](#) property is `true`.

### Enable ruler

#### *How to enable ruler in Document Editor component*

Using ruler we can refer to setting specific margins, tab stops, or indentations within a document to ensure consistent formatting in Document Editor.

The following example illustrates how to enable ruler in Document Editor

### **APP.VUE**

```

<template>
  <div id="app">

```

```

    <button id='container_ruler_button' v-on:click="onClick">Show/Hide
    Ruler</button>
    <ejs-documenteditor id="container" height="370px" style="width: 100%"
    ref="container" :serviceUrl='serviceUrl' :isReadOnly='false'
    :enablePrint='true' :enableSfdtExport='true' :enableSelection='true'
    :enableContextMenu='true' :enableSearch='true' :enableOptionsPane='true'
    :enableWordExport='true' :enableTextExport='true' :enableEditor='true'
    :enableImageResizer='true' :enableEditorHistory='true'
    :enableHyperlinkDialog='true' :enableTableDialog='true'
    :enableBookmarkDialog='true' :enableTableOfContentsDialog='true'
    :enablePageSetupDialog='true' :enableStyleDialog='true'
    :enableListDialog='true' :enableParagraphDialog='true'
    :enableFontDialog='true' :enableTablePropertiesDialog='true'
    :enableBordersAndShadingDialog='true'
    :enableTableOptionsDialog='true'></ejs-documenteditor>
  </div>
</template>
<script>
  import Vue from 'vue';
  import { DocumentEditorPlugin, DocumentEditorComponent, Print,
  SfdtExport, WordExport, TextExport, Selection, Search, Editor, ImageResizer,
  EditorHistory, ContextMenu, OptionsPane, HyperlinkDialog, TableDialog,
  BookmarkDialog, TableOfContentsDialog, PageSetupDialog, StyleDialog,
  ListDialog, ParagraphDialog, BulletsAndNumberingDialog, FontDialog,
  TablePropertiesDialog, BordersAndShadingDialog, TableOptionsDialog,
  CellOptionsDialog, StylesDialog } from '@syncfusion/ej2-vue-documenteditor';
  Vue.use(DocumentEditorPlugin);
  export default {
    data() {
      return {
        serviceUrl: 'https://ej2services.syncfusion.com/production/web-
        services/api/documenteditor/',
        height: '370px',
        isReadOnly: false,
        documentEditorSettings: {showRuler: true}
      }
    },
    provide: {
      //Inject require modules.
      DocumentEditor: [Print, SfdtExport, WordExport, TextExport,
      Selection, Search, Editor, ImageResizer, EditorHistory, ContextMenu,
      OptionsPane, HyperlinkDialog, TableDialog, BookmarkDialog,
      TableOfContentsDialog, PageSetupDialog, StyleDialog, ListDialog,
      ParagraphDialog, BulletsAndNumberingDialog, FontDialog,
      TablePropertiesDialog, BordersAndShadingDialog, TableOptionsDialog,
      CellOptionsDialog, StylesDialog]
    },
    methods: {
      onClick: function() {
        this.$refs.container.ej2Instances.documentEditorSettings.showRuler =
        !this.$refs.container.ej2Instances.documentEditorSettings.showRuler
      }
    },
    mounted: function() {
      this.$refs.container.ej2Instances.documentEditorSettings.showRuler = true;
    }
  }

```

```

    }
  }
</script>
<style>
  @import '../node_modules/@syncfusion/ej2-base/styles/material.css';
  @import '../node_modules/@syncfusion/ej2-buttons/styles/material.css';
  @import '../node_modules/@syncfusion/ej2-inputs/styles/material.css';
  @import '../node_modules/@syncfusion/ej2-popups/styles/material.css';
  @import '../node_modules/@syncfusion/ej2-lists/styles/material.css';
  @import '../node_modules/@syncfusion/ej2-navigations/styles/material.css';
  @import '../node_modules/@syncfusion/ej2-splitbuttons/styles/material.css';
  @import '../node_modules/@syncfusion/ej2-dropdowns/styles/material.css';
  @import "../node_modules/@syncfusion/ej2-vue-documenteditor/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/document-editor/ruler-cs1" %}

[How to enable ruler in Document Editor Container component](#)

Using ruler we can refer to setting specific margins, tab stops, or indentations within a document to ensure consistent formatting in Document Editor Container.

The following example illustrates how to enable ruler in Document Editor Container.

#### APP.VUE

```

<template>
  <div id="app">
    <button id='container_ruler_button' v-on:click="onClick">Show/Hide
    Ruler</button>
    <ejs-documenteditorcontainer ref="container" :serviceUrl='serviceUrl'
    :height='height' :enableToolbar='true'> </ejs-documenteditorcontainer>
  </div>
</template>
<script>
  import Vue from 'vue';
  import { DocumentEditorContainerPlugin,
  DocumentEditorContainerComponent, Toolbar } from '@syncfusion/ej2-vue-
  documenteditor';
  Vue.use(DocumentEditorContainerPlugin);
  export default {
    data() {
      return {
        serviceUrl: 'https://ej2services.syncfusion.com/production/web-
        services/api/documenteditor/',
        height: '590px',
        documentEditorSettings: { showRuler: true }
      }
    },
    provide: {
      DocumentEditorContainer: [Toolbar]
    },
    methods: {
      onClick: function() {

```



```

        this.$refs.container.ej2Instances.documentEditorSettings.showRuler
    = !this.$refs.container.ej2Instances.documentEditorSettings.showRuler
    },
    },
  },
</script>
<style>
  @import '../node_modules/@syncfusion/ej2-base/styles/material.css';
  @import '../node_modules/@syncfusion/ej2-buttons/styles/material.css';
  @import '../node_modules/@syncfusion/ej2-inputs/styles/material.css';
  @import '../node_modules/@syncfusion/ej2-popups/styles/material.css';
  @import '../node_modules/@syncfusion/ej2-lists/styles/material.css';
  @import '../node_modules/@syncfusion/ej2-
navigations/styles/material.css';
  @import '../node_modules/@syncfusion/ej2-
splitbuttons/styles/material.css';
  @import '../node_modules/@syncfusion/ej2-dropdowns/styles/material.css';
  @import "../node_modules/@syncfusion/ej2-vue-
documenteditor/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/document-editor/ruler-cs2" %}

## DropDownButton

### Getting Started with the Vue Dropdown Button Component in Vue 2

This article provides a step-by-step guide for setting up a Vue 2 project using [Vue-CLI](#) and integrating the Syncfusion Vue Dropdown Button component using the [Composition API](#) / [Options API](#).

#### Prerequisites

[System requirements for Syncfusion Vue UI components](#)

#### Dependencies

The list of dependencies required to use the DropDownButton component in your application is given as follows:

```

`js
|-- @syncfusion/ej2-vue-splitbuttons
|-- @syncfusion/ej2-base
|-- @syncfusion/ej2-splitbuttons
|-- @syncfusion/ej2-vue-base
|-- @syncfusion/ej2-popups
|-- @syncfusion/ej2-buttons
`

```

#### Setting up the Vue 2 project

To generate a Vue 2 project using Vue-CLI, use the [vue create](#) command. Follow these steps to install Vue CLI and create a new project:

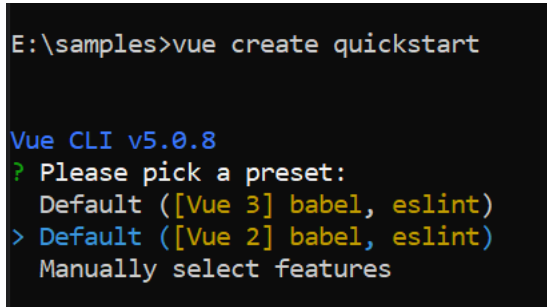
```

`bash

```

```
npm install -g @vue/cli
vue create quickstart
cd quickstart
npm run serve
`
or
`bash
yarn global add @vue/cli
vue create quickstart
cd quickstart
yarn run serve
`
```

When creating a new project, choose the option `Default ([Vue 2] babel, eslint)` from the menu.



```
E:\samples>vue create quickstart

Vue CLI v5.0.8
? Please pick a preset:
  Default ([Vue 3] babel, eslint)
> Default ([Vue 2] babel, eslint)
  Manually select features
```

Once the `quickstart` project is set up with default settings, proceed to add Syncfusion components to the project.

#### Add Syncfusion Vue packages

Syncfusion packages are available at [npmjs.com](https://www.npmjs.com). To use Vue components, install the required npm package.

This article uses the [Vue Dropdown Button component](#) as an example. Install the `@syncfusion/ej2-vue-splitbuttons` package by running the following command:

```
`bash
npm install @syncfusion/ej2-vue-splitbuttons --save
`
or
`bash
yarn add @syncfusion/ej2-vue-splitbuttons
`
```

### Import Syncfusion CSS styles

You can import themes for the Syncfusion Vue component in various ways, such as using CSS or SASS styles from npm packages, CDN, [CRG](#) and [Theme Studio](#). Refer to [themes topic](#) to know more about built-in themes and different ways to refer to themes in a Vue project.

In this article, the **Material** theme is applied using CSS styles, which are available in installed packages. The necessary **Material** CSS styles for the Dropdown Button component and its dependents were imported into the `<style>` section of **src/App.vue** file.

#### ~/SRC/APP.VUE

```
<style>
@import '../node_modules/@syncfusion/ej2-base/styles/material.css';
@import '../node_modules/@syncfusion/ej2-buttons/styles/material.css';
@import '../node_modules/@syncfusion/ej2-popups/styles/material.css';
@import '../node_modules/@syncfusion/ej2-splitbuttons/styles/material.css';
</style>
```

### Add Syncfusion Vue component

Follow the below steps to add the Vue Dropdown Button component using **Composition API** or **Options API**:

1\ First, import and register the Dropdown Button component in the **script** section of the **src/App.vue** file. If you are using the **Composition API**, you should add the **setup** attribute to the **script** tag to indicate that Vue will be using the **Composition API**.

#### COMPOSITION API (~/SRC/APP.VUE)

```
<script setup>
import { DropDownButtonComponent as EjsDropdownbutton } from
"@syncfusion/ej2-vue-splitbuttons";
</script>
```

#### OPTIONS API (~/SRC/APP.VUE)

```
<script>
import { DropDownButtonComponent } from "@syncfusion/ej2-vue-splitbuttons";
export default {
  components: {
    'ejs-dropdownbutton': DropDownButtonComponent
  }
}
</script>
```

2\ In the **template** section, define the Dropdown Button component with the [content](#) property.

#### ~/SRC/APP.VUE

```
<template>
<ejs-dropdownbutton :items='items'>Clipboard</ejs-dropdownbutton>
</template>
```

3\ Declare the value for the **items** property in the **script** section.

**COMPOSITION API (~SRC/APP.VUE)**

```
<script setup>
const items = [
  {
    text: 'Cut'
  },
  {
    text: 'Copy'
  },
  {
    text: 'Paste'
  }
];
</script>
```

**OPTIONS API (~SRC/APP.VUE)**

```
<script>
data () {
  return {
    items:[
      {
        text: 'Cut'
      },
      {
        text: 'Copy'
      },
      {
        text: 'Paste'
      }
    ]
  };
}
</script>
```

Here is the summarized code for the above steps in the **src/App.vue** file:

**COMPOSITION API (~SRC/APP.VUE)**

```
<template>
<ejs-dropdownbutton :items='items'>Clipboard</ejs-dropdownbutton>
</template>
<script setup>
import { DropDownButtonComponent as EjsDropDownbutton } from
"@syncfusion/ej2-vue-splitbuttons";
import { enableRipple } from '@syncfusion/ej2-base';
enableRipple(true);
const items = [
  {
    text: 'Cut'
  },
  {
    text: 'Copy'
  },
  {
    text: 'Paste'
  }
];
```

```
</script>
<style>
@import '../node_modules/@syncfusion/ej2-base/styles/material.css';
@import '../node_modules/@syncfusion/ej2-buttons/styles/material.css';
@import '../node_modules/@syncfusion/ej2-popups/styles/material.css';
@import '../node_modules/@syncfusion/ej2-splitbuttons/styles/material.css';
</style>
```

### OPTIONS API (~SRC/APP.VUE)

```
<template>
<ejs-dropdownbutton :items='items'>Clipboard</ejs-dropdownbutton>
</template>
<script>
import { DropDownButtonComponent } from "@syncfusion/ej2-vue-splitbuttons";
import { enableRipple } from '@syncfusion/ej2-base';
enableRipple(true);
export default {
  components: {
    'ejs-dropdownbutton': DropDownButtonComponent
  },
  data () {
    return {
      items:[
        {
          text: 'Cut'
        },
        {
          text: 'Copy'
        },
        {
          text: 'Paste'
        }
      ]
    };
  }
}
</script>
<style>
@import '../node_modules/@syncfusion/ej2-base/styles/material.css';
@import '../node_modules/@syncfusion/ej2-buttons/styles/material.css';
@import '../node_modules/@syncfusion/ej2-popups/styles/material.css';
@import '../node_modules/@syncfusion/ej2-splitbuttons/styles/material.css';
</style>
```

### Run the project

To run the project, use the following command:

```
`bash
```

```
npm run serve
```

```
`
```

or

```
`bash
```

```
yarn run serve
```

```
,
```

```
{% previewsample "page.domainurl/code-snippet/drop-down-button/default-cs2" %}
```

See Also

- [DropDownButton with icons](#)
- [How to hide dropdown arrow](#)
- [Dropdown popup with separator](#)

## Getting Started with the Vue DropDownButton Component in Vue 3

This article provides a step-by-step guide for setting up a [Vite](#) project with a JavaScript environment and integrating the Syncfusion Vue DropDownButton component using the [Composition API](#) / [Options API](#).

The **Composition API** is a new feature introduced in Vue.js 3 that provides an alternative way to organize and reuse component logic. It allows developers to write components as functions that use smaller, reusable functions called composition functions to manage their properties and behavior.

The **Options API** is the traditional way of writing Vue.js components, where the component logic is organized into a series of options that define the component's properties and behavior. These options include data, methods, computed properties, watchers, lifecycle hooks, and more.

### Prerequisites

#### [System requirements for Syncfusion Vue UI components](#)

#### Set up the Vite project

A recommended approach for beginning with Vue is to scaffold a project using [Vite](#). To create a new Vite project, use one of the commands that are specific to either NPM or Yarn.

```
`bash
```

```
npm create vite@latest
```

```
,
```

or

```
`bash
```

```
yarn create vite
```

```
,
```

Using one of the above commands will lead you to set up additional configurations for the project as below:

1. Define the project name: We can specify the name of the project directly. Let's specify the name of the project as **my-project** for this article.

```
`bash
```

```
? Project name: » my-project
```

```
,
```

2. Select **Vue** as the framework. It will create a Vue 3 project.

```
`bash
```

? Select a framework: » - Use arrow-keys. Return to submit.

Vanilla

Vue

React

Preact

Lit

Svelte

Others

,

3. Choose **JavaScript** as the framework variant to build this Vite project using JavaScript and Vue.

```
`bash
```

? Select a variant: » - Use arrow-keys. Return to submit.

JavaScript

TypeScript

Customize with create-vue ↗

Nuxt ↗

,

4. Upon completing the aforementioned steps to create the **my-project**, run the following command to install its dependencies:

```
`bash
```

```
cd my-project
```

```
npm install
```

,

or

```
`bash
```

```
cd my-project
```

```
yarn install
```

,

Now that **my-project** is ready to run with default settings, let's add Syncfusion components to the project.

### Add Syncfusion Vue packages

Syncfusion Vue component packages are available at [npmjs.com](https://www.npmjs.com). To use Syncfusion Vue components in the project, install the corresponding npm package.

This article uses the [Vue DropDownButton component](#) as an example. To use the Vue DropDownButton component in the project, the `@syncfusion/ej2-vue-splitbuttons` package needs to be installed using the following command:

```
`bash
```

```
npm install @syncfusion/ej2-vue-splitbuttons --save
```

```
,
```

or

```
`bash
```

```
yarn add @syncfusion/ej2-vue-splitbuttons
```

```
,
```

### Import Syncfusion CSS styles

You can import themes for the Syncfusion Vue component in various ways, such as using CSS or SASS styles from npm packages, CDN, [CRG](#) and [Theme Studio](#). Refer to [themes topic](#) to know more about built-in themes and different ways to refer to themes in a Vue project.

In this article, `Material` theme is applied using CSS styles, which are available in installed packages. The necessary `Material` CSS styles for the DropDownButton component and its dependents were imported into the `<style>` section of `src/App.vue` file.

#### ~/SRC/APP.VUE

```
<style>
@import '../node_modules/@syncfusion/ej2-base/styles/material.css';
@import '../node_modules/@syncfusion/ej2-buttons/styles/material.css';
@import '../node_modules/@syncfusion/ej2-popups/styles/material.css';
@import '../node_modules/@syncfusion/ej2-splitbuttons/styles/material.css';
</style>
```

The order of importing CSS styles should be in line with its dependency graph.

### Add Syncfusion Vue component

Follow the below steps to add the Vue DropDownButton component using `Composition API` or `Options API`:

1.First, import and register the DropDownButton component and its child directives in the `script` section of the `src/App.vue` file. If you are using the `Composition API`, you should add the `setup` attribute to the `script` tag to indicate that Vue will be using the `Composition API`.

#### COMPOSITION API (~/SRC/APP.VUE)

```
<script setup>
import { DropDownButtonComponent as EjsDropDownButton } from
"@syncfusion/ej2-vue-splitbuttons";
</script>
```



**OPTIONS API (~SRC/APP.VUE)**

```
<script>
import { DropDownButtonComponent } from "@syncfusion/ej2-vue-splitbuttons";
//Component registration
export default {
  name: "App",
  components: {
    "ejs-dropdownbutton": DropDownButtonComponent
  }
}
</script>
```

2. In the **template** section, define the DropDownButton component with the [items](#) property and content property.

**~/SRC/APP.VUE**

```
<template>
<ejs-dropdownbutton :items='items' content="Paste"></ejs-dropdownbutton>
</template>
```

3. Declare the values for the **items** property in the **script** section.

**COMPOSITION API (~SRC/APP.VUE)**

```
<script setup>
const items=[
{
  text: 'Cut'
},
{
  text: 'Copy'
},
{
  text: 'Paste'
}]
</script>
```

**OPTIONS API (~SRC/APP.VUE)**

```
<script>
data() {
  return {
    items:[
      {
        text: 'Cut'
      },
      {
        text: 'Copy'
      },
      {
        text: 'Paste'
      }
    ]
  }
}
```

```
};  
}  
</script>
```

3. Here is the summarized code for the above steps in the **src/App.vue** file:

#### **COMPOSITION API (~SRC/APP.VUE)**

```
<template>  
<ejs-dropdownbutton :items='items' content="Paste"></ejs-dropdownbutton>  
</template>  
<script setup>  
import { DropDownButtonComponent as EjsDropDownButton } from  
"@syncfusion/ej2-vue-splitbuttons";  
const items=[  
  {  
    text: 'Cut'  
  },  
  {  
    text: 'Copy'  
  },  
  {  
    text: 'Paste'  
  }  
]  
</script>  
<style>  
@import '../node_modules/@syncfusion/ej2-base/styles/material.css';  
@import '../node_modules/@syncfusion/ej2-buttons/styles/material.css';  
@import '../node_modules/@syncfusion/ej2-popups/styles/material.css';  
@import '../node_modules/@syncfusion/ej2-splitbuttons/styles/material.css';  
</style>
```

#### **OPTIONS API (~SRC/APP.VUE)**

```
<template>  
<ejs-dropdownbutton :items='items'>Clipboard</ejs-dropdownbutton>  
</template>  
<script>  
import { DropDownButtonComponent } from "@syncfusion/ej2-vue-splitbuttons";  
export default {  
  name: "App",  
  components: {  
    "ejs-dropdownbutton": DropDownButtonComponent  
  },  
  data () {  
    return {  
      items:[  
        {  
          text: 'Cut'  
        },  
        {  
          text: 'Copy'  
        },  
        {  
          text: 'Paste'  
        }  
      ]  
    }  
  }  
}
```

```
};  
}  
}  
</script>  
<style>  
@import '../node_modules/@syncfusion/ej2-base/styles/material.css';  
@import '../node_modules/@syncfusion/ej2-buttons/styles/material.css';  
@import '../node_modules/@syncfusion/ej2-popups/styles/material.css';  
@import '../node_modules/@syncfusion/ej2-splitbuttons/styles/material.css';  
</style>
```

### Run the project

To run the project, use the following command:

```
`bash
```

```
npm run dev
```

```
,
```

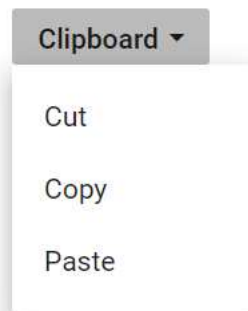
or

```
`bash
```

```
yarn run dev
```

```
,
```

The output will appear as follows:



**Sample:** [vue-3-drop down button-getting-started](#).

For migrating from Vue 2 to Vue 3, refer to the [migration](#) documentation.

See also

- [Getting Started with Vue UI Components using Composition API and TypeScript](#)
- [Getting Started with Vue UI Components using Options API and TypeScript](#)

## Icons in Vue Drop down button component

### DropDownButton icons

DropDownButton can have an icon to provide the visual representation of the action. To place the icon on a DropdownButton, set the [iconCss](#) property to `e-icons` with the required icon CSS. By default, the

icon is positioned to the left side of the DropdownButton. You can customize the icon's position using the [iconPosition](#) property.

In the following example, the DropdownButton with default iconPosition and iconPosition as **Top** is showcased:

#### APP.VUE

```
<template>
<div>
<ejs-dropdownbutton :items='items' iconCss= 'ddb-icons e-
message'>Message</ejs-dropdownbutton>
<ejs-dropdownbutton :items='items' iconCss= 'ddb-icons e-message'
iconPosition= 'Top'>Message</ejs-dropdownbutton>
</div>
</template>
<script>
import Vue from 'vue';
import { DropDownButtonPlugin } from "@syncfusion/ej2-vue-splitbuttons";
import { enableRipple } from '@syncfusion/ej2-base';
enableRipple(true);
Vue.use(DropDownButtonPlugin);
export default {
  data () {
    return {
      items:[
        {
          text: 'Edit'
        },
        {
          text: 'Delete'
        },
        {
          text: 'Mark as Read'
        },
        {
          text: 'Like Message'
        }
      ]
    };
  }
}
</script>
<style>
@import '../node_modules/@syncfusion/ej2-base/styles/material.css';
@import '../node_modules/@syncfusion/ej2-buttons/styles/material.css';
@import '../node_modules/@syncfusion/ej2-popups/styles/material.css';
@import '../node_modules/@syncfusion/ej2-
splitbuttons/styles/material.css';
@font-face {
  font-family: 'e-db-icons';
  src:
    url(data:application/x-font-ttf;charset=utf-
8;base64,AAEAAAAKAIAAAwAgTlMvMj0jSROAAAEoAAAAVmNtYXNfudgAAABkAAAAADpnbHlmSrK
TCAAAAdgAAAC4aGVhZBktK8cAAADQAAAAANmhoZWEHmQNtAAAArAAAACRobXR4D7gAAAAAYAAAAA
QbG9jYQB4ADoAAAHMAAAACm1heHABEAAAYAAABCAAAACBuYW1lH00mDAAAAPAAAAJJcG9zdIwksr0
AAATcAAAATQABAAADUv9qAFoEAAAA//4D6gABAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAQAAGc/PS18
PPPUACwPoAAAAANfSc3wAAAAA19JzfAAAAAAD6gPqAAACAAACAAAAAAAAAAAAAAAAEAAAwAAgAAAAA
```

```

AAgAAAAoACgAAAP8AAAAAAAAAAQPUAZAABQAAAAnoCvAAAAIwCegK8AAAB4AAxAQIAAAIABQMAAAA
AA
D6AAAA+gAAAAAAAAIAAADAAAAFAADAAEAAAAUAAQAjgAAAAQABAABADnBf//AADnA///AAAAAQAE
AAAAAQACAAMAAAAAAAAAAHAA6AFwAAAACAAAAAPqA2UABgAKAAA3IREjCQEjBRcBIQID6AL+Dv4
NAQEY3QG4/I+IAsL+GAHonroBcwAAAAIAAAAAA8YD6gAFAAoAADchESMJASUHCQImA6AD/jL+MQE
EywGWAZb+agICX/4+AcLXsv6cAWQBZAAAAAEAAAAA+oD6gALAAATCQEXCQE3CQEnCQECATP+zcI
BMgEzwf7OATLB/s3+zgMp/s3+zsIBM/7NwgEyATPB/s4BMgAAAAASAN4AAQAAAAAAAAAAAAAAAAQA
AAAAAQAKAAEAQAAAAAAAgAHAAsAAQAAAAAAAwAKABIAAQAAAAAABAAKABWAAQAAAAAABQALACY
AAQAAAAAABgAKADEAAQAAAAAACgAsADsAAQAAAAAACWASAGcAAwABBakAAAACAHkAAwABBakAAQA
UAHsAAwABBakAAgAOAI8AAwABBakAAwAUAJ0AAwABBakABAAUALEAAwABBakABQAWAMUAwABBak
ABgAUANsAAwABBakACgBYAO8AAwABBakACwAkAUcgZS1kYi1pY29uc1JlZ3VsYXJlLWwRiLWljb25
zZS1kYi1pY29uc1JlcnNpb24gMS4wZS1kYi1pY29uc0ZvbnQgZ2VuZXJhdGVkIHVzaW5nIFN5bmN
mdXNpb24gTWV0cm8gU3R1ZGlvd3d3LnN5bmNmdXNpb24uY29tACAAZQAtAGQAYgAtAGkAYwBvAG4
AcwBSAGUAZwB1AGwAYQByAGUALQBkAGIALQBpAGMAbwBuAHMAZQAtAGQAYgAtAGkAYwBvAG4AcwB
WAGUAcgBzAGkAbwBuACAAMQAuADAAZQAtAGQAYgAtAGkAYwBvAG4AcwBGAG8AbgB0ACAAZwB1AG4
AZQByAGEAdABLAGQAIAB1AHMAaQBuaGcAIABTAHkAbgBjAGYAdQBzAGkAbwBuACAATQB1AHQAcgB
vACAAUwB0AHUAZABpAG8AdwB3AHcALgBzAHkAbgBjAGYAdQBzAGkAbwBuAC4AYwBvAG0AAAAAAGa
AAEAQIBAwEEAQUADG1lc3NhZ2UtZWVpYyZWF
kLXVucmVhZAZkZWxldGUAAAAAAAAA==) format('trueType');
    font-weight: normal;
    font-style: normal;
  }
  .ddb-icons {
    font-family: 'e-db-icons' !important;
    speak: none;
    font-size: 55px;
    font-style: normal;
    font-weight: normal;
    font-variant: normal;
    text-transform: none;
    line-height: 1;
    -webkit-font-smoothing: antialiased;
    -moz-osx-font-smoothing: grayscale;
  }
  .e-message::before {
    content: '\e703';
  }
}
</style>

```

{% previewsample "page.domainurl/code-snippet/drop-down-button/default-cs13" %}

#### Icon only DropDownButton

Icon only DropDownButton can be achieved by using [iconCss](#) property and to hide drop down arrow [e-caret-hide](#) class is added using [cssClass](#) property.

#### APP.VUE

```

<template>
  <ejs-dropdownbutton :items='items' iconCss= 'e-icons e-menu' cssClass= 'e-
  caret-hide'></ejs-dropdownbutton>
</template>
<script>
import Vue from 'vue';
import { DropDownButtonPlugin } from "@syncfusion/ej2-vue-splitbuttons";
import { enableRipple } from '@syncfusion/ej2-base';
enableRipple(true);

```

```

Vue.use(DropDownButtonPlugin);
export default {
  data () {
    return {
      items:[
        {
          text: 'New tab'
        },
        {
          text: 'New window'
        },
        {
          text: 'New incognito window'
        },
        {
          separator: true
        },
        {
          text: 'Print'
        },
        {
          text: 'Cast'
        },
        {
          text: 'Find'
        }
      ]
    };
  }
}
</script>
<style>
  @import
'https://ej2.syncfusion.com/vue/documentation/node_modules/@syncfusion/ej2-
base/styles/material.css';
  @import '../node_modules/@syncfusion/ej2-buttons/styles/material.css';
  @import '../node_modules/@syncfusion/ej2-popups/styles/material.css';
  @import '../node_modules/@syncfusion/ej2-
splitbuttons/styles/material.css';
  .e-menu::before {
    content: '\e984';
  }
</style>

```

{% previewsample "page.domainurl/code-snippet/drop-down-button/default-cs14" %}

The Essential JS 2 provides a set of icons that can be loaded by applying **e-icons** class name to the element. You can also use third party icons on the DropDownButton using the [iconCss](#) property.

#### *DropDownButton with sprite image*

Sprite images can be loaded in DropDownButton instead of font icons using **[iconCss]**(  
<https://ej2.syncfusion.com/vue/documentation/api/drop-down-button/#iconcss>) property.

In this following example, **e-image** class is added with background url of the sprite image along with X and Y positions. The **width** and **height** of the element set as **32px**.

#### **APP.VUE**

```

<template>
<ejs-dropdownbutton :items='items' cssClass= 'e-caret-hide' iconCss= 'e-
image'></ejs-dropdownbutton>
</template>
<script>
import Vue from 'vue';
import { DropDownButtonPlugin } from "@syncfusion/ej2-vue-splitbuttons";
import { enableRipple } from '@syncfusion/ej2-base';
enableRipple(true);
Vue.use(DropDownButtonPlugin);
export default {
  data () {
    return {
      items:[
        {
          text: 'Display Settings'
        },
        {
          text: 'System Settings'
        },
        {
          text: 'Additional Settings'
        }
      ]
    };
  }
}
</script>
<style>
@import '../node_modules/@syncfusion/ej2-base/styles/material.css';
@import '../node_modules/@syncfusion/ej2-buttons/styles/material.css';
@import '../node_modules/@syncfusion/ej2-popups/styles/material.css';
@import '../node_modules/@syncfusion/ej2-
splitbuttons/styles/material.css';
.e-btn-icon.e-image {
  background: url(../spritesheet.png) -384px -48px;
  width: 32px;
  height: 32px;
}
</style>

```

{% previewsample "page.domainurl/code-snippet/drop-down-button/default-cs15" %}

### Vertical button

Vertical button in DropDownButton can be achieved by adding `e-vertical` class using [cssClass](#) property.

The following example illustrates how to provide vertical support in DropDownButton component.

### APP.VUE

```

<template>
<ejs-dropdownbutton :items='items' iconCss= 'ddb-icons e-message' cssClass=
'e-vertical' iconPosition= 'Top'>Message</ejs-dropdownbutton>
</template>
<script>
import Vue from 'vue';
import { DropDownButtonPlugin } from "@syncfusion/ej2-vue-splitbuttons";
import { enableRipple } from '@syncfusion/ej2-base';

```

```

enableRipple(true);
Vue.use(DropDownButtonPlugin);
export default {
  data () {
    return {
      items:[
        {
          text: 'Edit'
        },
        {
          text: 'Delete'
        },
        {
          text: 'Mark as Read'
        },
        {
          text: 'Like Message'
        }
      ]
    };
  }
}
</script>
<style>
  @import '../node_modules/@syncfusion/ej2-base/styles/material.css';
  @import '../node_modules/@syncfusion/ej2-buttons/styles/material.css';
  @import '../node_modules/@syncfusion/ej2-popups/styles/material.css';
  @import '../node_modules/@syncfusion/ej2-
splitbuttons/styles/material.css';
  @font-face {
    font-family: 'e-db-icons';
    src:
      url(data:application/x-font-ttf;charset=utf-
8;base64,AAEAAAAKAIAAAwAgTlMvMj0jSRoAAAEoAAAAVmNtYXdnFudgAAABKAAAAADpnbHlmSrK
TCAAAAdgAAAC4aGVhZBKtK8cAAADQAAAAANmhoZWEHmQNtAAAArAAAACRobXR4D7gAAAAAYAAAAA
QbG9jYQB4ADoAAAHMAAAACm1heHABEAAYAAABCAAAACBuYW1lH00mDAAAAPAAAAJJcG9zdIwKsr0
AAATcAAAATQABAAADUv9qAFoEAAAA//4D6gABAAAAAAAAAAAAAAAAABABAAAAQAAGc/PS18
PPPUACwPoAAAAANfSc3wAAAAA19JzfAAAAAAD6gPqAAAACAACAAAAAAAAAAAAEAAAAEAAwAAgAAAA
AAgAAAAoACgAAAP8AAAAAAAAAAQPUAZAABQAAAAnoCvAAAAIwCegK8AAAB4AAxAQIAAAIABQMAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAUGZFZABA5wPnBQNS/2oAWgPqAJYAAAABAAAAAAAAABAAAAAPoAAA
D6AAAA+gAAAAAAAAIAAADAAAAFAADAAEAAAAUAAQAjgAAAAQABAABADnBf//AADnA///AAAAQA
EAAAAQACAAMAAAAAAAAAHAA6AFwAAAACAAAAAAPqA2UABgAKAAA3IREjCQEjBRcBIQID6AL+Dv4
NAQEY3QG4/I+IAsL+GAHonroBcwAAAAIAAAAAA8YD6gAFAAoAADchESMJASUHCQImA6AD/jL+MQE
EywGWAZb+agICX/4+AcLXsv6cAWQBZAAAAEAAAAA+oD6gALAAATCQEXCQE3CQEnCQECATP+zcI
BMgEzwf7OATLB/s3+zgMp/s3+zsIBM/7NwgEyATPB/s4BMgAAAAASAN4AAQAAAAAAAAABAAAAQA
AAAAAQAKAAEAAQAAAAAAAAgAHAAsAAQAAAAAAAAAwAKABIAAQAAAAAAAABAABAAQAAAAAABQALACY
AAQAAAAAABgAKADEAAQAAAAAACgAsADsAAQAAAAAACwASAGcAAwABBAKAAAACAHkAAwABBAKAAQA
UAHsAAwABBAKAAgAOAI8AAwABBAKAAwAUAJ0AAwABBAKABAAUALEAAwABBAKABQAWAMUAwABBAK
ABgAUANsAAwABBAKACgBYAO8AAwABBAKACwAKAUcgZS1kYi1pY29uc0Zvbnp24uY29tACAAZQAtAGQAYgAtAGkAYwBvAG4
AcwBSAGUAZwB1AGwAYQByAGUALQBkAGIALQBpAGMABwBuAHMAZQAtAGQAYgAtAGkAYwBvAG4AcwB
WAGUAcgBzAGkAbwBuACAAMQAuADAAZQAtAGQAYgAtAGkAYwBvAG4AcwBGAG8AbgB0ACAAZwB1AG4
AZQByAGEAdABLAGQAIAB1AHMAaQBuAGcAIABTAHkAbgBjAGYAdQBzAGkAbwBuACAATQBlAHQAcgB
vACAAUwB0AHUAZABpAG8AdwB3AHcALgBzAHkAbgBjAGYAdQBzAGkAbwBuAC4AYwBvAG0AAAAAAGA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAEAQIBAwEEAQUADG1lc3NhZ2UtZWpibAtyZWf
kLXVucmVhZAZkZWxlZGUAAAAAAAAA==) format('trueType');
    font-weight: normal;
  }

```



```

        font-style: normal;
    }
    .ddb-icons {
        font-family: 'e-db-icons' !important;
        speak: none;
        font-size: 55px;
        font-style: normal;
        font-weight: normal;
        font-variant: normal;
        text-transform: none;
        line-height: 1;
        -webkit-font-smoothing: antialiased;
        -moz-osx-font-smoothing: grayscale;
    }
    .e-message::before {
        content: '\e703';
    }
}
</style>

```

{% previewsample "page.domainurl/code-snippet/drop-down-button/default-cs16" %}

See Also

- [Dropdown popup with icons](#)
- [Customized icon size](#)

## Popup items in Vue Drop down button component

### Icons

The popup action item have an icon or image to provide visual representation of the action. To place the icon on a popup item, set the [iconCss](#) property to `e-icons` with the required icon CSS. By default, the icon is positioned to the left side of the popup action item.

In the following sample, the icons for edit, delete, mark as read and like message menu items are added using the `iconCss` property.

### APP.VUE

```

<template>
<ejs-dropdownbutton :items='items' iconCss= 'ddb-icons e-
message'>Message</ejs-dropdownbutton>
</template>
<script>
import Vue from 'vue';
import { DropDownButtonPlugin } from "@syncfusion/ej2-vue-splitbuttons";
import { enableRipple } from '@syncfusion/ej2-base';
enableRipple(true);
Vue.use(DropDownButtonPlugin);
export default {
  data () {
    return {
      items:[
        {
          text: 'Edit',
          iconCss: 'ddb-icons e-edit'

```

```

    },
    {
        text: 'Delete',
        iconCss: 'ddb-icons e-delete'
    },
    {
        text: 'Mark As Read',
        iconCss: 'ddb-icons e-read'
    },
    {
        text: 'Like Message',
        iconCss: 'ddb-icons e-like'
    }
];
}
</script>
<style>
@import '../node_modules/@syncfusion/ej2-base/styles/material.css';
@import '../node_modules/@syncfusion/ej2-buttons/styles/material.css';
@import '../node_modules/@syncfusion/ej2-popups/styles/material.css';
@import '../node_modules/@syncfusion/ej2-splitbuttons/styles/material.css';
@font-face {
    font-family: 'ddb-icons';
    src:
        url(data:application/x-font-ttf;charset=utf-8;base64,AAEAAAAKAIAAAwAgTlMvMjltSfYAAAEoAAAAMVnTYXDNGodnAAABmAAAAD5nbHlm/RE9ZwAAAegAAAJ8aGVhZBOFuxsAADQAAAAANmhoZWElUQQHAAAArAAAACRobXR4GAAAAAAAAAYAAAAAYbG9jYQHiAO4AAAHYAAAAADmlheHABFACZAABCAAAACBuYW11LBM9QAABGQAAAI9cG9zdOdmKCAAAAakAAAAZgABAAAEAAAAAFwEAAAAAAD9AABAAAAAAAAAAAAAAAAAAAAABgABAAAAAQAAfi9ISf8PPPUACwQAAAAANG+uxUAAAAA2D67FQAAAAAD9AP0AAAACAACAAAAAAAAAAAAEAAAAGAI0ABAAAAAAAgAAAAAACgAAAP8AAAAAAAAAAQAAZAABQAAAokCzAAAAI8CiQLMAAAB6wAyAQgAAAIABQMAAAAA
            AAAAAAAAAAAAAAAAAAAAAAUGZFZABA5wDnBAQAAAAAXAQAAAAAAAAABAAAAAAAAABAAAAAQAAAAEAAAABAAAAAQAAAAEAAAAAAAAAAgAAAAAMAAAAUAAMAAQAAABQABAAqAAAAABAAEEAAEOce//8AAOcA//8AAAABAAQAAAAABAAIAAwEAEEAUAAAAAAAAALgBaAHYAIAE+AAAAWAAAAAD9AP0AAIABgAZAAAZJSc3FWEnNwcXPwM1LwcPAGwBJOO76QHT6qlu6XIIFBAICBAWmCakJCgkJCQw66jrpAdLpqW7pcggJCgkKCQimBwQDAQEDBAAAAAAAAEAAAAANNA/QAAWAHABAAGAAAAAREjESMRIxEnETMVITUzeSE3IXuhNSMIQLIH4SFhUICGED9ZoWFAPqF/nAcP/3qAhb96gIWQv1mQ0MC3YVCQkMAAAAAAgAAAAAD8wNuAAYACgAANYeRIwkBIwUXASEMA+gC/g3+DgEBGNwBufyOkGL/hCB6Z+5AXIAAAACAAAAAAPQA/QABQAKAAA3IREjCQE1BwkCMAOGA/4x/jIBA8sBlgGX/moMAL7+PgHC2LL+nAfKAWQAAAAACAAAAAAP0A8UAAwCMAAA3MxEjJAQ8DFRcPDBeZNx8ENxc/Cj0BLWU/Cy8INzU/CDUVBTU/DtUVcQclPwQ1LwsjdWEMraOB+QIKBAEBAQEYIREREHMiCQkoEAYhbZUHHjmT2w4FCAsNCwkFAwQCAGQJBgIBAQEDDgQJCAYHAwMBAQEBAwMDCQIBAQMWcWUEBAMDagICBAQKAQEBAaOHbWyFBQDDAwEBAQEEBQcJBQUFBhh+rQ8JBAMCAQEDAwoMFQMHBgwLDQcHWgGHAd4BBQMDdh8KBCw6HRscGi8JCBsm/ooBAR8DAQEBAGeBAwYKCGwGCAGIBQgJCASFBAQEBOGMGAwCICAwiBwgHBgYGBQUJBAIGAGQMCQYFYBgCJCQoJCAgHCwQCBQMCBAQEBOUGBwcIBwYGBgYKCGGAGIBAQEBRJezGhsNDQwNcyIemMQQEAQBQIAAAASAN4AAQAAAAAAAAABAAAAAQAAAAAAQAJAAEAQAAAAAAAgAHAaoAAQAAAAAAAwAJABEAAQAAAAAABAAJABoAAQAAAAAABQALACMAAQAAAAAABgAJAC4AAQAAAAAACgAsAdCAAQAAAAAACwASAGMAAwABBakAAAACAHUAAwABBakAAQASAHcAAwABBakAAgAOAiKAAwABBakAAwASAJcAAwABBakABAASAKkAAwABBakABQAWAlSAwABBakABgASANEAAwABBakACgBYAOMAawABBakACwAkATsgZGRiLWLjb25zUmVndWxhcmlRkyIlpy29uc2RkyIlpy29uc1ZlcnpBNpb24gMS4wZGRiLWLjb25zRm9udCBnZW51cmF0ZWQgdXNpbmcgU3luY2Zlc2lvbiBNZXRYbyBTdHVkaW93d3cuc3luY2Zlc2lvbi5jb20AIAABKAGQAYgAtAGkAYwBvAG4AcwBSAGUAZWBlAGwAYQByAGQAZABiAC0AaQBJAG8AbgBzAFYAZQByAHMAaQBvAG4AIAAxAC4AMABKAGQAYgAtAGkAYwBvAG4AcwBGAG8AbgB0ACAAZwBlAG4AZQByAGEAdABLAGQAIBLAHMAaQBuaGCAIABTAAhkAbgBjAGYAdQBzAGkAbwBuACAAATQBLAHQACgbvACAAUwB0AHUAZABpAG8AdwB3AHcALgBzAHkAbgBjAGYAdQBzAGkAbwBuAC4AYwBvAG0AAAAAAQq
```

```

AAAAAAAAAKAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAGAQIBAwEEAQUBBgEHAAdlZG10XzAzCWR1bGV
0ZV8wMgxtZXNzYWdlLW1haWwLcmVhZC11bnJlYWQJbGlrZS0tLTAAxAAAAA==)
format('truetype');
    font-weight: normal;
    font-style: normal;
}
.ddb-icons {
    font-family: 'ddb-icons' !important;
    speak: none;
    font-size: 55px;
    font-style: normal;
    font-weight: normal;
    font-variant: normal;
    text-transform: none;
    line-height: 1;
    -webkit-font-smoothing: antialiased;
    -moz-osx-font-smoothing: grayscale;
}
.e-message::before {
content: '\e702';
}
.e-edit::before {
content: '\e700';
}
.e-delete::before {
content: '\e701';
}
.e-read::before {
content: '\e703';
}
.e-like::before {
content: '\e704';
}
}
</style>

```

{% previewsample "page.domainurl/code-snippet/drop-down-button/default-cs17" %}

### Navigations

Actions in DropDownButton can be used to navigate to the other web page when action item is clicked. This can be achieved by providing link to the action item using `url` property.

In the following sample, navigation URL for Flipkart, Amazon, and Snapdeal action items are added using the `url` property:

### APP.VUE

```

<template>
<ejs-dropdownbutton :items='items' iconCss= 'e-cart-icon e-shopping'
:beforeItemRender='onBeforeItemRender'>Shop By</ejs-dropdownbutton>
</template>
<script>
import Vue from 'vue';
import { DropDownButtonPlugin } from "@syncfusion/ej2-vue-splitbuttons";
import { enableRipple } from '@syncfusion/ej2-base';
enableRipple(true);
Vue.use(DropDownButtonPlugin);

```

```
export default {
  data () {
    return {
      items:[
        {
          text: 'Flipkart',
          iconCss: 'e-cart-icon e-link',
          url: 'https://www.google.co.in/search?q=flipkart'
        },
        {
          text: 'Amazon',
          iconCss: 'e-cart-icon e-link',
          url: 'https://www.google.co.in/search?q=amazon'
        },
        {
          text: 'Snapdeal',
          iconCss: 'e-cart-icon e-link',
          url: 'https://www.google.co.in/search?q=snapdeal'
        }
      ]
    };
  },
  methods: {
    onBeforeItemRender: function(args) {
      args.element.getElementsByTagName('a')[0].setAttribute('target',
'_blank');
    }
  }
}
</script>
<style>
  @import '../node_modules/@syncfusion/ej2-base/styles/material.css';
  @import '../node_modules/@syncfusion/ej2-buttons/styles/material.css';
  @import '../node_modules/@syncfusion/ej2-popups/styles/material.css';
  @import '../node_modules/@syncfusion/ej2-splitbuttons/styles/material.css';
  @font-face {
    font-family: 'cart';
    src:
      url(data:application/x-font-ttf;charset=utf-
8;base64,AAEAAAAKAIAAAwAgTlMvMj0gSQ4AAAEoAAAAVmNtYXDNdEOdVAAABiAAAADZnbHlmGat
ngwAAAcgAAADYAGVhZBKtP4wAAADQAAAAANmhoZWEHmQNpAAAArAAAACRobXR4B+j//gAAAYAAAAA
IbG9jYQBsAAAAAHAAAAABmlheHABDwBQAAABCAAAACBuYWl1fiv2lQAAAqAAAAIBcG9zdIZzcJA
AAASkAAAAOgABAAADUv9qAFoEAP/+//wD7AABAAAAAAAAAAAAAAAAAAAgABAAAAQAA2UwSaF8
PPPUACwPoAAAAANfsfWUAAAAA19J9Zf/+AAAD7APdAAAAACAACAAAAAAAAAAEAAAACAEQAAwAAAAA
AAgAAAAoACgAAAP8AAAAAAAAAAQP0AZAABQAAAanoCvAAAAIwCegK8AAAB4AAxAQIAAAIABQMAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAUGZFZABA5wDnAANS/2oAWgPdAJYAAAABAAAAAAAAABAAAAAPo//4
AAAACAAAAAwAAABQAAwABAAAAFAAEACIAAAAEAAQAAQAA5wD//wAA5wD//wAAAAEABAAAAAEAAAA
AAAAAbAAAAAP//gAAA+wD3QAJABIAQwAAAJR4BMjY0JicOAQUeATI2NCYiBgEOAwclIgYXEx4BMwU
yFgcOAQclIgYXBhYXBt4BPwE2PwI2NxM+AycmIyIGAeABJzonJx0gJ/6jASc6Jyc6JwMXIDgZPyX
9giQkBkIIQyQBaCQgCQo/JP7RIxcBARcjAVgkQg0QDgkICgkNkw4xMBwCBScJE1UdJyc6JwEBJx0
dJyc6JycDZQg0PigBAY0k/uAkMAQnHRsmAQMTDA8QAQMBLCILhYWFxYhAYciNg4YDBMCAAAAEgD
eAAEAAAAAAAAAAQAAAAEAAAAAAAAEABAABAAEAAAAAAAAIABwAFAAEAAAAAAAAAMABAAMAAEAAAAAAQ
ABAAQAAEAAAAAAAAUACwAUAAEAAAAAAAAAYABAAfAAEAAAAAAAAoALAAjAAEAAAAAAAAsAEgBPAAMAAQ
JAAAAAgBhAAMAAQQJAAEACABjAAMAAQQJAAIADgBrAAMAAQQJAAACAB5AAMAAQQJAAQACACBAAM
AAQQJAAUAFgCJAAMAAQQJAAAYACACfAAMAAQQJAAoAWACnAAMAAQQJAAAsAJAD/IGNhcnRSZWd1bGF
yY2FyZGNhcnRWZXJzaW9uIDEuMGNhcnRb250IGdlbmVyYXRlZCB1c2luZyBTeW5jZnVzaW9uIE1
ldHJvIFN0dWRpb3d3dy5zeW5jZnVzaW9uLmNvbQAgAGMAYQByAHQAUGBlAGcAdQBsAGEAcgBjAGE
```

```

AcgB0AGMAYQByAHQAVgB1AHIAcwBpAG8AbgAgADEALgAwAGMAYQByAHQARgBvAG4AdAAgAGcAZQB
uAGUAcgBhAHQAZQBkACAAdQBzAGkAbgBnACAAUwB5AG4AYwBmAHUAcwBpAG8AbgAgAE0AZQB0AHI
AbwAgAFMAdAB1AGQAaQBvAHcAdwB3AC4AcwB5AG4AYwBmAHUAcwBpAG8AbgAuAGMabwBtAAAAAAI
AAAAAAAAACgAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAgECAQMAEHNob3BwaW5nLWNhcnQtMDUAAAA
A) format('truetype');
    font-weight: normal;
    font-style: normal;
  }
  .e-cart-icon {
    font-family: 'cart' !important;
    speak: none;
    font-size: 55px;
    font-style: normal;
    font-weight: normal;
    font-variant: normal;
    text-transform: none;
    line-height: 1;
    -webkit-font-smoothing: antialiased;
    -moz-osx-font-smoothing: grayscale;
  }
  .e-shopping::before {
    content: '\e700';
  }
  .e-link::before {
    content: '\e700';
  }
}
</style>

```

{% previewsample "page.domainurl/code-snippet/drop-down-button/default-cs18" %}

## Template

### Item templating

Popup items can be customized using the [beforeItemRender](#) event. The item render event triggers while rendering each popup action item. The event argument will be used to identify the action item and customize based on the requirement.

The following popup template is customized using `beforeItemRender` event by appending `span` and `div` element on each `li` rendering:

## APP.VUE

```

<template>
<ejs-dropdownbutton :items='items' iconCss= 'e-ddb-icons e-paste' cssClass=
'e-vertical' iconPosition= 'Top'
:beforeItemRender='onBeforeItemRender'>Paste</ejs-dropdownbutton>
</template>
<script>
import Vue from 'vue';
import { DropDownButtonPlugin } from "@syncfusion/ej2-vue-splitbuttons";
import { enableRipple } from '@syncfusion/ej2-base';
enableRipple(true);
Vue.use(DropDownButtonPlugin);
export default {
  data () {
    return {
      items:[

```

```

        {
            text: 'Edit'
        },
        {
            text: 'Cut'
        }
    ];
},
methods: {
    onBeforeItemRender: function(args) {
        if (args.item.text === 'Edit') {
            args.element.innerHTML = '<span></span><div><b>Paste
Text</b><div>Provides option to paste only the<br>selected
text.</div></div>';
            args.element.style.height = '80px';
            var span = args.element.children[0];
            span.setAttribute('class', 'e-cm-icons e-pastetext e-align');
            var div = args.element.children[1];
            div.setAttribute('class', 'e-div-align');
        } else {
            args.element.innerHTML = '<span></span><div><b>Paste
Special</b><div>Provides options to paste formulas,<br> values, comments,
validations etc...</div></div>';
            args.element.style.height = '80px';
            var span = args.element.children[0];
            span.setAttribute('class', 'e-cm-icons e-pastespecial e-
align');
            var div = args.element.children[1];
            div.setAttribute('class', 'e-div-align');
        }
    }
}
}
</script>
<style>
    @import '../node_modules/@syncfusion/ej2-base/styles/material.css';
    @import '../node_modules/@syncfusion/ej2-buttons/styles/material.css';
    @import '../node_modules/@syncfusion/ej2-popups/styles/material.css';
    @import '../node_modules/@syncfusion/ej2-
splitbuttons/styles/material.css';
    /* csslint ignore:start */
    @font-face {
        font-family: 'e-context-menu';
        src:
            url(data:application/x-font-ttf;charset=utf-
8;base64,AAEAAAAKAIAAAwAgTlMvMjJNRVMAAAEoAAAAVmNtYXDicOK6AAABjAAAADhnbHlmGcE
PFQAAAcwAAAMwaGVhZA69CA8AADQAAAAANmhoZWEH9AQEAAArAAACRobXR4DAAAAAAAAAYAAAAA
MbG9jYQDYAZgAAAEHEAAACG1heHABEGDAAAABCAAAACBuYW11xY1dlQAABPwAAAKFcG9zdPJwcMo
AAAEAAAAAABAAAAEAAAAAFWEAAAAAADlwABAAAAAAAAAAAAAAAAAAAAAwABAAAAAQAAgmhm818
PPPUACwQAAAAANYD4Y8AAAAA1gPhjwAAAAADlwP0AAAAACAACAAAAAAAAAAAAEAAAADALQABQAAAAA
AAgAAAAoACgAAAP8AAAAAAAAAAQQAABQAAAokCzAAAAI8CiQLMAAAB6wAyAQgAAAIABQMAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAUGZFZABA4mDiYQAAAAAXAQAAAAAAAAABAAAAAABAAAAAQAAAA
EAAAAAAAAAgAAAAAMAAAAUAMAAQAAABQABAakAAAAABAAEAAEAJOJh//8AAOJg//8AAAAABAAQAAAA
BAAIAAAAAANgBmAFAAAAAAOXA/QABAALAC0ATgCzAAABISchJzcVHwc/By8HDwYBFSE1MxEhESU
HFQ8GLwc/Bx8GJysBDw4RHw4zITM/DhEvDisBLw4rAQ8NAUQBd1xAfr0BAwQGBwgICgkJCAGBAM
BAQMEBgICQkKCAgHBgQD/qYB1179jQFoAQMEBgHCQkJCQgGBgQDAQEDBAYGCAkJCQkHBwYEA6y
9CgkICQgHBwcGBQQEAWMBAQEBAWMDBQUGBwcHCAkICQoCeAoJCAkIBwcHBgUEBAMDAQEBAQMDAwU

```

```
FBgcHBwgJCAkKvQqEBgUHBwcICQkJCgoKCwsLCwoKCgkJCQgHBwcFBgQBBYVRuh0FBQkIBwUFAgE
BAgUFBwgJCgkJCACGBAMBAQMEBgICQeifX39LwLRMwQFCAGHBQUCAQECBQUHCAgJCQkIBwUEAwE
BAwQFBwgJIgICAwQFBQYGBwgICakJCf0pCQkJCAGIBwYGBQUEAwICAgIDBAUFBgYHCAgICQkJAtc
JCQkICAgHBgYFBQQDAgIKCQkICAgHBgYFBAQDAgICAgMEBAUGBgICAgJCQAFAAAAAAXA/QABwA
PABCAOACdAAABHwIjPwIDMzczFzMDIycVITUzESERJQcVDwYvBz8HHwYnKwEPDhEfDjMhMz8OES8
OKwEvDisBDw0B/wQKK3MmBQ6dMyeHKDWCO90B1l79jQFoAQMEBgHCQkJCQgGBgQDAQEDBAYGCAk
JCQkHBwYEA6y9CgkICQgHBwcGBQqEAwMBAQEBAwMDBQUGBwcHCAkICQoCeAoJCAkIBwCHBgUEBAM
DAQEBAQMDAwUFBgCHBgJCAkKvQqEBgUHBwcICQkJCgoKCwsLCwoKCgkJCQgHBwcFBgQCFREigG4
SM/6wd3cBe/t9ff0vAtEzBAUICAcFBQIBAQIFBQcICakJCQgHBQQDAQEDBAUHCAkiAgIDBAUFBgY
HCAgICQkJ/SkJCQkICAgHBgYFBQQDAgICAgMEBQUGBgICAgJCQkC1wkJCQgICAcGBgUFBAMCAgo
JCQgICAcGBgUEBAMCAgICAwQEBQYGBwgICakJAAAAABIA3gABAAAAAEEAABAAAAAABAA8
AAQABAAAAAACAACAEABAAAAAADA8AFwABAAAAAEEA8AJgABAAAAAFAAsANQABAAAAA
GAA8AQABAAAAAKACwATwABAAAAAALABIAewADAAEECQAAAAIAjQADAAEECQABAB4AjwADAAE
ECQACAA4ArQADAAEECQADAB4AuWADAAEECQAEAB4A2QADAAEECQAFABYA9wADAAEECQAGAB4BDQA
DAAEECQAKAFgBKwADAAEECQALACQBgyBDb250ZXh0TWVudSAoMilSZWdlbGFyQ29udGV4dE1lbnU
gKDIpQ29udGV4dE1lbnUgKDIpVmVyc2lvbiAxLjBDb250ZXh0TWVudSAoMilGb250IGdlbmVyYXR
lZCB1c2luZyBTew5jZnVzaW9uIE1ldHJvIFN0dWRpb3d3dy5zeW5jZnVzaW9uLmNvbQAgAEMAbwB
uAHQAZQB4AHQATQBlAG4AdQAgACgAMgApAFIAZQBnAHUAbABhAHIAQwBvAG4AdABlAHgAdABNAGU
AbgB1ACAACAAYACkAQwBvAG4AdABlAHgAdABNAGUAbgB1ACAACAAYACkAVgBlAHIAcwBpAG8AbG9
gADEALgAwAEMAbwBuAHQAZQB4AHQATQBlAG4AdQAgACgAMgApAEYAbwBuAHQATQBlAHIAAHIA
AYQB0AGUAZAAGAHUAcwBpAG4AZwAgAFMAEQBuAGMAZgB1AHMAAQwBvAG4ATQBlAHIAAHIAAHIA
TAHQAdQkAgkAbwB3AHCAAdwAuAHMAEQBuAGMAZgB1AHMAAQwBvAG4ALgBjAG8AbG9wAAAAACAAAA
AAAOAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAMBAgEDAQAD01UX1Bhc3RlU3B1Y2lhbAxBNVF9QYXN
0ZVRleHQAAA==) format('trueType');
font-weight: normal;
font-style: normal;
}
/* csslint ignore:stop */
.e-cm-icons {
font-family: 'e-context-menu';
font-style: normal;
font-variant: normal;
font-weight: normal;
line-height: 1;
text-transform: none;
}
@font-face {
font-family: 'ddb-icons';
src:
url(data:application/x-font-ttf;charset=utf-
8;base64,AAEAAAKAIAAAwAgTlMvMj0gSRkAAAEoAAAAVmNtYXDNdE+dkAAABlAAAAADxnbHlmlh3
3NQAAAdwAAAjMAgVhZBKOK9sAAADQAAAAANmhoZWEHeANwAAAArAAAAACRobXR4E6AAAAAAYAAAA
UbG9jYQGOAegAAAHQAAAADG1heHABEWBlAAABCAAAACBuYW1l1LBM9QAABCgAAAI9cG9zdMjntbU
AAAZoAAAAUAABAAADUv9qAFoEAAAAAADygABAAAAAABQABAAAAAQAAojXaQl8
PPPUACwPoAAAAANfSc4gAAAAA19JziAAA//oDyGPsAAAACAACAAAAAEEAFAAFKABAAAAAA
AAgAAAAoACgAAAP8AAAAAQAAPtAZAABQAAAnoCvAAAAIwCegK8AAAB4AAxAQIAAAIABQMAAAA
AAAAAAAAAAAAAAAAAAAAAAAAUGZFZABA5wDnAwNS/2oAwGPsAJYAAAABAAAAAABAAAAAPoAAA
D6AAAA+gAAAPoAAAAAACAAAAAwAAABQAAAwABAAAAFAAEACgAAAAEAQAQA5wP//wAA5wD//wA
AAEAABAAAAEAAGADAAQAAAAAI4AwGEAASyAAwAA//oDNQPsAA4AHQBYAAALHgEOAScmJy4BNz4
BMzIFFgYHBgcGLG2NzYzYmYBHGEXDgEHDgIWFxYXFjY3NjQ3PgE3HgEXFhQXHG3PgE3PgE
uAScuAScuASc+ATc+AQcLASYWAVEfFx06IBkNCQIHcy8bCQG9BwIJDRkgOhoXHwoKgi/+TRlRdyE
OIxo+ExckFAQMfikwVhcmBwYlFRYkBWCMF1YwFCALDAQUIxcUPhojDiAOUR4cAQvEwwsB6gtDTyc
JCBsSKxYhJ0gWKxIaCQknUEILAYcCf2TPI0w2HBUMdg0sOzsakQ4ONzcniiYXNBgYNBcmiyc3OA8
GHRQaOzssDQ4mFRw2TiLOZGdBA/5vAZEDQQAEEAAAAAQA+kABQANABCAHwAAARUzFSErAYERIZU
jNSEBIREhESMVITUjMyMVITUjNSMC733+iT8B9D4+/oj+igE4AXc//c4++j8BOT+7AbZ8+gF2/ks
Bdz4//ksB9AF2fHw+Pj8AAAAIAAAAA7cD6QACACQAAAEhEwMOAQcVITUmJyY1ND8BIRcWFxYVFAc
gKwEVITUmJyYnASMCKP8AguQrOy0BGkIRHREkASstEgEEDhQxEQGaJxUcLP7PDAFNAVL+PHBHCBS
```

```

bBgsUKR8wX3owBg4NFgsQGxsDFx1zAyMAAAACAAAAAPKA+oAAgATAAABFxEBDgEHHgEXETMRMxE
zETM1IQL+zP1abpADA5t0f2F+XP41AfbMAZgBJwmYcHSbA/48A2r8lgNqfgAAAAASAN4AAQAAAAA
AAAAABAAAAAQAAAAAAQAJAAEAAQAAAAAAAgAHAAoAAQAAAAAAAwAJABEAAQAAAAABAAJABoAAQA
AAAAABQALACMAAQAAAAABgAJAC4AAQAAAAAACgAsADcAAQAAAAAACwASAGMAAwABBAkAAAACAHU
AAwABBAkAAQASAHcAAwABBAkAAgAOAIkAAwABBAkAAwASAJcAAwABBAkABAASAKkAAwABBAkABQA
WALsAAwABBAkABgASANEAAwABBAkACgBYAOMAawABBAkACwAkATsgZGRiLWljB25zUmVndWxhcmR
kYilpY29uc2RkYilpY29uc1ZlcnNpb24gMS4wZGRiLWljB25zRm9udCBnZW5lcmF0ZWQgdXNpbmc
gU3luY2Zlc2l2b20AIAABkAGQAYgAtAGkAYwBvAG4AcwBSAGUAZwB1AGwAYQByAGQAZABiAC0AaQBjAG8AbgBzAGQAZABiAC0AaQBjAG8AbgBzAFY
AZQByAHMAaQBvAG4AIAAxAAC4AMABkAGQAYgAtAGkAYwBvAG4AcwBGAG8AbgB0ACAAZwB1AG4AZQB
yAGEAdABLAGQAIAB1AHMAaQBvAGcAIABTAHkAbgBjAGYAdQBzAGkAbwBuACAATQBlAHQAcgBvACA
AUwB0AHUAZABpAG8AdwB3AHcALgBzAHkAbgBjAGYAdQBzAGkAbwBuAC4AYwBvAG0AAAAAAgAAAA
AAAAKAAAAAAAAAAAAAAAAAAAAAAAAAFAQIBAwEEAQUBBgADY3V0CHBhc3RlXzAxBGZvbnQ
OcGFyYS1tYXJrLS0tMDMAAA==) format('trueType');
    font-weight: normal;
    font-style: normal;
  }
  .e-ddb-icons {
    font-family: 'ddb-icons' !important;
    speak: none;
    font-size: 55px;
    font-style: normal;
    font-weight: normal;
    font-variant: normal;
    text-transform: none;
    line-height: 1;
    -webkit-font-smoothing: antialiased;
    -moz-osx-font-smoothing: grayscale;
  }
  .e-pastespecial::before {
    content: '\e260';
  }
  .e-pastetext::before {
    content: '\e261';
  }
  .e-paste::before {
    content: '\e701';
  }
  .e-dropdown-popup ul {
    max-width: 400px;
    white-space: nowrap;
  }
  .e-align {
    float: left;
    width: 15%;
    margin-top: 15px;
    font-size: 45px;
    color: grey;
  }
  .e-div-align {
    float: right;
    width: 75%;
    line-height: 23px;
    margin: 0 15px 0 0;
  }
</style>

```



```
{% previewsample "page.domainurl/code-snippet/drop-down-button/default-cs19" %}
```

### Popup templating

The whole popup can be customized as per the requirement and it can be customized by handling it in [target](#) property.

In the following sample, the whole popup item is customized as table template by giving `div` as target and it can be achieved using `target` property.

### APP.VUE

```
<template>
  <div>
    <div id="target" style='border: 1px solid grey;'>
      <table>
        <caption style='height: 18px; background-color: #e0e0e0;'><b>Insert
Table</b></caption>
        <tr class='e-row'><td class='e-cell'></td><td class='e-
cell'></td><td class='e-cell'></td><td class='e-cell'></td><td class='e-
cell'></td><td class='e-cell'></td></tr>
        <tr class='e-row'><td class='e-cell'></td><td class='e-
cell'></td><td class='e-cell'></td><td class='e-cell'></td><td class='e-
cell'></td><td class='e-cell'></td></tr>
        <tr class='e-row'><td class='e-cell'></td><td class='e-
cell'></td><td class='e-cell'></td><td class='e-cell'></td><td class='e-
cell'></td><td class='e-cell'></td></tr>
        <tr class='e-row'><td class='e-cell'></td><td class='e-
cell'></td><td class='e-cell'></td><td class='e-cell'></td><td class='e-
cell'></td><td class='e-cell'></td></tr>
        <tr class='e-row'><td class='e-cell'></td><td class='e-
cell'></td><td class='e-cell'></td><td class='e-cell'></td><td class='e-
cell'></td><td class='e-cell'></td></tr>
        <tr class='e-row'><td class='e-cell'></td><td class='e-
cell'></td><td class='e-cell'></td><td class='e-cell'></td><td class='e-
cell'></td><td class='e-cell'></td></tr>
      </table>
    </div>
    <ejs-dropdownbutton target= '#target' iconCss= 'e-icons e-table'
cssClass= 'e-vertical' iconPosition= 'Top'>Table</ejs-dropdownbutton>
  </div>
</template>
<script>
import Vue from 'vue';
import { DropDownButtonPlugin } from "@syncfusion/ej2-vue-splitbuttons";
import { enableRipple } from '@syncfusion/ej2-base';
enableRipple(true);
Vue.use(DropDownButtonPlugin);
export default {}
</script>
<style>
  @import '../node_modules/@syncfusion/ej2-base/styles/material.css';
  @import '../node_modules/@syncfusion/ej2-buttons/styles/material.css';
  @import '../node_modules/@syncfusion/ej2-popups/styles/material.css';
  @import '../node_modules/@syncfusion/ej2-
splitbuttons/styles/material.css';
```

```

.e-cell {
  border: 1px solid grey;
  padding: 8px;
}
.e-table::before {
  content: '\e705';
}
.e-row {
  padding-left: 3px;
  padding-right: 3px;
}

```

{% previewsample "page.domainurl/code-snippet/drop-down-button/default-cs20" %}

### Separator

The Separators are the horizontal lines that are used to separate the popup items. You cannot select the separators. You can enable separators to group the popup items using the `separator` property.

In the following sample, cut, copy, and paste popup items are grouped using the separator property:

### APP.VUE

```

<template>
<ejs-dropdownbutton :items='items'>Clipboard</ejs-dropdownbutton>
</template>
<script>
import Vue from 'vue';
import { DropDownButtonPlugin } from "@syncfusion/ej2-vue-splitbuttons";
import { enableRipple } from '@syncfusion/ej2-base';
enableRipple(true);
Vue.use(DropDownButtonPlugin);
export default {
  data () {
    return {
      items:[
        {
          text: 'Cut',
          iconCss: 'e-db-icons e-cut'
        },
        {
          text: 'Copy',
          iconCss: 'e-icons e-copy'
        },
        {
          text: 'Paste',
          iconCss: 'e-db-icons e-paste'
        },
        {
          separator: true
        },
        {
          text: 'Font',
          iconCss: 'e-db-icons e-font'
        },
        {

```

```

    text: 'Paragraph',
    iconCss: 'e-db-icons e-paragraph'
  });
}
}
</script>
<style>
  @import '../node_modules/@syncfusion/ej2-base/styles/material.css';
  @import '../node_modules/@syncfusion/ej2-buttons/styles/material.css';
  @import '../node_modules/@syncfusion/ej2-popups/styles/material.css';
  @import '../node_modules/@syncfusion/ej2-splitbuttons/styles/material.css';
  @font-face {
    font-family: 'ddb-icons';
    src:
      url(data:application/x-font-ttf;charset=utf-8;base64,AAEAAAAKAIAAAwAgTlMvMj0gSRkAAAEoAAAVmNtYXdDnE+dkAAABlAAAADxnbHlmh3
3NQAAAdwAAAJMaGVhZBKOK9sAAADQAAAAANmhoZWEHeANwAAAArAAAACRobXR4E6AAAAAAYAAAA
UbG9jYQGOAegAAAHQAAAAADG1heHABEWBlAAABCAAAACBuYW1l1LBM9QAABCgAAAI9cG9zdmJntbU
AAAZoAAAAUAABAAADUv9qAFoEAAAAAADyGABAAAAAAAAAAAAAAAAABQABAAAAQAQAAojXaQl8
PPPUACwPoAAAAANfSc4gAAAAA19JziAAA//oDyGPsAAAAACAACAAAAAAAAAAAAEAAAFaFkABAAAAA
AAgAAAAoACgAAAP8AAAAAAAAAAQPtAZAABQAAANoCvAAAAIwCegK8AAAB4AAxAQIAAAIABQMAAAA
AAAAAAAAAAAAAAAAAAAAAAAAUGZFZABA5wDnAwNS/2oAWGPsAJYAAAABAAAAAAAAABAAAAAPoAAA
D6AAAA+gAAAPoAAAAAACAAAAAwAAABQAAWABAAAAFAAEACgAAAAEAAQAAQAA5wP//wAA5wD//wA
AAAEABAAAAEAAAgADAAQAAAAAAAI4AwgEAA5YAAwAA//oDNQPsAA4AHQBYAAALHgEOAScmJy4BNz4
BMzIFFGgYHBGcGLgE2NzYzMyYBHgEXDgEHDgEHDgIWFxYXFjY3NjQ3PgE3HgEXFhQXHgE3PgE3PgE
uAScuAScuASc+ATc+AQcLASYWAVEfFxo6IBkNCQIHcy8bCQG9BwIJDRkgOhoXHwoKgi/+TR1RDyE
OIxo+ExckFAQMFIkVhCMBWylFRYkBWCMF1YwFCALDAQUIxcUPhojDiAOUR4cAQvEwwsB6gtDTyc
JCBsSKxYhJ0gWKxIaCQknUEILAYcCf2TPI0w2HBUMdg0sOzsakQ4ONzcniiYXNBgYNBcmiyc3OA8
GHRQaOzssDQ4mFRw2TiLOZGdBA/5vAZEDQQAIAAAAAAQAa+kABQANABcAHwAAARUzFSErAYERIZU
jNSEBIREhESMVITUjMyMVITUjNSMC733+iT8B9D4+/oj+igE4AXc//c4++j8BOT+7AbZ8+gF2/ks
Bdz4//ksB9AF2fHw+Pj8AAAAIAAAAAA7cD6QACACQAAAEhEwMOAQcVITUmJyY1ND8BIRcWFxYVFAc
GKwEVITUmJyYnASMCKP8AguQrOy0BGkIRHREkASstEgEEDhQxEQGAJxUcLP7PDAFNAVL+PHBHCBs
bBgsUKR8wX3owBg4NFgsQGxsDFx1zAyMAAAACAAAAAAPKA+oAAgATAAABFxEBDgEHHgEXETMRMxE
zETM1IQL+zPlabpADA5t0f2F+XP41AfbMAZgBJwmYcHSbA/48A2r8lgNqfgAAAAASAN4AAQAAAAA
AAAAABAAAAAQAAAAAAQAIAAEAAQAAAAAAAgAHAaoAAQAAAAAAAwAJABEAAQAAAAAABAAJABoAAQA
AAAAABQALACMAAQAAAAAABgAJAC4AAQAAAAAACgAsADcAAQAAAAAACwASAGMAAwABBakAAAACAHU
AAwABBakAAQASAHCAAwABBakAAgAOAIkAAwABBakAAwASAJcAAwABBakABAASAKkAAwABBakABQA
WALSAAwABBakABgASANEAAwABBakACgBYAOMAAwABBakACwAkATsgZGRiLWljb25zUmVndWxhcmR
kYilpY29uc2RkYi1pY29uc1ZlcnNpb24gMS4wZGRiLWljb25zRm9udCBnZW5lcmF0ZWQgdXNpbmc
gU3luY2Z1c2l2b2lBNzZXRYbyBTdHVkaW93d3cuc3luY2Z1c2l2b2lBNzZXRYbyBTdHVkaW93d3c
vAG4AcwBSAGUAZwBlAGwAYQByAGQAZABiAC0AaQBjAG8AbgBzAGQAZABiAC0AaQBjAG8AbgBzAFY
AZQByAHMAaQBvAG4AIAAxAC4AMABkAGQAYgAtAGkAYWwvAG4AcwBGAG8AbgB0ACAAZwBlAG4AZQB
yAGEAdABlAGQAIABlAHMAaQBuAGcAIABTAHkAbgBjAGYAdQBzAGkAbwBuACAATQBlAHQAcgBvACA
AUwB0AHUAZABpAG8AdwB3AHcALgBzAHkAbgBjAGYAdQBzAGkAbwBuAC4AYWwvAG0AAAAAAGAAAA
AAAAKAAAAAAAAAAAAAAAAAAAAAAAAAAAAAFaQIBAwEEAQUBBgADY3V0CHBhc3RlXzAxBGZvbG9u
OcGFyYSltYXJrLS0tMDMAAA==) format('truetype');
    font-weight: normal;
    font-style: normal;
  }
  .e-db-icons {
    font-family: 'ddb-icons' !important;
    speak: none;
    font-size: 55px;
    font-style: normal;
    font-weight: normal;
  }

```

```
font-variant: normal;
text-transform: none;
line-height: 1;
-webkit-font-smoothing: antialiased;
-moz-osx-font-smoothing: grayscale;
}
.e-edit::before {
content: '\ea9a';
}
.e-cut::before {
content: '\e700';
}
.e-copy::before {
content: '\e70a';
}
.e-paste::before {
content: '\e701';
}
.e-font::before {
content: '\e702';
}
.e-paragraph::before {
content: '\e703';
}
}
</style>
```

{% previewsample "page.domainurl/code-snippet/drop-down-button/default-cs21" %}

See Also

- [Integration with ListView component](#)

### Accessibility in Vue Drop down button component

The Drop down button component followed the accessibility guidelines and standards, including [ADA](#), [Section 508](#), [WCAG 2.2](#) standards, and [WCAG roles](#) that are commonly used to evaluate accessibility.

The accessibility compliance for the Drop down button component is outlined below.

| Accessibility Criteria | Compatibility |

| -- | -- |

| [WCAG 2.2](#) Support |  |

| [Section 508](#) Support |  |

| Screen Reader Support |  |

| Right-To-Left Support |  |

| Color Contrast |  |

| Mobile Device Support |  |

| Keyboard Navigation Support |  |

| [Accessibility Checker](#) Validation |  |

| [Axe-core](#) Accessibility Validation |  |

<style>

```
.post .post-content img {  
display: inline-block;  
margin: 0.5em 0;  
}
```

</style>

<div> - All features of the component meet the requirement.</div>

<div> - Some features of the component do not meet the requirement.</div>

<div> - The component does not meet the requirement.</div>

#### WAI-ARIA attributes

The Drop down button component followed the [WAI-ARIA] patterns to meet the accessibility. The following ARIA attributes are used in the Drop down button component:

| Attributes | Purpose |

| --- | --- |

| **role** | Indicates the Drop down button component as **button**, Drop down button popup as **menu**, and the dropdown popup action items as **menuitem**. |

| **aria-haspopup** | Indicates the availability of the popup element. |

| **aria-expanded** | Indicates whether the popup can be expanded or collapsed, as well as indicates whether its current state is expanded or collapsed. |

| **aria-owns** | Identifies an elements in order to define a visual, functional, or contextual parent/child relationship between DOM elements where the DOM hierarchy cannot be used to represent the relationship. |

| **aria-disabled** | Indicates that the element is perceivable but disabled, so it is not editable or otherwise operable. |

### Keyboard interaction

The Dropdown button component followed the [keyboard interaction] guideline, making it easy for people who use assistive technologies (AT) and those who completely rely on keyboard navigation. The following keyboard shortcuts are supported by the Drop down button component.

| **Press** | **To do this** |

| --- | --- |

| **Esc** | **Closes the popup.** |

| **Enter** | **Opens the popup, or activates the highlighted item and closes the popup.** |

| **Space** | **Opens the popup.** |

| **Up** | **Navigates up or to the previous action item.** |

| **Alt + Up Arrow** | **Closes the popup.** |

| **Alt + Down Arrow** | **Opens the popup.** |

### Ensuring accessibility

The Drop down component's accessibility levels are ensured through an [accessibility-checker](#) and [axe-core](#) software tools during automated testing.

The accessibility compliance of the Drop down button component is shown in the following sample. Open the [sample](#) in a new window to evaluate the accessibility of the Drop down button component with accessibility tools.

{% previewsample "page.domainurl/code-snippet/drop-down-button/default-cs1" %}

See also

- [Accessibility in Syncfusion Vue components](#)

### Style and appearance in Vue Drop down button component

To modify the DropDownButton appearance, you need to override the default CSS of DropDownButton component. Please find the list of CSS classes and its corresponding section in DropDownButton. Also, you have an option to create your own custom theme for the controls using our [Theme Studio](#).

CSS Class | Purpose of Class

| .e-dropdown-btn | To customize the dropdown button

| .e-dropdown-btn: hover | To customize the dropdown button on hover

| .e-dropdown-btn.e-active | To customize the dropdown button on active

| .e-dropdown-popup | To customize the dropdown button pop up

| .e-dropdown-popup ul .e-item: hover | To customize the dropdown button pop up items on hover

| .e-dropdown-popup ul .e-item: active | To customize the dropdown button pop up items on active

### How To

#### Change caret icon in Vue Drop down button component

Dropdown arrow can be customized on popup open and close. It can be handled in [beforeOpen](#)

and [beforeClose](#) event.

In the following example, the up arrow is updated on popup close and down arrow is updated on popup open using `beforeOpen` and `beforeClose` event by adding and removing `e-caret-up` class.

#### APP.VUE

```
<template>
<ejs-dropdownbutton id="ddBtn" :items='items' :beforeOpen='onBeforeOpen'
:beforeClose='onBeforeClose'>Clipboard</ejs-dropdownbutton>
</template>
<script>
import Vue from 'vue';
import { DropDownButtonPlugin } from "@syncfusion/ej2-vue-splitbuttons";
import { enableRipple } from '@syncfusion/ej2-base';
enableRipple(true);
Vue.use(DropDownButtonPlugin);
export default {
  data () {
    return {
      items:[
        {
          text: 'Cut'
        },
        {
          text: 'Copy'
        },
        {
          text: 'Paste'
        }
      ]
    };
  },
  methods: {
    // Removing 'e-caret-up' class.
    onBeforeClose: (args) => {
      document.getElementById('ddBtn').ej2_instances[0].cssClass = '';
    },
    // Adding 'e-caret-up' class.
    onBeforeOpen: (args) => {
      document.getElementById('ddBtn').ej2_instances[0].cssClass = 'e-
caret-up';
    }
  }
}
</script>
<style>
@import '../node_modules/@syncfusion/ej2-base/styles/material.css';
@import '../node_modules/@syncfusion/ej2-buttons/styles/material.css';
@import '../node_modules/@syncfusion/ej2-popups/styles/material.css';
@import '../node_modules/@syncfusion/ej2-
splitbuttons/styles/material.css';
.e-caret {
  transform: rotate(0deg);
  transition: transform 200ms ease-in-out;
}
.e-caret-up .e-caret {
  transform: rotate(180deg);
}
```

```
}
</style>
```

{% previewsample "page.domainurl/code-snippet/drop-down-button/default-cs3" %}

Create dropdownbutton with rounded corner in Vue Drop down button component

DropDownButton with rounded corner can be achieved by adding `border-radius` CSS property to button element.

In the following example, `e-round-corner` class is defined with `5px border-radius` property and added that class to button element using `cssClass` property.

#### APP.VUE

```
<template>
<ejs-dropdownbutton :items='items' cssClass='e-round-corner'>Clipboard</ejs-
dropdownbutton>
</template>
<script>
import Vue from 'vue';
import { DropDownButtonPlugin } from "@syncfusion/ej2-vue-splitbuttons";
import { enableRipple } from '@syncfusion/ej2-base';
enableRipple(true);
Vue.use(DropDownButtonPlugin);
export default {
  data () {
    return {
      items:[
        {
          text: 'Cut'
        },
        {
          text: 'Copy'
        },
        {
          text: 'Paste'
        }
      ]
    };
  }
}
</script>
<style>
@import '../node_modules/@syncfusion/ej2-base/styles/material.css';
@import '../node_modules/@syncfusion/ej2-buttons/styles/material.css';
@import '../node_modules/@syncfusion/ej2-popups/styles/material.css';
@import '../node_modules/@syncfusion/ej2-
splitbuttons/styles/material.css';
.e-round-corner {
  border-radius: 5px;
}
</style>
```

{% previewsample "page.domainurl/code-snippet/drop-down-button/default-cs4" %}



Create right to left dropdownbutton in Vue Drop down button component

DropDownButton component has RTL support. This can be achieved by setting [enableRtl](#) as true.

The following example illustrates how to enable right-to-left support in DropDownButton component.

#### APP.VUE

```
<template>
<ejs-dropdownbutton :items='items' iconCss='ddb-icons e-message'
enableRtl='true'>Message</ejs-dropdownbutton>
</template>
<script>
import Vue from 'vue';
import { DropDownButtonPlugin } from "@syncfusion/ej2-vue-splitbuttons";
import { enableRipple } from '@syncfusion/ej2-base';
enableRipple(true);
Vue.use(DropDownButtonPlugin);
export default {
  data () {
    return {
      items:[
        {
          text: 'Edit'
        },
        {
          text: 'Delete'
        },
        {
          text: 'Mark as Read'
        },
        {
          text: 'Like Message'
        }
      ]
    };
  }
}
</script>
<style>
@import '../node_modules/@syncfusion/ej2-base/styles/material.css';
@import '../node_modules/@syncfusion/ej2-buttons/styles/material.css';
@import '../node_modules/@syncfusion/ej2-popups/styles/material.css';
@import '../node_modules/@syncfusion/ej2-splitbuttons/styles/material.css';
@font-face {
  font-family: 'e-db-icons';
  src:
    url(data:application/x-font-ttf;charset=utf-
8;base64,AAEAAAIAIAAAwAgTlMvMj0jSRoAAAEoAAAVmNtYXdnFudgAAABkAAAADpnbHlmSrK
TCAAAAdgAAAC4aGVhZBKtK8cAAADQAAAAANmhoZWEHmQNtAAAArAAAACRobXR4D7gAAAAAYAAAA
QbG9jYQB4ADoAAAHMAAAACmlheHABEAAAYAAABCAAAACBuYW1lH00mDAAAAPAAAAJJcG9zdIwSr0
AAATcAAAATQABAAADUv9qAFoEAAAA//4D6gABAAAAAAAAAAAAAAAAABABAAAAQAAGc/PS18
PPPUACwPoAAAAANfSc3wAAAAA19JzfAAAAAAD6gPqAAAAACAACAAAAAAAAAAAAEAAAwAAgAAAA
AAgAAAAoACgAAAP8AAAAAAAAAAQPuAZAABQAAANoCvAAAAIwCegK8AAAB4AAxAQIAAAIABQMAAAA
AAAAAAAAAAAAAAAAAAAAAAAAUGZFZABA5wPnBQNS/2oAWgPqAJYAAAABAAAAAAAAABAAAAAPoAAA
D6AAAA+gAAAAAAAAIAAADAAAAFAADAAEAAAAUAAQAJgAAAAQABAABAADnBf//AADnA//AAAAQA
EAAAAAQACAMAAAAAAAAAAHAA6AFwAAAAACAAAAAPqA2UABgAKAAA3IREjCQEjBRcBIQID6AL+Dv4
NAQEY3QG4/I+IAsL+GAHonroBcwAAAAIAAAAAA8YD6gAFAAoAADchESMJASUHCQImA6AD/jL+MQE
```

```

EywGWAzb+agICX/4+AcLXsv6cAWQBZAAAAAEAAAAAA+oD6gALAAATCQEXCQE3CQEnCQECATP+zci
BMgEzwf7OATLB/s3+zgMp/s3+zsIBM/7NwgEyATPB/s4BMgAAAAASAN4AAQAAAAAAAAABAAAAQA
AAAAAAQAKAAEAAQAAAAAAAgAHAAsAAQAAAAAAAwAKABIAAQAAAAAABAABAAQAAAAAABQALACY
AAQAAAAAABgAKADEAAQAAAAAACgAsAdSAAQAAAAAACwASAGcAAwABBAkAAAAACAHkAAwABBAkAAQA
UAHsAAwABBAkAAgAOAI8AAwABBAkAAwAUAJ0AAwABBAkABAAUALEAAwABBAkABQAWAMUAAwABBAk
ABgAUANsAAwABBAkACgBYAO8AAwABBAkACwAkAUcgZS1kYilpY29uc1JlZ3VsYXJlLWw1j25
zZS1kYilpY29uc1ZlcnNpb24gMS4wZS1kYilpY29uc0ZvbnQgZ2VuZXXJhdGVkIHVzaW5nIFN5bmN
mdXNpb24gTWV0cm8gU3RlZGlvd3d3LnN5bmNmdXNpb24uY29tACAAZQAtAGQAYgAtAGkAYwBvAG4
AcwBSAGUAZwB1AGwAYQByAGUALQBkAGIALQBpAGMAbwBuAHMAZQAtAGQAYgAtAGkAYwBvAG4AcwB
WAGUAcgBzAGkAbwBuACAAMQAUADAAZQAtAGQAYgAtAGkAYwBvAG4AcwBGAG8AbgB0ACAAZwB1AG4
AZQByAGEAdAB1AGQAIAB1AHMAaQBUAGcAIABTAHkAbgBjAGYAdQBzAGkAbwBuACAATQB1AHQAcgB
vACAAUwB0AHUAZABpAG8AdwB3AHcALgBzAHkAbgBjAGYAdQBzAGkAbwBuAC4AYwBvAG0AAAAAAGa
AAAAAAAAKAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAEAQIBAwEEAQADG1lc3NhZ2UtZWw1j25
kLXVucmVhZAZkZWw1dGUAAAAAAAA==) format('truetype');

font-weight: normal;
font-style: normal;
}
.ddb-icons {
font-family: 'e-db-icons' !important;
speak: none;
font-size: 55px;
font-style: normal;
font-weight: normal;
font-variant: normal;
text-transform: none;
line-height: 1;
-webkit-font-smoothing: antialiased;
-moz-osx-font-smoothing: grayscale;
}
.e-message::before {
content: '\e703';
}
</style>

```

```
{% previewsample "page.domainurl/code-snippet/drop-down-button/default-cs5" %}
```

## Customize icon and width in Vue Drop down button component

Width of the DropDownButton can be customized by setting required width to the dropdown element.

The following UI can be achieved by setting `iconPosition` as `Top`, width as `85px` and size of the font icon as `40px` by adding `e-custom` class.

## APP.VUE

```
<template>
<ejs-dropdownbutton :items='items' iconCss='e-icons e-search' cssClass='e-
custom' iconPosition='Top'          content='Find & Select'></ejs-
dropdownbutton>
</template>
<script>
import Vue from 'vue';
import { DropDownButtonPlugin } from "@syncfusion/ej2-vue-splitbuttons";
import { enableRipple } from '@syncfusion/ej2-base';
enableRipple(true);
Vue.use(DropDownButtonPlugin);
export default {
```

```

    data () {
      return {
        items:[
          {
            text: 'Find'
          },
          {
            text: 'Replace'
          },
          {
            text: 'Go To'
          },
          {
            text: 'Go To Special'
          }
        ]
      };
    }
  }
</script>
<style>
  @import '../node_modules/@syncfusion/ej2-base/styles/material.css';
  @import '../node_modules/@syncfusion/ej2-buttons/styles/material.css';
  @import '../node_modules/@syncfusion/ej2-popups/styles/material.css';
  @import '../node_modules/@syncfusion/ej2-splitbuttons/styles/material.css';
  .e-search::before {
    content: '\e993';
  }
  .e-dropdown-btn.e-custom {
    width: 85px;
  }
  .e-dropdown-btn.e-custom .e-search::before {
    font-size: 40px;
  }
</style>

```

{% previewsample "page.domainurl/code-snippet/drop-down-button/default-cs6" %}

### Disable a dropdownbutton in Vue Drop down button component

DropDownButton component can be enabled/disabled by giving [disabled](#) property. It can be disabled by setting disabled property as **true**.

#### APP.VUE

```

<template>
  <ejs-dropdownbutton :items='items' iconCss='ddb-icons e-message'
    disabled='true'>Message</ejs-dropdownbutton>
</template>
<script>
import Vue from 'vue';
import { DropDownButtonPlugin } from "@syncfusion/ej2-vue-splitbuttons";
import { enableRipple } from '@syncfusion/ej2-base';
enableRipple(true);
Vue.use(DropDownButtonPlugin);
export default {
  data () {

```

```
return {
  items:[
    {
      text: 'Edit'
    },
    {
      text: 'Delete'
    },
    {
      text: 'Mark as Read'
    },
    {
      text: 'Like Message'
    }
  ]
};
}
</script>
<style>
@import '../node_modules/@syncfusion/ej2-base/styles/material.css';
@import '../node_modules/@syncfusion/ej2-buttons/styles/material.css';
@import '../node_modules/@syncfusion/ej2-popups/styles/material.css';
@import '../node_modules/@syncfusion/ej2-splitbuttons/styles/material.css';
@font-face {
  font-family: 'e-db-icons';
  src:
url(data:application/x-font-ttf;charset=utf-8;base64,AAEAAAAKAIAAwAgTlMvMj0jSRoAAAEoAAAAMVnTYXDnFudgAAABkAAAAADpnbHlmSrK
TCAAAAdgAAAC4AGvhZBKtK8cAADQAAAAANmhoZWEhmQNtAAAArAAAACRobXR4D7gAAAAAYAAAAA
QbG9jYQB4ADoAAAHMAAACmlheHABEAAYAAABC AAAACBuYW1lH00mDAAPAAAAJJCg9zdIwksr0
AAATcAAAATQABAAADUv9qAFoEAAAA//4D6gABAAAAAAAAAAAAAAAAAAAAAAAAABAABAAAAAQAAgc/PS18
PPPUACwPoAAAAANfSc3wAAAAA19JzfAAAAAAD6gPqAAAAACAACAAAAAAAAAAAAAEAAAAEEAAwAAgAAAAA
AAgAAAAAoACgAAP8AAAAAAAAAAAAQPuAZAABQAAAAnoCvAAAAAIwCegK8AAAB4AAxAQIAAAIABQMAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAUGZFZABA5wPnBQNS/2oAWGPqAJYAAAAABAAAAAAAAABAAAAAPoAAA
D6AAAA+gAAAAAAAAIAAAADAFAADAAAEAAAAUAQAjgAAAAQABAABAADnBf//AADnA///AAAAAQAE
AAAAAQACAAMAAAAAAAAAHAA6AFwAAAAACAAAAAPqa2UABgAKAAA3IREjCQEjBRcBIQID6AL+Dv4
NAQEY3QG4/I+IASL+GAHonroBcwAAAAIAAAAAA8YD6gAFAAoAADchESMJASUHCQImA6AD/jL+MQE
EywGWAZb+agICX/4+AcLXsv6cAWQBZAAAAAEAAAAA+oD6gALAAATCQEXCQE3CQEnCQECATP+zCI
BMgEzwf7OATLB/s3+zgMp/s3+zsIBM/7NwgEyATPB/s4BMgAAAAASAN4AAQAAAAAAAAABAAAAAQAE
AAAAAAQAKAAEAAQAAAAAAAAAgAHAAsAAQAAAAAAAAAwAKABI AAQAAAAAAAAABAABAAQAAAAAAABQALACY
AAQAAAAAAAAABgAKADEAAQAAAAAAACgAsAdSAAQAAAAAACWASAGcAAwABBakAAAACAhKAAwABBakAAQA
UAHsAAwABBakAAgAOAI8AAwABBakAAwAUAJ0AAwABBakABAAUALEAAwABBakABQAWAMUAAwABBak
ABgAUANsAAwABBakACgBYAO8AAwABBakACwAkAUcgZSlkyilpy29uc1JlZ3VsYXJlLWRiLWljbj25
zZSlkyilpy29uc1Zlcnpjb24gMS4wZSlkyilpy29uc0ZvbncgZ2VuZXJhdGVkIHVzaW5nIFN5bmN
mdXNpb24gTWV0cm8gU3RlZGlvd3d3LnN5bmNmZDlmc3RlZ3R1bnN5bmNmdXNpb24uY29tACAAZQAtAGQAYgAtAGkAYwBvAG4
AcwBSAGUAZwBlAGwAYQByAGUALQBkAGIALQBpAGMAbwBuAHMAZQAtAGQAYgAtAGkAYwBvAG4AcwB
WAGUAcgBzAGkAbwBuACAAMQAuADAAZQAtAGQAYgAtAGkAYwBvAG4AcwBGAG8AbgBOACAAZwBlAG4
AZQByAGEadABlAGQAIBlAHMAaQBwAGcAlABTAHkAbgBjAGYAdQBzAGkAbwBuACAATQBIAHQAcgB
vACAAUwBOAHUAZABpAG8AdwB3AHcALgBzAHkAbgBjAGYAdQBzAGkAbwBuAC4AYwBvAG0AAAAAAgA
AAAAAAAAAKAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAEAQIBAwEEAAQUADG1lc3NhZ2UtbfWfbpAtyZWFF
kLXVucmVhZAZkZWxlZGUAAAAAA==) format('truetype');
  font-weight: normal;
  font-style: normal;
}
.ddb-icons {
  font-family: 'e-db-icons' !important;
```

```

        speak: none;
        font-size: 55px;
        font-style: normal;
        font-weight: normal;
        font-variant: normal;
        text-transform: none;
        line-height: 1;
        -webkit-font-smoothing: antialiased;
        -moz-osx-font-smoothing: grayscale;
    }
    .e-message::before {
        content: '\e703';
    }
}
</style>

```

{% previewsample "page.domainurl/code-snippet/drop-down-button/default-cs7" %}

Group popup items with listview component in Vue Drop down button component

Header in popup items is possible in DropDownButton by templating entire popup with ListView. Create ListView with id `#listview` and provide it as a [target](#) for DropDownButton.

Install Syncfusion `List` packages using below command.

```
`bash
```

```
npm install @syncfusion/ej2-vue-list --save
```

```
,
```

In the following example, ListView element is given as `target` to DropDownButton and header can be achieved by [groupBy](#) property.

#### APP.VUE

```

<template>
<div>
    <ejs-dropdownbutton target= '#listview' iconCss= 'e-icons e-down'
    cssClass= 'e-caret-hide'></ejs-dropdownbutton>
    <ejs-listview id='listview' :dataSource='dataSource' :fields='fields'
    showCheckBox= 'true'
    ></ejs-listview>
</div>
</template>
<script>
import Vue from 'vue';
import { DropDownButtonPlugin } from "@syncfusion/ej2-vue-splitbuttons";
import { ListViewPlugin } from '@syncfusion/ej2-vue-lists';
import { enableRipple } from '@syncfusion/ej2-base';
enableRipple(true);
Vue.use(DropDownButtonPlugin);
Vue.use(ListViewPlugin);
export default {
    data () {
        return {
            dataSource :[
                { class: 'data', text: 'Print', id: 'data1', category:
'Customize Quick Access Toolbar' },

```

```

        { class: 'data', text: 'Save As', id: 'data2', category:
'Customize Quick Access Toolbar' },
        { class: 'data', text: 'Update Folder', id: 'data3',
category: 'Customize Quick Access Toolbar' },
        { class: 'data', text: 'Reply', id: 'data4', category:
'Customize Quick Access Toolbar' }
    ],
    fields: { text: 'text', groupBy: 'category' }
}
}
}
</script>
<style>
    @import '../node_modules/@syncfusion/ej2-base/styles/material.css';
    @import '../node_modules/@syncfusion/ej2-buttons/styles/material.css';
    @import '../node_modules/@syncfusion/ej2-popups/styles/material.css';
    @import '../node_modules/@syncfusion/ej2-
splitbuttons/styles/material.css';
    @import '../node_modules/@syncfusion/ej2-lists/styles/material.css';
    .e-down::before {
        content: '\e969';
    }
</style>

```

{% previewsample "page.domainurl/code-snippet/drop-down-button/default-cs8" %}

### Hide dropdown arrow in Vue Drop down button component

You can hide the dropdown arrow from the DropDownButton by adding class `e-caret-hide` to DropDownButton element using [cssClass](#) property.

### APP.VUE

```

<template>
<ejs-dropdownbutton :items='items' cssClass= 'e-caret-hide'>Clipboard</ejs-
dropdownbutton>
</template>
<script>
import Vue from 'vue';
import { DropDownButtonPlugin } from "@syncfusion/ej2-vue-splitbuttons";
import { enableRipple } from '@syncfusion/ej2-base';
enableRipple(true);
Vue.use(DropDownButtonPlugin);
export default {
    data () {
        return {
            items:[
                {
                    text: 'Cut'
                },
                {
                    text: 'Copy'
                },
                {
                    text: 'Paste'
                }
            ]
        };
    }
};

```

```

    }
  }
</script>
<style>
  @import '../node_modules/@syncfusion/ej2-base/styles/material.css';
  @import '../node_modules/@syncfusion/ej2-buttons/styles/material.css';
  @import '../node_modules/@syncfusion/ej2-popups/styles/material.css';
  @import '../node_modules/@syncfusion/ej2-splitbuttons/styles/material.css';
</style>

```

{% previewsample "page.domainurl/code-snippet/drop-down-button/default-cs9" %}

Open a dialog on popup item click in Vue Drop down button component

This section explains about how to open a dialog on DropdownButton popup item click. This can be achieved by handling dialog open in [select](#) event of the DropdownButton.

Install Syncfusion **List** packages using below command.

```
`bash
```

```
npm install @syncfusion/ej2-vue-list --save
```

In the following example, Dialog will open while selecting **Other Folder...** item.

#### APP.VUE

```

<template>
<div>
  <ejs-dropdownbutton :items='items' iconCss='ddb-icons e-
  folder' cssClass='e-vertical'
    iconPosition='Top' :select='onSelect'>Move</ejs-dropdownbutton>
  <ejs-dialog id="dialog" content="Move Items To 'Web Team'" header='Move
  Items' :buttons='buttons' width='250px' height='150px' :visible='false'
  :position='position'></ejs-dialog>
</div>
</template>
<script>
import Vue from 'vue';
import { DropDownButtonPlugin } from "@syncfusion/ej2-vue-splitbuttons";
import { DialogPlugin } from "@syncfusion/ej2-vue-popups";
import { enableRipple } from '@syncfusion/ej2-base';
enableRipple(true);
Vue.use(DropDownButtonPlugin);
Vue.use(DialogPlugin);
export default {
  data () {
    return {
      items:[
        {
          text: 'Archive'
        },
        {
          text: 'Inbox'
        },
      ],
    }
  },

```

```

        {
            text: 'HR Portal'
        },
        {
            separator: true
        },
        {
            text: 'Other Folder...'
        },
        {
            text: 'Copy to Folder'
        }
    ]],
    buttons: [{
        buttonModel: {
            isPrimary: true,
            content: 'OK',
            cssClass: 'e-flat',
        },
        click: function () {
            this.hide();
        }
    }],
    position: {X: 100, Y: 100}
};

},
methods: {
    onSelect: function(args) {
        if (args.item.text === 'Other Folder...') {
            document.getElementById("dialog").ej2_instances[0].show();
        }
    }
},
}
</script>
<style>
    @import '../node_modules/@syncfusion/ej2-base/styles/material.css';
    @import '../node_modules/@syncfusion/ej2-buttons/styles/material.css';
    @import '../node_modules/@syncfusion/ej2-popups/styles/material.css';
    @import '../node_modules/@syncfusion/ej2-splitbuttons/styles/material.css';
    @font-face {
        font-family: 'e-db-icons';
        src:
            url(data:application/x-font-ttf;charset=utf-8;base64,AAEAAAKAIAAAwAgTlMvMj0jSRoAAAEoAAAAVmNtYXNfudgAAABkAAAAADpnbHlmSrK
            TCAAAAdgAAAC4aGVhZBktK8cAAADQAAAAANmhoZWEHmQNtAAAArAAAACRobXR4D7gAAAAAYAAAAA
            QbG9jYQB4ADoAAAHMAAAACmlheHABEAAAYAAABCAAAACBuYW1lH00mDAAAAPAAAAJJCg9zdIwksr0
            AAATcAAAATQABAAADUv9qAFoEAAAA//4D6gABAAAAAAAAAAAAAAAAABAAAAAAAAQAAGc/PS18
            PPPUACwPoAAAAANfSc3wAAAAA19JzfAAAAAAD6gPqAAAAACAACAAAAAAAAAAAAEAAwAAgAAAAA
            AAgAAAAoACgAAAP8AAAAAAAAAAQPUAZAABQAAANoCvAAAAIwCegK8AAAB4AAxAQIAAAIABQMAAAA
            AAAAAAAAAAAAAAAAAAAUGZFZABA5wPnBQNS/2oAWgPqAJYAAAAABAAAAAAAAABAAAAAPoAAA
            D6AAAA+gAAAAAAIAAADAAAAFAADAAEAAAAUAAQAjgAAAAQABAABADnBf//AADnA//AAAAQA
            EAAAAQACAMAAAAAAAAAAHAA6AFwAAAAACAAAAAPqA2UABgAKAAA3IREjCQEjBRcBIQID6AL+Dv4
            NAQEY3QG4/I+IAsL+GAHonroBcwAAAAIAAAAAA8YD6gAFAAoAADchESMJASUHCQImA6AD/jL+MQE
            EywGWAZb+agICX/4+AcLXsv6cAWQBZAAAAEAAAAAA+oD6gALAAATCQEXCQE3CQEnCQECATP+zcI
            BMgEzwf7OATLB/s3+zsIBM/7NwgEyATPB/s4BMgAAAAASAN4AAQAAAAAAAAABAAAAQA
            AAAAAQAKAAEAQAQAAAAAAgAHAAsAAQAAAAAAAwAKABIAAQAAAAABAAKABwAAQAAAAABQALACY

```



```
AAQAAAAAABgAKADEEAQAQAAAAAACgAsADsAAQAAAAAACwASAGcAAwABBakAAAACAHkAAwABBakAAQA  
UAHsAAwABBakAAgAOAI8AAwABBakAAwAUAJ0AAwABBakABAAUAEAAwABBakABQAWAMUAAwABBak  
ABgAUANsAAwABBakACgBYAO8AAwABBakACwAkAUcgZS1kYi1pY29uc1JlZ3VsYXJlLWwiLWljbj25  
zZS1kYi1pY29uc1ZlcnNpb24gMS4wZS1kYi1pY29uc0Zvb2VudG9zaGVzaW5nIFN5bmNmdXNpb24gTWV0cm8gU3RlZGlvd3d3LnN5bmNmdXNpb24uY29tACAAZQAtAGQAYgAtAGKAYwBvAG4AcwBSAGUAZwBlAGwAYQBByAGUALQBkAGIALQBpAGMAbwBuAHMAZQAtAGQAYgAtAGKAYwBvAG4AcwBWAGUAcgBzAgKAbwBuACAAMQAUADAZQAtAGQAYgAtAGKAYwBvAG4AcwBGAG8AbgBOACAAZwBlAG4AZQByAGEAdABlAGQAIABlAHMAAQBUAGcAlABTAHkAbgBjAGYAdQBzAGKAbwBuACAATQBlAHQAcbGvACAAUwBOAHUAZABpAG8AdwB3AHcALGbZAHAkAbgBjAGYAdQBzAGKAbwBuAC4AYwBvAG0AAAAAAGAAAAAAAKAAAAAAAAAAAAAAAAAAAAAAAAAAAAEAEQIBAwEEAAQUADG1lc3NhZ2UtbnFpbAtyZWFiLmVucmVhZAZkZWxldGUAAAAAAAA==) format('trueType');  
  
font-weight: normal;  
font-style: normal;  
}  
.ddb-icons {  
font-family: 'e-db-icons' !important;  
speak: none;  
font-size: 55px;  
font-style: normal;  
font-weight: normal;  
font-variant: normal;  
text-transform: none;  
line-height: 1;  
-webkit-font-smoothing: antialiased;  
-moz-osx-font-smoothing: grayscale;  
}  
.e-folder::before {  
content: '\e703';  
}  

```

```
{% previewsample "page.domainurl/code-snippet/drop-down-button/default-cs10" %}
```

## Position popup open in Vue Drop down button component

Popup open position can be changed according to the requirement. Popup open position can be changed in `open` event by setting `top` and `left` for the popup element.

In the following example, the `top` position of the popup element is changed in `open` event.

## APP.VUE

```
<template>
<ejs-dropdownbutton id="ddBtn" :items='items' cssClass= 'e-caret-up'
:open='onOpen'>Clipboard</ejs-dropdownbutton>
</template>
<script>
import Vue from 'vue';
import { DropDownButtonPlugin } from "@syncfusion/ej2-vue-splitbuttons";
import { OpenCloseMenuEventArgs } from '@syncfusion/ej2-splitbuttons';
import { enableRipple } from '@syncfusion/ej2-base';
enableRipple(true);
Vue.use(DropDownButtonPlugin);
export default {
  data () {
    return {
      items:[
```

```

        {
            text: 'Cut'
        },
        {
            text: 'Copy'
        },
        {
            text: 'Paste'
        }
    ]
};
},
methods: {
    onOpen: function(args: OpenCloseMenuEventArgs) {
        args.element.parentElement.style.top =
document.getElementById('ddBtn').ej2_instances[0].element.getBoundingClientRect().top - args.element.parentElement.offsetHeight + 'px';
    }
}
}
</script>
<style>
    @import '../node_modules/@syncfusion/ej2-base/styles/material.css';
    @import '../node_modules/@syncfusion/ej2-buttons/styles/material.css';
    @import '../node_modules/@syncfusion/ej2-popups/styles/material.css';
    @import '../node_modules/@syncfusion/ej2-splitbuttons/styles/material.css';
    .e-dropdown-btn{
        margin: 25% 5px 25px 30%;
    }
</style>

```

{% previewsample "page.domainurl/code-snippet/drop-down-button/default-cs11" %}

Underline a character in the item text in Vue Drop down button component

Underline a particular character in a text can be handled in [beforeItemRender](#) event by adding `<u>` tag in between the text and given as innerHTML in `li` rendering.

In the following example, **C** is underlined in the text **Copy**.

#### APP.VUE

```

<template>
<ejs-dropdownbutton :items='items'
:beforeItemRender='onBeforeItemRender'>Clipboard</ejs-dropdownbutton>
</template>
<script>
import Vue from 'vue';
import { DropDownButtonPlugin } from "@syncfusion/ej2-vue-splitbuttons";
import { enableRipple } from '@syncfusion/ej2-base';
enableRipple(true);
Vue.use(DropDownButtonPlugin);
export default {
    data () {
        return {
            items:[
                {

```

```

        text: 'Cut'
      },
      {
        text: 'Copy'
      },
      {
        text: 'Paste'
      }
    ]],
  },
  methods: {
    onBeforeItemRender: function(args) {
      if (args.item.text === 'Copy') {
        //To underline a particular text.
        args.element.innerHTML = '<u>C</u>opy';
      }
    }
  }
}
</script>
<style>
  @import '../node_modules/@syncfusion/ej2-base/styles/material.css';
  @import '../node_modules/@syncfusion/ej2-buttons/styles/material.css';
  @import '../node_modules/@syncfusion/ej2-popups/styles/material.css';
  @import '../node_modules/@syncfusion/ej2-splitbuttons/styles/material.css';
</style>

```

{% previewsample "page.domainurl/code-snippet/drop-down-button/default-cs12" %}

## DropDownList

### Getting Started with the Vue Drop down list Component in Vue 2

This article provides a step-by-step guide for setting up a Vue 2 project using [Vue-CLI](#) and integrating the Syncfusion Vue Drop down list component using the [Composition API](#) / [Options API](#)[Link to the Video](#).

To get start quickly with DropDownList Component using Vue CLI, you can check on this video:

#### Prerequisites

[System requirements for Syncfusion Vue UI components](#)

#### Setting up the Vue 2 project

To generate a Vue 2 project using Vue-CLI, use the [vue create](#) command. Follow these steps to install Vue CLI and create a new project:

```

`bash
npm install -g @vue/cli
vue create quickstart
cd quickstart
npm run serve
`

```

or

```
`bash
```

```
yarn global add @vue/cli
```

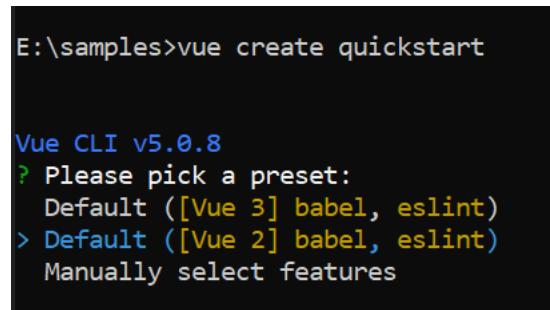
```
vue create quickstart
```

```
cd quickstart
```

```
yarn run serve
```

```
`
```

When creating a new project, choose the option **Default ([Vue 2] babel, eslint)** from the menu.



```
E:\samples>vue create quickstart

Vue CLI v5.0.8
? Please pick a preset:
  Default ([Vue 3] babel, eslint)
> Default ([Vue 2] babel, eslint)
  Manually select features
```

Once the **quickstart** project is set up with default settings, proceed to add Syncfusion components to the project.

#### Add Syncfusion Vue packages

Syncfusion packages are available at [npmjs.com](https://www.npmjs.com). To use Vue components, install the required npm package.

This article uses the [Vue Drop down list component](#) as an example. Install the **@syncfusion/ej2-vue-dropdowns** package by running the following command:

```
`bash
```

```
npm install @syncfusion/ej2-vue-dropdowns --save
```

```
`
```

or

```
`bash
```

```
yarn add @syncfusion/ej2-vue-dropdowns
```

```
`
```

#### Import Syncfusion CSS styles

You can import themes for the Syncfusion Vue component in various ways, such as using CSS or SASS styles from npm packages, CDN, [CRG](#) and [Theme Studio](#). Refer to [themes topic](#) to know more about built-in themes and different ways to refer to themes in a Vue project.

In this article, the **Material** theme is applied using CSS styles, which are available in installed packages. The necessary **Material** CSS styles for the Drop down list component and its dependents were imported into the `<style>` section of **src/App.vue** file.

### ~/SRC/APP.VUE

```
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-inputs/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-dropdowns/styles/material.css";
</style>
```

#### Add Syncfusion Vue component

Follow the below steps to add the Vue Drop down list component using **Composition API** or **Options API**:

1\ First, import and register the Drop down list component in the **script** section of the **src/App.vue** file. If you are using the **Composition API**, you should add the **setup** attribute to the **script** tag to indicate that Vue will be using the **Composition API**.

#### **COMPOSITION API (~/SRC/APP.VUE)**

```
<script setup>
import { DropDownListComponent as EjsDropdownlist } from "@syncfusion/ej2-vue-dropdowns";
</script>
```

#### **COMPOSITION API (~/SRC/APP.VUE)**

```
<script>
import { DropDownListComponent } from "@syncfusion/ej2-vue-dropdowns";
export default {
  components: {
    'ejs-dropdownlist': DropDownListComponent
  }
}
</script>
```

2\ In the **template** section, define the Drop down list component with the **dataSource** and **[placeholder]** (<https://ej2.syncfusion.com/vue/documentation/api/drop-down-list/#placeholder>) property.

### ~/SRC/APP.VUE

```
<template>
<div id="app">
<div id='container' style="margin:50px auto 0; width:250px;">
<br>
<ejs-dropdownlist id='dropdownlist' :dataSource='sportsData'
placeholder='Select a game'></ejs-dropdownlist>
</div>
</div>
</template>
```

#### Binding data source

After initialization, populate the DropDownList with data using the **dataSource** property. Here, an array of string values is passed to the DropDownList component.

**COMPOSITION API (~SRC/APP.VUE)**

```
<template>
<div id="app">
<div id='container' style="margin:50px auto 0; width:250px;">
<br>
<ejs-dropdownlist id='dropdownlist' :dataSource='sportsData'
placeholder='Select a game'></ejs-dropdownlist>
</div>
</div>
</template>
<script setup>
import { DropDownListComponent as EjsDropdownlist } from "@syncfusion/ej2-vue-dropdowns";
const sportsData = ['Badminton', 'Cricket', 'Football', 'Golf', 'Tennis'];
</script>
```

**OPTIONS API (~SRC/APP.VUE)**

```
<template>
<div id="app">
<div id='container' style="margin:50px auto 0; width:250px;">
<br>
<ejs-dropdownlist id='dropdownlist' :dataSource='sportsData'
placeholder='Select a game'></ejs-dropdownlist>
</div>
</div>
</template>
<script>
import { DropDownListComponent } from "@syncfusion/ej2-vue-dropdowns";
export default {
  components: {
    'ejs-dropdownlist': DropDownListComponent
  },
  data () {
    return {
      sportsData: ['Badminton', 'Cricket', 'Football', 'Golf', 'Tennis']
    }
  }
}
</script>
```

Here is the summarized code for the above steps in the **src/App.vue** file:

**COMPOSITION API (~SRC/APP.VUE)**

```
<template>
  <div id="app">
    <div id='container' style="margin:50px auto 0; width:250px;">
      <br>
      <ejs-dropdownlist id='dropdownlist' :dataSource='sportsData'
placeholder='Select a game'></ejs-dropdownlist>
    </div>
  </div>
</template>
<script setup>
```

```
import { DropDownListComponent as EjsDropdownlist } from "@syncfusion/ej2-vue-dropdowns";
const sportsData = ['Badminton', 'Cricket', 'Football', 'Golf', 'Tennis'];
</script>
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-inputs/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-dropdowns/styles/material.css";
</style>
```

### OPTIONS API (~SRC/APP.VUE)

```
<template>
  <div id="app">
    <div id='container' style="margin:50px auto 0; width:250px;">
      <br>
      <ejs-dropdownlist id='dropdownlist' :dataSource='sportsData'
placeholder='Select a game'></ejs-dropdownlist>
    </div>
  </div>
</template>
<script>
import { DropDownListComponent } from "@syncfusion/ej2-vue-dropdowns";
export default {
  components: {
    'ejs-dropdownlist': DropDownListComponent
  },
  data () {
    return {
      sportsData: ['Badminton', 'Cricket', 'Football', 'Golf', 'Tennis']
    }
  }
}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-inputs/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-dropdowns/styles/material.css";
</style>
```

### Run the project

To run the project, use the following command:

```
`bash
```

```
npm run serve
```

```
,
```

```
or
```

```
`bash
```

```
yarn run serve
```

```
,
```

```
{% previewsample "page.domainurl/code-snippet/drop-down-list/getting-started/getting-started-cs1"
%}
```

### Configure the Popup List

By default, the width of the popup list automatically adjusts according to the DropDownList input element's width, and the height of the popup list has '300px'.

The height and width of the popup list can also be customized using the [popupHeight](<https://ej2.syncfusion.com/vue/documentation/api/drop-down-list/#popupheight>) and [popupWidth] properties respectively

### COMPOSITION API (~SRC/APP.VUE)

```
<template>
  <div id="app">
    <div id='container' style="margin:50px auto 0; width:250px;">
      <br>
      <ejs-dropdownlist id='dropdownlist' popupHeight="200px"
popupWidth="250px" :dataSource='sportsData' placeholder='Select a
game'></ejs-dropdownlist>
    </div>
  </div>
</template>
<script setup>
import { DropDownListComponent as EjsDropDownlist } from "@syncfusion/ej2-
vue-dropdowns";
const sportsData = ['Badminton', 'Cricket', 'Football', 'Golf', 'Tennis'];
</script>
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-inputs/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-dropdowns/styles/material.css";
</style>
```

### OPTIONS API (~SRC/APP.VUE)

```
<template>
  <div id="app">
    <div id='container' style="margin:50px auto 0; width:250px;">
      <br>
      <ejs-dropdownlist id='dropdownlist' popupHeight="200px"
popupWidth="250px" :dataSource='sportsData' placeholder='Select a
game'></ejs-dropdownlist>
    </div>
  </div>
</template>
<script>
import { DropDownListComponent } from "@syncfusion/ej2-vue-dropdowns";
export default {
  components: {
    'ejs-dropdownlist': DropDownListComponent
  },
  data () {
    return {
      sportsData: ['Badminton', 'Cricket', 'Football', 'Golf', 'Tennis'];
    }
  }
}
```



```
}  
}  
</script>  
<style>  
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";  
@import "../node_modules/@syncfusion/ej2-inputs/styles/material.css";  
@import "../node_modules/@syncfusion/ej2-vue-dropdowns/styles/material.css";  
</style>
```

{% previewsample "page.domainurl/code-snippet/drop-down-list/getting-started/popup-cs1" %}

You can refer to our [Vue Dropdown List](#) feature tour page for its groundbreaking feature representations. You can also explore our [Vue Dropdown list example](#) that shows how to render the Dropdown List in Vue.

See Also

- [How to bind the data](#)

## Getting Started with the Vue DropDownList Component in Vue 3

This article provides a step-by-step guide for setting up a [Vite](#) project with a JavaScript environment and integrating the Syncfusion Vue DropDownList component using the [Composition API](#) / [Options API](#).

The **Composition API** is a new feature introduced in Vue.js 3 that provides an alternative way to organize and reuse component logic. It allows developers to write components as functions that use smaller, reusable functions called composition functions to manage their properties and behavior.

The **Options API** is the traditional way of writing Vue.js components, where the component logic is organized into a series of options that define the component's properties and behavior. These options include data, methods, computed properties, watchers, lifecycle hooks, and more.

Prerequisites

[System requirements for Syncfusion Vue UI components](#)

Set up the Vite project

A recommended approach for beginning with Vue is to scaffold a project using [Vite](#). To create a new Vite project, use one of the commands that are specific to either NPM or Yarn.

```
`bash
```

```
npm create vite@latest
```

```
`
```

or

```
`bash
```

```
yarn create vite
```

```
`
```

Using one of the above commands will lead you to set up additional configurations for the project as below:

1. Define the project name: We can specify the name of the project directly. Let's specify the name of the project as `my-project` for this article.

```
`bash
```

```
? Project name: » my-project
```

```
,
```

2. Select `Vue` as the framework. It will create a Vue 3 project.

```
`bash
```

```
? Select a framework: » - Use arrow-keys. Return to submit.
```

```
Vanilla
```

```
Vue
```

```
React
```

```
Preact
```

```
Lit
```

```
Svelte
```

```
Others
```

```
,
```

3. Choose `JavaScript` as the framework variant to build this Vite project using JavaScript and Vue.

```
`bash
```

```
? Select a variant: » - Use arrow-keys. Return to submit.
```

```
JavaScript
```

```
TypeScript
```

```
Customize with create-vue ↗
```

```
Nuxt ↗
```

```
,
```

4. Upon completing the aforementioned steps to create the `my-project`, run the following command to install its dependencies:

```
`bash
```

```
cd my-project
```

```
npm install
```

```
,
```

```
or
```

```
`bash
```

```
cd my-project
```

yarn install

,

Now that `my-project` is ready to run with default settings, let's add Syncfusion components to the project.

Add Syncfusion Vue packages

Syncfusion Vue component packages are available at [npmjs.com](https://www.syncfusion.com). To use Syncfusion Vue components in the project, install the corresponding npm package.

This article uses the [Vue DropDownList component](#) as an example. To use the Vue DropDownList component in the project, the `@syncfusion/ej2-vue-dropdowns` package needs to be installed using the following command:

```
`bash
```

```
npm install @syncfusion/ej2-vue-dropdowns --save
```

,

or

```
`bash
```

```
yarn add @syncfusion/ej2-vue-dropdowns
```

,

### Import Syncfusion CSS styles

You can import themes for the Syncfusion Vue component in various ways, such as using CSS or SASS styles from npm packages, CDN, [CRG](#) and [Theme Studio](#). Refer to [themes topic](#) to know more about built-in themes and different ways to refer to themes in a Vue project.

In this article, `Material` theme is applied using CSS styles, which are available in installed packages. The necessary `Material` CSS styles for the DropDownList component and its dependents were imported into the `<style>` section of `src/App.vue` file.

#### ~/SRC/APP.VUE

```
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-inputs/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-dropdowns/styles/material.css";
</style>
```

The order of importing CSS styles should be in line with its dependency graph.

### Add Syncfusion Vue component

Follow the below steps to add the Vue DropDownList component using `Composition API` or `Options API`:

1.First, import and register the DropDownList component in the `script` section of the `src/App.vue` file. If you are using the `Composition API`, you should add the `setup` attribute to the `script` tag to indicate that Vue will be using the `Composition API`.

**COMPOSITION API (~SRC/APP.VUE)**

```
<script setup>
import { DropDownListComponent as EjsDropdownlist } from "@syncfusion/ej2-vue-dropdowns";
</script>
```

**OPTIONS API (~SRC/APP.VUE)**

```
<script>
import { DropDownListComponent } from "@syncfusion/ej2-vue-dropdowns";
//Component registration
export default {
  name: "App",
  components: {
    'ejs-dropdownlist' : DropDownListComponent,
  }
}
</script>
```

2. In the **template** section, define the DropDownList component with the [dataSource](#) property and column definitions.

**~/SRC/APP.VUE**

```
<template>
<div class="control_wrapper">
<ejs-dropdownlist id='dropdownlist' :dataSource='sportsData'></ejs-
dropdownlist>
</div>
</template>
```

Here is the summarized code for the above steps in the **src/App.vue** file:

**COMPOSITION API (~SRC/APP.VUE)**

```
<template>
<div class="control_wrapper">
<ejs-dropdownlist id='dropdownlist' :dataSource='data[0].sportsData'></ejs-
dropdownlist>
</div>
</template>
<script setup>
import { DropDownListComponent as EjsDropdownlist } from "@syncfusion/ej2-vue-dropdowns";
const data = [{ sportsData : ['Badminton', 'Cricket', 'Football', 'Golf', 'Tennis'] }]
</script>
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-inputs/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-dropdowns/styles/material.css";
</style>
```

**OPTIONS API (~SRC/APP.VUE)**

```
<template>
<div class="control_wrapper">
<ejs-dropdownlist id='dropdownlist' :dataSource='sportsData'></ejs-
dropdownlist>
</div>
</template>
<script>
import { DropDownListComponent } from "@syncfusion/ej2-vue-dropdowns";
//Component registration
export default {
name: 'App',
components: {
"ejs-dropdownlist": DropDownListComponent
},
data () {
return {
sportsData: ['Badminton', 'Cricket', 'Football', 'Golf', 'Tennis']
}
}
}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-inputs/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-dropdowns/styles/material.css";
</style>
```

### Run the project

To run the project, use the following command:

```
`bash
```

```
npm run dev
```

```
,
```

```
or
```

```
`bash
```

```
yarn run dev
```

```
,
```

### Configure the Popup List

By default, the width of the popup list automatically adjusts according to the DropDownList input element's width, and the height of the popup list has '300px'.

The height and width of the popup list can also be customized using the [popupHeight](<https://ej2.syncfusion.com/vue/documentation/api/drop-down-list/#popupheight>) and [popupWidth] properties respectively

### COMPOSITION API (~SRC/APP.VUE)

```
<template>
<div id="app">
<div id='container' style="margin:50px auto 0; width:250px;">
<br>
```

```

<ejs-dropdownlist id='dropdownlist' popupHeight="200px" popupWidth="250px"
:dataSource='data[0].sportsData' placeholder='Select a game'></ejs-
dropdownlist>
</div>
</div>
</template>
<script setup>
import { DropDownListComponent as EjsDropDownlist } from "@syncfusion/ej2-
vue-dropdowns";
const data = [{ sportsData : ['Badminton', 'Cricket', 'Football', 'Golf',
'Tennis'] }]
</script>
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-inputs/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-dropdowns/styles/material.css";
</style>

```

### OPTIONS API (~SRC/APP.VUE)

```

<template>
<div id="app">
<div id='container' style="margin:50px auto 0; width:250px;">
<br>
<ejs-dropdownlist id='dropdownlist' popupHeight="200px" popupWidth="250px"
:dataSource='sportsData' placeholder='Select a game'></ejs-dropdownlist>
</div>
</div>
</template>
<script>
import { DropDownListComponent } from "@syncfusion/ej2-vue-dropdowns";
//Component registration
export default {
name: 'App',
components: {
"ejs-dropdownlist": DropDownListComponent
},
data () {
return {
sportsData: ['Badminton', 'Cricket', 'Football', 'Golf', 'Tennis'],
}
}
}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-inputs/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-dropdowns/styles/material.css";
</style>

```

The output will appear as follows:



See Also

- [How to bind the data](#)

### Data binding in Vue Drop down list component

The DropDownList loads the data either from local data sources or remote data services using the [dataSource](#) property. It supports the data type of array or **DataManager**.

The DropDownList also supports different kinds of data services such as OData, OData V4, and Web API, and data formats such as XML, JSON, and JSONP with the help of **DataManager** adaptors.

| Fields | Type | Description |

|-----|-----|-----|

| text | **string** | Specifies the display text of each list item. |

| value | **number or string** | Specifies the hidden data value mapped to each list item that should contain a unique value. |

| groupBy | **string** | Specifies the category under which the list item has to be grouped. |

| iconCss | **string** | Specifies the icon class of each list item. |

When binding complex data to the DropDownList, fields should be mapped correctly. Otherwise, the selected item remains undefined.

### Binding local data

Local data can be represented in two ways as described below.

#### 1. Array of simple data

The DropDownList has support to load array of primitive data such as strings and numbers. Here, both value and text field act the same.

### APP.VUE

```
<template>
  <div id="app">
    <div id='container' style="margin:50px auto 0; width:250px;">
      <br>
      <ejs-dropdownlist id='dropdownlist' :dataSource='sportsData'
placeholder='Select a game'></ejs-dropdownlist>
    </div>
  </div>
</template>
<script>
import Vue from 'vue';
```

```

import { DropDownListPlugin } from "@syncfusion/ej2-vue-dropdowns";
Vue.use(DropDownListPlugin);
export default {
  data () {
    return {
      sportsData: ['Badminton', 'Cricket', 'Football', 'Golf', 'Tennis'];
    }
  }
}
</script>
<style>
@import
"https://ej2.syncfusion.com/vue/documentation/../../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../../node_modules/@syncfusion/ej2-inputs/styles/material.css";
@import "../../node_modules/@syncfusion/ej2-vue-dropdowns/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/drop-down-list/data-binding/simple-cs1" %}

## 2. Array of JSON data

The DropDownList can generate its list items through an array of complex data. For this, the appropriate columns should be mapped to the [fields](#) property.

In the following example, **Id** column and **Game** column from complex data have been mapped to the **value** field and **text** field, respectively.

### APP.VUE

```

<template>
  <div id="app">
    <div id='container' style="margin:50px auto 0; width:250px;">
      <br>
      <ejs-dropdownlist id='dropdownlist' placeholder='Select a game'
      :dataSource='sportsData' :fields='fields'></ejs-dropdownlist>
    </div>
  </div>
</template>
<script>
import Vue from 'vue';
import { DropDownListPlugin } from "@syncfusion/ej2-vue-dropdowns";
Vue.use(DropDownListPlugin);
export default {
  data () {
    return {
      sportsData: [
        { Id: 'game1', Game: 'Badminton' },
        { Id: 'game2', Game: 'Football' },
        { Id: 'game3', Game: 'Tennis' }
      ],
      fields: { text: 'Game', value: 'Id' },
    }
  }
}
</script>

```



```
<style>
@import
"https://ej2.syncfusion.com/vue/documentation/../../node_modules/@syncfusion/ej2-base/styles/material.css";
@import ".../node_modules/@syncfusion/ej2-inputs/styles/material.css";
@import ".../node_modules/@syncfusion/ej2-vue-
dropdowns/styles/material.css";
</style>
```

```
{% previewsample "page.domainurl/code-snippet/drop-down-list/data-binding/json-cs1" %}
```

### 3. Array of Complex data

The DropDownList can generate its list items through an array of complex data. For this, the appropriate columns should be mapped to the [fields](#) property.

In the following example, **Code.Id** column and **Country.Name** column from complex data have been mapped to the **value** field and **text** field, respectively.

#### APP.VUE

```
<template>
  <div id="app">
    <div id='container' style="margin:50px auto 0; width:250px;">
      <br>
      <ejs-dropdownlist id='dropdownlist' placeholder='Select a country'
      :dataSource='countriesData' :fields='fields'></ejs-dropdownlist>
    </div>
  </div>
</template>
<script>
import Vue from 'vue';
import { DropDownListPlugin } from "@syncfusion/ej2-vue-dropdowns";
Vue.use(DropDownListPlugin);
export default {
  data () {
    return {
      countriesData : [
        { Country: { Name: 'Australia' }, Code: { Id: 'AU' } },
        { Country: { Name: 'Bermuda' }, Code: { Id: 'BM' } },
        { Country: { Name: 'Canada' }, Code: { Id: 'CA' } },
        { Country: { Name: 'Cameroon' }, Code: { Id: 'CM' } },
        { Country: { Name: 'Denmark' }, Code: { Id: 'DK' } },
        { Country: { Name: 'France' }, Code: { Id: 'FR' } }
      ],
      fields: { text: 'Country.Name', value: 'Code.Id' }
    }
  }
}
</script>
<style>
@import
"https://ej2.syncfusion.com/vue/documentation/../../node_modules/@syncfusion/ej2-base/styles/material.css";
@import ".../node_modules/@syncfusion/ej2-inputs/styles/material.css";
@import ".../node_modules/@syncfusion/ej2-vue-
dropdowns/styles/material.css";
```

```
</style>
```

```
{% previewsample "page.domainurl/code-snippet/drop-down-list/data-binding/complex-cs1" %}
```

### Binding remote data

The DropDownList supports retrieval of data from remote data services with the help of **DataManager** component. The [Query](#) property is used to fetch data from the database and bind it to the DropDownList.

The following sample displays the first 6 contacts from “Customers” table of the **Northwind** Data Service.

### APP.VUE

```
<template>
  <div id="app">
    <div id='container' style="margin:50px auto 0; width:250px;">
      <br>
      <ejs-dropdownlist id='dropdownlist' placeholder='Select a customer'
sortOrder='Ascending' :dataSource='dataSource' :query='query'
:fields='fields'></ejs-dropdownlist>
    </div>
  </div>
</template>
<script>
import Vue from 'vue';
import { DropDownListPlugin } from "@syncfusion/ej2-vue-dropdowns";
Vue.use(DropDownListPlugin);
import { DataManager, Query, ODataV4Adaptor } from "@syncfusion/ej2-data";
export default {
  data () {
    return {
      query : new Query().from('Customers').select(['ContactName',
'CustomerID']).take(6),
      dataSource : new DataManager({
        url: 'https://services.odata.org/V4/Northwind/Northwind.svc/',
        adaptor: new ODataV4Adaptor,
        crossDomain: true
      }),
      fields: { text: 'ContactName', value: 'CustomerID' }
    }
  }
}
</script>
<style>
@import "../../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../../node_modules/@syncfusion/ej2-inputs/styles/material.css";
@import "../../node_modules/@syncfusion/ej2-vue-
dropdowns/styles/material.css";
</style>
```

```
{% previewsample "page.domainurl/code-snippet/drop-down-list/data-binding/remote-cs1" %}
```

## See Also

- [How to load data using template](#)
- [How to group the data using header](#)
- [How to filter the bound data](#)
- [How to get the count of the data when using remote data](#)
- [How to achieve cascading](#)
- [How to add item in between the options](#)
- [How to remove an item](#)
- [How to preselect the items in dropdownlist](#)

## Templates in Vue Drop down list component

The DropDownList has been provided with several options to customize each list item, group title, selected value, header, and footer elements. It uses the Essential JS 2 **Template engine** to compile and render the elements properly.

## Item template

The content of each list item within the DropDownList can be customized with the help of [itemTemplate](#) property.

In the following sample, each list item is split into two columns to display relevant data's.

**APP.VUE**

```
<template>
  <div id="app">
    <div id='container' style="margin:50px auto 0; width:250px;">
      <br>
      <ejs-dropdownlist id='dropdownlist' placeholder='Select an employee'
sortOrder='Ascending' :itemTemplate='itemTemplate' :dataSource='dataSource'
:query='query' :fields='fields'></ejs-dropdownlist>
    </div>
  </div>
</template>
<script>
import Vue from 'vue';
import { DropDownListPlugin } from "@syncfusion/ej2-vue-dropdowns";
Vue.use(DropDownListPlugin);
import { DataManager, Query, ODataV4Adaptor } from "@syncfusion/ej2-data";
var itemVue = Vue.component("itemTemplate", {
  template: `<span><span class='name'>{{data.FirstName}}</span><span class
='city'>{{data.City}}</span></span>`,
  data() {
    return {
      data: {}
    };
  }
});
export default {
  data () {
    return {
      itemTemplate : function() {
        return {template: itemVue};
      },

```

```

        query : new Query().from('Employees').select(['FirstName', 'City',
'EmployeeID']).take(6),
        dataSource : new DataManager({
            url: 'https://services.odata.org/V4/Northwind/Northwind.svc/',
            adaptor: new ODataV4Adaptor,
            crossDomain: true
        }),
        fields: { text: 'FirstName', value: 'EmployeeID' }
    }
}
</script>
<style>
@import ".../node_modules/@syncfusion/ej2-base/styles/material.css";
@import ".../node_modules/@syncfusion/ej2-inputs/styles/material.css";
@import ".../node_modules/@syncfusion/ej2-vue-
dropdowns/styles/material.css";
.city {
    right: 15px;
    position: absolute;
}
</style>

```

{% previewsample "page.domainurl/code-snippet/drop-down-list/templates/item-template-cs1" %}

### Value template

The currently selected value that is displayed by default on the DropDownList input element can be customized using the [valueTemplate](#) property.

In the following sample, the selected value is displayed as a combined text of both **FirstName** and **City** in the DropDownList input, which is separated by a hyphen.

### APP.VUE

```

<template>
  <div id="app">
    <div id='container' style="margin:50px auto 0; width:250px;">
      <br>
      <ejs-dropdownlist id='dropdownlist' placeholder='Select an employee'
sortOrder='Ascending' :itemTemplate='itemTemplate'
:valueTemplate='valueTemplate' :dataSource='dataSource' :query='query'
:fields='fields'></ejs-dropdownlist>
    </div>
  </div>
</template>
<script>
import Vue from 'vue';
import { DropDownListPlugin } from "@syncfusion/ej2-vue-dropdowns";
Vue.use(DropDownListPlugin);
import { DataManager, Query, ODataV4Adaptor } from "@syncfusion/ej2-data";
var itemVue = Vue.component("itemTemplate", {
  template: `<span><span class='name'>{{data.FirstName}}</span><span class
='city'>{{data.City}}</span></span>`,
  data() {
    return {
      data: {}
    }
  }
});

```

```

    };
  }
});
var valueVue = Vue.component("valueTemplate", {
  template: `<span>{{data.FirstName}} - {{data.City}}</span>`,
  data() {
    return {
      data: {}
    };
  }
});
export default {
  data () {
    return {
      itemTemplate : function() {
        return {template: itemVue};
      },
      valueTemplate: function() {
        return {template: valueVue};
      },
      query : new Query().from('Employees').select(['FirstName', 'City',
'EmployeeID']).take(6),
      dataSource : new DataManager({
        url: 'https://services.odata.org/V4/Northwind/Northwind.svc/',
        adaptor: new ODataV4Adaptor,
        crossDomain: true
      }),
      fields: { text: 'FirstName', value: 'EmployeeID' }
    }
  }
}
</script>
<style>
@import "../../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../../node_modules/@syncfusion/ej2-inputs/styles/material.css";
@import "../../node_modules/@syncfusion/ej2-vue-
dropdowns/styles/material.css";
.city {
  right: 15px;
  position: absolute;
}
</style>

```

{% previewsample "page.domainurl/code-snippet/drop-down-list/templates/value-template-cs1" %}

### Group template

The group header title under which appropriate sub-items are categorized can also be customized with the help of [groupTemplate](#) property. This template is common for both inline and floating group header template.

In the following sample, employees are grouped according to their city.

### APP.VUE

```

<template>
  <div id="app">

```

```

<div id='container' style="margin:5px auto; width:250px;">
  <br>
  <ejs-dropdownlist id='dropdownlist' placeholder='Select an employee'
sortOrder='Ascending' :groupTemplate="groupTemplate"
:dataSource='dataSource' :query='query' :fields='fields'></ejs-dropdownlist>
</div>
</div>
</template>
<script>
import Vue from 'vue';
import { DropDownListPlugin } from "@syncfusion/ej2-vue-dropdowns";
Vue.use(DropDownListPlugin);
import { DataManager, Query, ODataV4Adaptor, Predicate } from "@syncfusion/ej2-
data";
var groupVue = Vue.component("groupTemplate", {
  template: `<strong>{{data.City}}</strong>`,
  data() {
    return {
      data: {}
    };
  }
});
export default {
  data () {
    return {
      groupTemplate : function (e) {
        return {
          template: groupVue
        }
      },
      query : new Query().from('Employees').select(['FirstName', 'City',
'EmployeeID']).take(6).where(new Predicate('City', 'equal',
'London').or('City', 'equal', 'Seattle')),
      dataSource : new DataManager({
        url: 'https://services.odata.org/V4/Northwind/Northwind.svc/',
        adaptor: new ODataV4Adaptor,
        crossDomain: true
      }),
      fields: { text: 'FirstName', value: 'EmployeeID', groupBy: 'City' }
    }
  }
}
</script>
<style>
@import "../../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../../node_modules/@syncfusion/ej2-inputs/styles/material.css";
@import "../../node_modules/@syncfusion/ej2-vue-
dropdowns/styles/material.css";
.city {
  right: 15px;
  position: absolute;
}
</style>

```

{% previewsample "page.domainurl/code-snippet/drop-down-list/templates/group-template-cs1" %}

### Header template

The header element is shown statically at the top of the popup list items within the DropDownList, and any custom element can be placed as a header element using the [headerTemplate](#) property.

In the following sample, the list items and its headers are designed and displayed as two columns similar to multiple columns of the grid.

### APP.VUE

```
<template>
  <div id="app">
    <div class='combobox'>
      <br>
      <ejs-combobox id='combobox' sortOrder="Ascending"
:dataSource='employeeData' :itemTemplate='itemTemplate'
:headerTemplate='headerTemplate' :fields='fields' :query='query'
placeholder="Select an employee"></ejs-combobox>
    </div>
  </div>
</template>
<script>
import Vue from 'vue';
import { ComboBoxPlugin } from "@syncfusion/ej2-vue-dropdowns";
Vue.use(ComboBoxPlugin);
import { Query, DataManager, Predicate, ODataV4Adaptor } from
'@syncfusion/ej2-data';
var headerVue = Vue.component("headerTemplate", {
  template: `<span class='head'><span class='name'>Name</span><span
class='city'>City</span></span>`,
  data() {
    return {
      data: {}
    };
  }
});
var itemVue = Vue.component("itemTemplate", {
  template: `<span class='item' ><span
class='name'>{{data.FirstName}}</span><span
class='city'>{{data.City}}</span></span>`,
  data() {
    return {
      data: {}
    };
  }
});
export default {
  data () {
    return {
      employeeData : new DataManager({
        url: 'https://services.odata.org/V4/Northwind/Northwind.svc/',
        adaptor: new ODataV4Adaptor,
        crossDomain: true
      }),
      query : new Query().from('Employees').select(['FirstName', 'City',
'EmployeeID']).take(6),
      fields : { text: 'FirstName', value: 'EmployeeID' },
      headerTemplate : function(e) {
```

```

        return {
            template: headerVue
        };
    },
    itemTemplate : function(e) {
        return {
            template: itemVue
        };
    }
}
}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-inputs/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-
dropdowns/styles/material.css";
.head,
.item {
    display: table;
    width: 100%;
    margin: auto;
}
.head {
    height: 40px;
    font-size: 15px;
    font-weight: 600;
}
.name,
.city {
    display: table-cell;
    vertical-align: middle;
    width: 50%;
}
.head .name {
    text-indent: 16px;
}
.head .city {
    text-indent: 10px;
}
#app {
    color: #008cff;
    height: 40px;
    position: absolute;
    width: 90%;
    top: 10%;
}
.combobox {
    width: 30%;
    margin: 0 auto;
}
</style>

```

{% previewsample "page.domainurl/code-snippet/drop-down-list/templates/header-template-cs1" %}



### Footer template

The DropDownList has options to show a footer element at the bottom of the list items in the popup list. Here, you can place any custom element as a footer element using the [footerTemplate](#) property.

In the following sample, footer element displays the total number of list items present in the DropDownList.

### APP.VUE

```
<template>
  <div id="app">
    <div id='container' style="margin:50px auto 0; width:250px;">
      <br>
      <ejs-dropdownlist id='dropdownlist' placeholder='Select a game'
:footerTemplate='footerTemplate' :dataSource='dataSource'></ejs-
dropdownlist>
    </div>
  </div>
</template>
<script>
import Vue from 'vue';
import { DropDownListPlugin } from "@syncfusion/ej2-vue-dropdowns";
Vue.use(DropDownListPlugin);
var itemVue = Vue.component("itemTemplate", {
  template: `<span class='foot'> Total list item: 5</span>`,
  data() {
    return {
      data: {}
    };
  }
});
export default {
  data () {
    var sportsData = ['Badminton', 'Cricket', 'Football', 'Golf',
'Tennis'];
    return {
      footerTemplate : function(e) {
        return {
          template: itemVue
        }
      },
      dataSource : sportsData,
    }
  }
}
</script>
<style>
@import "../../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../../node_modules/@syncfusion/ej2-inputs/styles/material.css";
@import "../../node_modules/@syncfusion/ej2-vue-
dropdowns/styles/material.css";
.foot {
  text-indent: 1.2em;
  display: block;
  font-size: 15px;
  line-height: 40px;
  border-top: 1px solid #e0e0e0;
}
```

```
}
</style>
```

{% previewsample "page.domainurl/code-snippet/drop-down-list/templates/footer-template-cs1" %}

### No records template

The DropDownList is provided with support to custom design the popup list content when no data is found and no matches found on search with the help of [noRecordsTemplate](#) property.

In the following sample, popup list content displays the notification of no data available.

### APP.VUE

```
<template>
  <div id="app">
    <div id='container' style="margin:50px auto 0; width:250px;">
      <br>
      <ejs-dropdownlist id='dropdownlist' placeholder='Select an item'
:noRecordsTemplate='noRecordsTemplate' :dataSource='dataSource'></ejs-
dropdownlist>
    </div>
  </div>
</template>
<script>
import Vue from 'vue';
import { DropDownListPlugin } from "@syncfusion/ej2-vue-dropdowns";
Vue.use(DropDownListPlugin);
var itemVue = Vue.component("noRecordsTemplate", {
  template: `<span class='norecord'> NO DATA AVAILABLE</span>`,
  data() {
    return {
      data: {}
    };
  }
});
export default {
  data () {
    var sportsData = [];
    return {
      noRecordsTemplate : function (e) {
        return {
          template: itemVue
        }
      },
      dataSource : sportsData
    }
  }
}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-inputs/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-
dropdowns/styles/material.css";
</style>
```

```
{% previewsample "page.domainurl/code-snippet/drop-down-list/templates/no-template-cs1" %}
```

### Action failure template

There is also an option to custom design the popup list content when the data fetch request fails at the remote server. This can be achieved using the [actionFailureTemplate](#) property.

In the following sample, when the data fetch request fails, the DropDownList displays the notification.

### APP.VUE

```
<template>
  <div id="app">
    <div id='container' style="margin:50px auto 0; width:250px;">
      <br>
      <ejs-dropdownlist id='dropdownlist' placeholder='Select an customer'
sortOrder='Ascending' :actionFailureTemplate='failureTemplate'
:dataSource='dataSource' :query='query' :fields='fields'></ejs-dropdownlist>
    </div>
  </div>
</template>
<script>
import Vue from 'vue';
import { DropDownListPlugin } from "@syncfusion/ej2-vue-dropdowns";
Vue.use(DropDownListPlugin);
import { DataManager, Query, ODataV4Adaptor } from "@syncfusion/ej2-data";
var itemVue = Vue.component("failureTemplate", {
  template: `<span class='action-failure'> Data fetch get fails</span>`,
  data() {
    return {
      data: {}
    };
  }
});
export default {
  data () {
    return {
      failureTemplate : function(e) {
        return {
          template: itemVue
        }
      },
      query : new Query().from('Customers').select(['ContactName',
'CustomerID']).take(6),
      dataSource : new DataManager({
        url: 'https://services.odata.org/V4/Northwind/Northwind.svc/',
        adaptor: new ODataV4Adaptor,
        crossDomain: true
      }),
      fields: { text: 'ContactName', value: 'CustomerID' }
    }
  }
}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-inputs/styles/material.css";
```

```
@import "../../node_modules/@syncfusion/ej2-vue-  
dropdowns/styles/material.css";  
</style>
```

{% previewsample "page.domainurl/code-snippet/drop-down-list/templates/failure-template-cs1" %}

See Also

- [How to achieve filtering](#)
- [How to group the data using header](#)
- [How to show the list items with icon](#)
- [How to render tooltip for the options](#)

## Virtualization in DropDown List

Dropdown list virtualization is a technique used to efficiently render extensive lists of items while minimizing the impact on performance. This method is particularly advantageous when dealing with large datasets because it ensures that only a fixed number of DOM (Document Object Model) elements are created. When scrolling through the list, existing DOM elements are reused to display relevant data instead of generating new elements for each item. This recycling process is managed internally.

During virtual scrolling, the data retrieved from the data source depends on the popup height and the calculation of the list item height. Enabling the [enableVirtualization](#) option in a dropdown list activates this virtualization technique.

When fetching data from the data source, the [actionBegin](#) event is triggered before data retrieval begins. Then, the [actionComplete](#) event is triggered once the data is successfully fetched.

Furthermore, Incremental Search is supported with virtualization in the DropDownList component. When a key is typed, the focus is moved to the respective element, and the value is updated in the component in the open popup state. In the closed popup state, the respective value is updated in the component based on the typed key. The Incremental Search functionality is well-suited for scenarios involving remote data binding.

When the enableVirtualization property is enabled, the `skip` and `take` properties provided by the user within the Query class at the initial state or during the `actionBegin` or `actionComplete` events will not be considered, since it is internally managed and calculated based on certain dimensions with respect to the popup height.

## Binding local data

The DropDownList can generate its list items through an array of complex data. For this, the appropriate columns should be mapped to the [fields](#) property. When using virtual scrolling, the list updates based on the scroll offset value, triggering a request to fetch more data from the server. As the data is being fetched, the `actionBegin` event occurs before the data retrieval starts. Once the data retrieval is successful, the `actionComplete` event is triggered, indicating that the data fetch process is complete.

In the following example, `id` column and `text` column from complex data have been mapped to the `value` field and `text` field, respectively.

## APP.VUE

```
<template>  
<div id="app">
```

```

    <div id="wrapper1">
      <ejs-dropdownlist id='dropdownlist' :dataSource='itemData'
placeholder='e.g Item 1' :fields='fields' :enableVirtualization='true'
:allowFiltering='false' popupHeight="200px"></ejs-dropdownlist>
    </div>
</div>
</template>
<script>
import { DropDownListComponent, VirtualScroll } from "@syncfusion/ej2-vue-
dropdowns";
let records = [];
function dataSource() {
  for (let i = 1; i <= 150; i++) {
    let item = {
      id: 'id' + i,
      text: "Item " + i,
    };
    records.push(item);
  }
}
dataSource();
//Component registration
export default {
  name: 'App',
  components: {
    "ejs-dropdownlist": DropDownListComponent
  },
  data () {
    return {
      itemData: records,
      fields: { value: 'id', text: 'text' },
      allowFiltering: true,
    }
  },
  provide: {
    dropdownlist: [VirtualScroll]
  }
}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-inputs/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-
dropdowns/styles/material.css";
@import "../node_modules/@syncfusion/ej2-
notifications/styles/material.css";
#wrapper1{
  min-width: 250px;
  float: left;
  margin-left: 350px;
}
#wrapper2{
  min-width: 250px;
  float: right;
  margin-right: 100px;
}
</style>

```

```
{% previewsample "page.domainurl/code-snippet/drop-down-list/virtual-scroll" %}
```

### Binding remote data

The DropDownList supports retrieval of data from remote data services with the help of **DataManager** component. When using remote data, it initially fetches all the data from the server, triggering the **actionBegin** and **actionComplete** events, and then stores the data locally. During virtual scrolling, additional data is retrieved from the locally stored data, triggering the **actionBegin** and **actionComplete** events at that time as well.

The following sample displays the OrderId from the **Orders** Data Service.

### APP.VUE

```
<template>
<div id="app">
  <div id="wrapper1">
    <ejs-dropdownlist id='dropdownlist' :dataSource='itemData'
placeholder='e.g Item 1' :fields='fields' :enableVirtualization='true'
:allowFiltering='true' popupHeight="200px"></ejs-dropdownlist>
  </div>
</div>
</template>
<script>
import { DropDownListComponent, VirtualScroll } from "@syncfusion/ej2-vue-
dropdowns";
import { Query, DataManager, WebApiAdaptor } from '@syncfusion/ej2-data';
var remoteData = new DataManager({
  url: 'https://services.syncfusion.com/js/production/api/orders',
  adaptor: new WebApiAdaptor,
  crossDomain: true
});
//Component registration
export default {
  name: 'App',
  components: {
    "ejs-dropdownlist": DropDownListComponent
  },
  data () {
    return {
      itemData: remoteData,
      fields: { text: 'OrderID', value: 'OrderID' },
      allowFiltering: true,
    }
  },
  provide: {
    dropdownlist: [VirtualScroll]
  }
}
</script>
<style>
@import ".../node_modules/@syncfusion/ej2-base/styles/material.css";
@import ".../node_modules/@syncfusion/ej2-inputs/styles/material.css";
@import ".../node_modules/@syncfusion/ej2-vue-
dropdowns/styles/material.css";
```

```
@import "../../../node_modules/@syncfusion/ej2-notifications/styles/material.css";
#wrapper1{
  min-width: 250px;
  float: left;
  margin-left: 350px;
}
#wrapper2{
  min-width: 250px;
  float: right;
  margin-right: 100px;
}
</style>
```

{% previewsample "page.domainurl/code-snippet/drop-down-list/virtual-scroll-remote" %}

### Grouping

The DropDownList component supports grouping with Virtualization. It allows you to organize elements into groups based on different categories. Each item in the list can be classified using the [groupBy](#) field in the data table. After grouping, virtualization works similarly to local data binding, providing a seamless user experience. When the data source is bound to remote data, an initial request is made to retrieve all data for the purpose of grouping. Subsequently, the grouped data works in the same way as local data binding on virtualization.

The following sample shows the example for Grouping with Virtualization.

### APP.VUE

```
<template>
<div id="app">
  <div id="wrapper1">
    <ejs-dropdownlist id='dropdownlist' :dataSource='itemData'
placeholder='e.g Item 1' :fields='fields' :enableVirtualization='true'
:allowFiltering='true' popupHeight="200px"></ejs-dropdownlist>
  </div>
</div>
</template>
<script>
import { DropDownListComponent, VirtualScroll } from "@syncfusion/ej2-vue-dropdowns";
let records = [];
function dataSource() {
  for (var i = 1; i <= 150; i++) {
    var item = {};
    item.id = 'id' + i;
    item.text = "Item ".concat(i);
    var randomGroup = Math.floor(Math.random() * 4) + 1;
    switch (randomGroup) {
      case 1:
        item.group = 'Group A';
        break;
      case 2:
        item.group = 'Group B';
        break;
      case 3:
        item.group = 'Group C';
    }
  }
}
```

```

        break;
      case 4:
        item.group = 'Group D';
        break;
      default:
        break;
    }
    records.push(item);
  }
}
dataSource();
//Component registration
export default {
  name: 'App',
  components: {
    "ejs-dropdownlist": DropDownListComponent
  },
  data () {
    return {
      itemData: records,
      fields: { groupBy: 'group', value: 'id', text: 'text' },
      allowFiltering: true,
    }
  },
  provide: {
    dropdownlist: [VirtualScroll]
  }
}
</script>
<style>
@import ".../node_modules/@syncfusion/ej2-base/styles/material.css";
@import ".../node_modules/@syncfusion/ej2-inputs/styles/material.css";
@import ".../node_modules/@syncfusion/ej2-vue-
dropdowns/styles/material.css";
@import ".../node_modules/@syncfusion/ej2-
notifications/styles/material.css";
#wrapper1{
  min-width: 250px;
  float: left;
  margin-left: 350px;
}
#wrapper2{
  min-width: 250px;
  float: right;
  margin-right: 100px;
}
</style>

```

{% previewsample "page.domainurl/code-snippet/drop-down-list/virtual-scroll-group" %}

### Filtering with Virtualization

The DropDownList component supports Filtering with Virtualization. The DropDownList includes a built-in feature that enables data filtering when the [allowFiltering](#) option is enabled. In the context of Virtual Scrolling, the filtering process operates in response to the typed characters. Specifically, the DropDownList sends a request to the server, utilizing the full data source, to achieve filtering. Before



initiating the request, an action event is triggered. Upon successful retrieval of data from the server, an action complete event is triggered. The initial data is loaded when the popup is opened. Whether the filter list has a selection or not, the popup closes.

The following sample shows the example for Filtering with Virtualization.

#### APP.VUE

```
<template>
<div id="app">
  <div id="wrapper1">
    <ejs-dropdownlist id='dropdownlist' :dataSource='itemData'
placeholder='e.g Item 1' :fields='fields' :enableVirtualization='true'
:allowFiltering='true' popupHeight="200px"></ejs-dropdownlist>
  </div>
</div>
</template>
<script>
import { DropDownListComponent, VirtualScroll } from "@syncfusion/ej2-vue-
dropdowns";
let records = [];
function dataSource() {
  for (let i = 1; i <= 150; i++) {
    let item = {
      id: 'id' + i,
      text: "Item " + i,
    };
    records.push(item);
  }
}
dataSource();
//Component registration
export default {
  name: 'App',
  components: {
    "ejs-dropdownlist": DropDownListComponent
  },
  data () {
    return {
      itemData: records,
      fields: { value: 'id', text: 'text' },
      allowFiltering: true,
    }
  },
  provide: {
    dropdownlist: [VirtualScroll]
  }
}
</script>
<style>
@import "../../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../../node_modules/@syncfusion/ej2-inputs/styles/material.css";
@import "../../node_modules/@syncfusion/ej2-vue-
dropdowns/styles/material.css";
@import "../../node_modules/@syncfusion/ej2-
notifications/styles/material.css";
#wrapper1{
```

```

min-width: 250px;
float: left;
margin-left: 350px;
}
#wrapper2{
min-width: 250px;
float: right;
margin-right: 100px;
}
</style>

```

{% previewsample "page.domainurl/code-snippet/drop-down-list/virtual-scroll-filter" %}

### Grouping in Vue Drop down list component

The DropDownList supports wrapping nested elements into a group based on different categories. The category of each list item can be mapped through the [groupByLink to the Video](#) field in the data table. The group header is displayed both as inline and fixed headers. The fixed group header content is updated dynamically on scrolling the popup list with its category value.

How to group and filter the DropDownList Items, you can check on this video:

In the following sample, vegetables are grouped according on its category using `groupBy` field.

#### APP.VUE

```

<template>
  <div id="app">
    <div id='container' style="margin:20px auto 0; width:250px;">
      <br>
      <ejs-dropdownlist id='dropdownlist' popupHeight='200px'
placeholder='Select a vegetable' :fields='fields'
:dataSource='dataSource'></ejs-dropdownlist>
    </div>
  </div>
</template>
<script>
import Vue from 'vue';
import { DropDownListPlugin } from "@syncfusion/ej2-vue-dropdowns";
Vue.use(DropDownListPlugin);
export default {
  data () {
    var vegetableData = [
      { Vegetable: 'Cabbage', Category: 'Leafy and Salad', Id: 'item1' },
      { Vegetable: 'Spinach', Category: 'Leafy and Salad', Id: 'item2' },
      { Vegetable: 'Wheat grass', Category: 'Leafy and Salad', Id: 'item3' },
    ],
    { Vegetable: 'Yarrow', Category: 'Leafy and Salad', Id: 'item4' },
    { Vegetable: 'Pumpkins', Category: 'Leafy and Salad', Id: 'item5' },
    { Vegetable: 'Chickpea', Category: 'Beans', Id: 'item6' },
    { Vegetable: 'Green bean', Category: 'Beans', Id: 'item7' },
    { Vegetable: 'Horse gram', Category: 'Beans', Id: 'item8' },
    { Vegetable: 'Garlic', Category: 'Bulb and Stem', Id: 'item9' },
    { Vegetable: 'Nopal', Category: 'Bulb and Stem', Id: 'item10' },
    { Vegetable: 'Onion', Category: 'Bulb and Stem', Id: 'item11' }
  ],
  return {

```

```

        dataSource : vegetableData,
        fields : { groupBy: 'Category', text: 'Vegetable', value: 'Id' }
    }
}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-inputs/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-
dropdowns/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/drop-down-list/grouping-cs1" %}

### Customization

The grouping header is also provided with customization option. This allows custom designing using the `groupTemplate` property for both inline and fixed headers.

See Also

- [Group Template support to DropDownList.](#)
- [How to disable the fixed group header](#)

### Filtering in Vue Drop down list component

The DropDownList has built-in support to filter data items when `allowFiltering` is enabled. The filter operation starts as soon as you start typing characters in the search box.

To display filtered items in the popup, filter the required data and return it to the DropDownList via `updateData` method by using the [filteringLink to the Video](#) event.

How to group and filter the DropDownList Items, you can check on this video:

The following sample illustrates how to query the data source and pass the data to the DropDownList through the `updateData` method in `filtering` event.

### APP.VUE

```

<template>
  <div id="app">
    <div id='container' style="margin:20px auto 0; width:250px;">
      <br>
      <ejs-dropdownlist id='dropdownlist' placeholder='Select a country'
      :allowFiltering='allowFiltering' :filtering='filtering'
      :dataSource='dataSource' :fields='fields'></ejs-dropdownlist>
    </div>
  </div>
</template>
<script>
import Vue from 'vue';
import { DropDownListPlugin } from "@syncfusion/ej2-vue-dropdowns";
Vue.use(DropDownListPlugin);
import { DataManager, Query, ODataV4Adaptor, Predicate } from "@syncfusion/ej2-
data";
export default {

```

```

data () {
  var searchData = [
    { Index: "s1", Country: "Alaska" }, { Index: "s2", Country:
"California" },
    { Index: "s3", Country: "Florida" }, { Index: "s4", Country:
"Georgia" }
  ];
  return {
    dataSource : searchData,
    fields : { text: "Country", value: "Index" },
    allowFiltering: true,
  },
  methods: {
    filtering: function(e) {
      var searchData = [
        { Index: "s1", Country: "Alaska" }, { Index: "s2", Country:
"California" },
        { Index: "s3", Country: "Florida" }, { Index: "s4", Country:
"Georgia" }
      ];
      var query = new Query();
      //frame the query based on search string with filter type.
      query = (e.text != "") ? query.where("Country", "startswith",
e.text, true) : query;
      //pass the filter data source, filter query to updateData
      method.
      e.updateData(searchData, query);
    }
  }
}
</script>
<style>
@import "../../../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../../../node_modules/@syncfusion/ej2-inputs/styles/material.css";
@import "../../../node_modules/@syncfusion/ej2-vue-
dropdowns/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/drop-down-list/filtering/default-cs1" %}

### Limit the minimum filter character

When filtering the list items, you can set the limit for character count to raise remote request and fetch filtered data on the DropDownList. This can be done by manual validation within the filter event handler.

In the following example, the remote request does not fetch the search data until the search key contains three characters

### APP.VUE

```

<template>
  <div id="app">
    <div id='container' style="margin:20px auto 0; width:250px;">
      <br>
    </div>
  </div>
</template>

```

```

        <ejs-dropdownlist id='dropdownlist' placeholder='Select a name'
        sortOrder="Ascending" :query='query' :allowFiltering='allowFiltering'
        :filtering='filtering' :dataSource='dataSource' :fields='fields'></ejs-
        dropdownlist>
    </div>
</div>
</template>
<script>
import Vue from 'vue';
import { DropDownListPlugin } from "@syncfusion/ej2-vue-dropdowns";
Vue.use(DropDownListPlugin);
import { DataManager, Query, ODataV4Adaptor, Predicate } from "@syncfusion/ej2-
data";
export default {
    data () {
        return {
            dataSource : new DataManager({
                url:
                'https://services.odata.org/V4/Northwind/Northwind.svc/Customers',
                adaptor: new ODataV4Adaptor,
                crossDomain: true
            }),
            query : new Query().select(['ContactName', 'CustomerID']).take(7),
            fields : { text: 'ContactName', value: 'CustomerID' },
            allowFiltering: true,
        }
    },
    methods: {
        filtering: function(e) {
            var searchData = new DataManager({
                url:
                'https://services.odata.org/V4/Northwind/Northwind.svc/Customers',
                adaptor: new ODataV4Adaptor,
                crossDomain: true
            });
            if(e.text == '') e.updateData(searchData);
            else{
                // restrict the remote request until search key contains 3
                characters.
                if (e.text.length < 3) { return; }
                var query = new Query().select(['ContactName',
                'CustomerID']);
                query = (e.text !== '') ? query.where('ContactName',
                'startswith', e.text, true) : query;
                e.updateData(searchData, query);
            }
        }
    }
}
</script>
<style>
@import "../../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../../node_modules/@syncfusion/ej2-inputs/styles/material.css";
@import "../../node_modules/@syncfusion/ej2-vue-
dropdowns/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/drop-down-list/filtering/min-filter-char-cs1" %}

### Change the filter type

While filtering, you can change the filter type to `contains`, `startsWith`, or `endsWith` for string type within the filter event handler

In the following examples, data filtering is done with `endsWith` type.

### APP.VUE

```
<template>
  <div id="app">
    <div id='container' style="margin:20px auto 0; width:250px;">
      <br>
      <ejs-dropdownlist id='dropdownlist' placeholder='Select a name'
popupHeight='250px' sortOrder="Ascending" :query='query'
:allowFiltering='allowFiltering' :filtering='filtering'
:dataSource='dataSource' :fields='fields'></ejs-dropdownlist>
    </div>
  </div>
</template>
<script>
import Vue from 'vue';
import { DropDownListPlugin } from "@syncfusion/ej2-vue-dropdowns";
Vue.use(DropDownListPlugin);
import { DataManager, Query, ODataV4Adaptor, Predicate } from "@syncfusion/ej2-
data";
export default {
  data () {
    return {
      dataSource : new DataManager({
        url:
'https://services.odata.org/V4/Northwind/Northwind.svc/Customers',
        adaptor: new ODataV4Adaptor,
        crossDomain: true
      }),
      query : new Query().select(['ContactName', 'CustomerID']).take(7),
      fields : { text: 'ContactName', value: 'CustomerID' },
      allowFiltering: true,
    }
  },
  methods: {
    filtering: function(e) {
      var searchData = new DataManager({
        url:
'https://services.odata.org/V4/Northwind/Northwind.svc/Customers',
        adaptor: new ODataV4Adaptor,
        crossDomain: true
      });
      // load overall data when search key empty.
      if(e.text == '') e.updateData(searchData);
      else{
        let query: Query = new Query().select(['ContactName',
'CustomerID']);
        // change the type of filtering
```

```

        query = (e.text !== '') ? query.where('ContactName',
        'endswith', e.text, true) : query;
        e.updateData(searchData, query);
    }
}
}
}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-inputs/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-
dropdowns/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/drop-down-list/filtering/filter-type-cs1" %}

### Case sensitive filtering

Data items can be filtered either with or without case sensitivity using the DataManager. This can be done by passing the fourth optional parameter of the `where` clause.

The following example shows how to perform case-sensitive filter

### APP.VUE

```

<template>
  <div id="app">
    <div id='container' style="margin:20px auto 0; width:250px;">
      <br>
      <ejs-dropdownlist id='dropdownlist' placeholder='Select a name'
      popupHeight='250px' sortOrder="Ascending" :query='query'
      :allowFiltering='allowFiltering' :filtering='filtering'
      :dataSource='dataSource' :fields='fields'></ejs-dropdownlist>
    </div>
  </div>
</template>
<script>
import Vue from 'vue';
import { DropDownListPlugin } from "@syncfusion/ej2-vue-dropdowns";
Vue.use(DropDownListPlugin);
import { DataManager, Query, ODataV4Adaptor, Predicate } from "@syncfusion/ej2-
data";
export default {
  data () {
    return {
      dataSource : new DataManager({
        url:
        'https://services.odata.org/V4/Northwind/Northwind.svc/Customers',
        adaptor: new ODataV4Adaptor,
        crossDomain: true
      }),
      query : new Query().select(['ContactName', 'CustomerID']).take(7),
      fields : { text: 'ContactName', value: 'CustomerID' },
      allowFiltering: true,
    }
  },
},

```

```

methods: {
  filtering: function(e) {
    var searchData = new DataManager({
      url:
'https://services.odata.org/V4/Northwind/Northwind.svc/Customers',
      adaptor: new ODataV4Adaptor,
      crossDomain: true
    });
    // load overall data when search key empty.
    if(e.text == '') e.updateData(searchData);
    else{
      let query: Query = new Query().select(['ContactName',
'CustomerID']);
      //enable the case sensitive filtering by passing false to
4th parameter.
      query = (e.text !== '') ? query.where('ContactName',
'startswith', e.text, false) : query;
      e.updateData(searchData, query);
    }
  }
}
}
</script>
<style>
@import "../../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../../node_modules/@syncfusion/ej2-inputs/styles/material.css";
@import "../../node_modules/@syncfusion/ej2-vue-
dropdowns/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/drop-down-list/filtering/case-sensitive-filter-cs1" %}

### Diacritics Filtering

The DropDownList supports diacritics filtering which will ignore the [diacritics](#) and makes it easier to filter the results in international characters lists when the [ignoreAccent](#) is enabled.

In the following sample, data with diacritics are bound as dataSource for DropDownList.

### APP.VUE

```

<template>
  <div id="app">
    <div id='container' style="margin:20px auto 0; width:250px;">
      <br>
      <ejs-dropdownlist id='dropdownlist' placeholder='Select a value'
filterBarPlaceholder='e.g: aero' :ignoreAccent='ignoreAccent'
:allowFiltering='allowFiltering' :dataSource='dataSource'></ejs-
dropdownlist>
    </div>
  </div>
</template>
<script>
import Vue from 'vue';
import { DropDownListPlugin } from "@syncfusion/ej2-vue-dropdowns";
Vue.use(DropDownListPlugin);

```



```

import { DataManager, Query, ODataV4Adaptor, Predicate } from "@syncfusion/ej2-data";
export default {
  data () {
    return {
      dataSource : [
        'Aeróbics',
        'Aeróbics en Agua',
        'Aerografía',
        'Aeromodelaje',
        'Águilas',
        'Ajedrez',
        'Ala Delta',
        'Álbumes de Música',
        'Alusivos',
        'Análisis de Escritura a Mano'
      ],
      allowFiltering : true,
      ignoreAccent : true
    }
  }
}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-inputs/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-dropdowns/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/drop-down-list/filtering/diacritics-filter-cs1" %}

See Also

- [How to limit the search result while filtering](#)
- [How to highlight the matched characters in filtering](#)
- [How to modify the result data using remote data source](#)

## Localization in Vue Drop down list component

The Localization library allows you to localize static text content of the [noRecordsTemplate](#) and [actionFailureTemplate](#) properties according to the culture currently assigned to the DropDownList.

| Locale key | en-US (default) |

|-----|-----|

| noRecordsTemplate | No records found |

| actionFailureTemplate | The request failed |

## Loading translations

To load translation object to your application, use load function of the **L10n** class.

In the following sample, French culture is set to the DropDownList and no data is loaded. Hence, the `noRecordsTemplate` property displays its text in French culture initially, and if the sample is run offline, the `actionFailureTemplate` property displays its text appropriately.

#### APP.VUE

```
<template>
  <div id="app">
    <div id='container' style="margin:20px auto 0; width:250px;">
      <br>
      <ejs-dropdownlist id='dropdownlist' placeholder='Sélectionnez un
client' locale="fr-BE" :fields='fields' :query='query'
:dataSource='dataSource'></ejs-dropdownlist>
    </div>
  </div>
</template>
<script>
import Vue from 'vue';
import { DropDownListPlugin } from "@syncfusion/ej2-vue-dropdowns";
Vue.use(DropDownListPlugin);
import { DataManager, Query, ODataV4Adaptor, Predicate } from "@syncfusion/ej2-
data";
import { L10n } from '@syncfusion/ej2-base';
export default {
  data () {
    var data = new DataManager({
      url:
'https://services.odata.org/V4/Northwind/Northwind.svc/Customers',
      adaptor: new ODataV4Adaptor,
      crossDomain: true
    });
    var query = new Query().select(['ContactName', 'CustomerID']).take(0);
    L10n.load({
      'fr-BE': {
        'dropdowns': {
          'noRecordsTemplate': "Aucun enregistrement trouvé",
          'actionFailureTemplate': "Modèle d'échec d'action"
        }
      }
    });
    return {
      fields: { text: 'ContactName', value: 'CustomerID' },
      dataSource: data,
      query: query
    }
  }
}
</script>
<style>
@import ".../node_modules/@syncfusion/ej2-base/styles/material.css";
@import ".../node_modules/@syncfusion/ej2-inputs/styles/material.css";
@import ".../node_modules/@syncfusion/ej2-vue-
dropdowns/styles/material.css";
</style>
```

{% previewsample "page.domainurl/code-snippet/drop-down-list/locale-cs1" %}

## See Also

- [Accessibility](#)
- [How to bind the data to the combobox](#)

## Style in Vue Drop down list component

The following content provides the exact CSS structure that can be used to modify the control's appearance based on the user preference.

### Customizing the appearance of wrapper element

Use the following CSS to customize the appearance of wrapper element.

```
、  
  
.e-ddl.e-input-group.e-control-wrapper .e-input {  
font-size: 20px;  
font-family: emoji;  
color: #ab3243;  
background: #32a5ab;  
}  
、
```

### Customizing the dropdown icon's color

Use the following CSS to customize the dropdown icon's color.

```
、  
  
.e-ddl.e-input-group .e-input-group-icon,.e-ddl.e-input-group.e-control-wrapper .e-input-group-  
icon:hover {  
color: #bb233d;  
font-size: 13px;  
}  
、
```

### Customizing the focus color

Use the following CSS to customize the focusing color of input element.

```
、  
  
.e-ddl.e-input-group.e-control-wrapper.e-input-focus::before, .e-ddl.e-input-group.e-control-wrapper.e-  
input-focus::after {  
background: #c000ff;  
}  
、
```

### Customizing the outline theme's focus color

Use the following CSS to customize the focusing color of outline theme.

```

、

.e-outline.e-input-group.e-input-focus:hover:not(.e-success):not(.e-warning):not(.e-error):not(.e-disabled):not(.e-float-icon-left),.e-outline.e-input-group.e-input-focus.e-control-wrapper:hover:not(.e-success):not(.e-warning):not(.e-error):not(.e-disabled):not(.e-float-icon-left),.e-outline.e-input-group.e-input-focus:not(.e-success):not(.e-warning):not(.e-error):not(.e-disabled),.e-outline.e-input-group.e-control-wrapper.e-input-focus:not(.e-success):not(.e-warning):not(.e-error):not(.e-disabled) {
border-color: #b1bd15;
box-shadow: inset 1px 1px #b1bd15, inset -1px 0 #b1bd15, inset 0 -1px #b1bd15;
}
、

```

#### Customizing the disabled component's text color

Use the following CSS to customize the text color when the component is disabled.

```

、

.e-input-group.e-control-wrapper .e-input[disabled] {
-webkit-text-fill-color: #0d9133;
}
、

```

#### Customizing the float label element's focusing color

Use the following CSS to customize the focusing color of float label element.

```

、

.e-float-input.e-input-group:not(.e-float-icon-left) .e-float-line::before,.e-float-input.e-control-wrapper.e-input-group:not(.e-float-icon-left) .e-float-line::before,.e-float-input.e-input-group:not(.e-float-icon-left) .e-float-line::after,.e-float-input.e-control-wrapper.e-input-group:not(.e-float-icon-left) .e-float-line::after {
background-color: #2319b8;
}

.e-ddl.e-lib.e-input-group.e-control-wrapper.e-float-input.e-input-focus .e-float-text.e-label-top {
color: #2319b8;
}
、

```

#### Customizing the color of the placeholder text

Use the following CSS to customize the text color of placeholder.

```

、

.e-ddl.e-input-group input.e-input::placeholder {
color: red;
}

```

### Customizing the background color of focus, hover, and active item's

Use the following CSS to customize the background color of focus, hover and active item's.

```
.e-dropdownbase .e-list-item.e-item-focus, .e-dropdownbase .e-list-item.e-active, .e-dropdownbase .e-list-item.e-active.e-hover, .e-dropdownbase .e-list-item.e-hover {
background-color: #1f9c99;
color: #2319b8;
}
```

### Customizing the appearance of pop-up element

Use the following CSS to customize the appearance of popup element.

```
.e-dropdownbase .e-list-item, .e-dropdownbase .e-list-item.e-item-focus {
background-color: #29c2b8;
color: #207cd9;
font-family: emoji;
min-height: 29px;
}
```

### Adding mandatory asterisk to placeholder and float label

You can add a mandatory asterisk(\*) to placeholder and float label using `<b>.e-input-group.e-control-wrapper.e-float-input .e-float-text::after</b>` class.

### APP.VUE

```
<template>
  <div id="app">
    <div id='container' style="margin:50px auto 0; width:250px;">
      <br>
      <ejs-dropdownlist id='dropdownlist' :dataSource='sportsData'
placeholder='Select a game' :floatLabelType= "auto"></ejs-dropdownlist>
    </div>
  </div>
</template>
<script>
import Vue from 'vue';
import { DropDownListPlugin } from "@syncfusion/ej2-vue-dropdowns";
Vue.use(DropDownListPlugin);
export default {
  data () {
    return {
      sportsData: ['Badminton', 'Cricket', 'Football', 'Golf', 'Tennis'];
    }
  }
}
```

```

    }
  }
</script>
<style>
@import ".../node_modules/@syncfusion/ej2-base/styles/material.css";
@import ".../node_modules/@syncfusion/ej2-inputs/styles/material.css";
@import ".../node_modules/@syncfusion/ej2-vue-
dropdowns/styles/material.css";
.e-input-group.e-control-wrapper.e-float-input .e-float-text::after {
  content: '*';
  color: red;
}
</style>

```

```
{% previewsample "page.domainurl/code-snippet/drop-down-list/getting-started/getting-started-cs2"
%}
```

### Accessibility in Vue Drop down list component

The DropDownList component has been designed, keeping in mind the WAI-ARIA specifications, and applies the WAI-ARIA roles, states, and properties along with keyboard support. This component is characterized by complete keyboard interaction support and ARIA accessibility support that makes it easy for people who use assistive technologies (AT) or those who completely rely on keyboard navigation.

The DropDownList component followed the accessibility guidelines and standards, including [ADA, Section 508](#), [WCAG 2.2](#) standards, and [WCAG roles](#) that are commonly used to evaluate accessibility.

The accessibility compliance for the DropDownList component is outlined below.

| Accessibility Criteria | Compatibility |

| -- | -- |

| [WCAG 2.2 Support](#) |  |

| [Section 508 Support](#) |  |

| [Screen Reader Support](#) |  |

| [Right-To-Left Support](#) |  |

| [Color Contrast](#) |  |

| [Mobile Device Support](#) |  |

| [Keyboard Navigation Support](#) |  |

| [Accessibility Checker Validation](#) |  |

| [Axe-core Accessibility Validation](#) |  |

<style>

.post .post-content img {

display: inline-block;

margin: 0.5em 0;

}

</style>

<div> - All features of the component meet the requirement.</div>

<div> - Some features of the component do not meet the requirement.</div>

<div> - The component does not meet the requirement.</div>

### WAI-ARIA attributes

The DropDownList component uses the **Listbox** role, and each list item has an **option** role. The following **ARIA attributes** denote the DropDownList state.

#### | Properties | Functionalities |

| --- | --- |

| aria-haspopup | Indicates whether the DropDownList input element has a popup list or not. |

| aria-expanded | Indicates whether the popup list has expanded or not. |

| aria-selected | Indicates the selected option. |

| aria-readonly | Indicates the readonly state of the DropDownList element. |

| aria-disabled | Indicates whether the DropDownList component is in a disabled state or not. |

| aria-activedescendent | This attribute holds the ID of the active list item to focus its descendant child element. |

| aria-owns | This attribute contains the ID of the popup list to indicate popup as a child element. |

### Keyboard interaction

You can use the following key shortcuts to access the DropDownList without interruptions.

#### | Keyboard shortcuts | Actions |

| --- | --- |

| Arrow Down | Selects the first item in the DropDownList when no item selected. Otherwise, selects the item next to the currently selected item. |

| Arrow Up | Selects the item previous to the currently selected one. |

| Page Down | Scrolls down to the next page and selects the first item when popup list opens. |

| Page Up | Scrolls up to the previous page and selects the first item when popup list opens. |

| Enter | Selects the focused item, and when it is in an open state the popup list closes. Otherwise, toggles the popup list. |

| Tab | Focuses on the next TabIndex element on the page when the popup is closed. Otherwise, closes the popup list and remains the focus of the component. |

| Shift + tab | Focuses on the previous TabIndex element on the page when the popup is closed. Otherwise, closes the popup list and remains the focus of the component. |

| Alt + Down | Opens the popup list. |

| Alt + Up | Closes the popup list. |

| Esc(Escape) | Closes the popup list when it is in an open state and the currently selected item remains the same. |

| Home | Selects the first item. |

| End | Selects the last item. |

In the below sample, focus the DropDownList component using alt+t keys.

#### APP.VUE

```
<template>
  <div id="app">
    <div id='container' style="margin:50px auto 0; width:250px;">
      <br>
      <ejs-dropdownlist id='dropdownlist' ref="DropdownInstance"
placeholder='Select a game' popupHeight='200px' :fields='fields'
:dataSource='dataSource'></ejs-dropdownlist>
    </div>
  </div>
</template>
<script>
import Vue from 'vue';
import { DropDownListPlugin } from "@syncfusion/ej2-vue-dropdowns";
Vue.use(DropDownListPlugin);
export default {
  mounted () {
    let dropdownObj = this.$refs.DropdownInstance;
    document.onkeyup = function (e) {
      if (e.altKey && e.keyCode === 84) {
        // press alt+t to focus the control.
        dropdownObj.$el.focus();
      }
    },
    data () {
      return {
        dataSource: [
          { Id: 'Game1', Game: 'Badminton' },
          { Id: 'Game2', Game: 'Basketball' },
          { Id: 'Game3', Game: 'Cricket' },
          { Id: 'Game4', Game: 'Football' },
        ],
      }
    }
  }
}
```



```

        { Id: 'Game5', Game: 'Golf' },
        { Id: 'Game6', Game: 'Hockey' },
        { Id: 'Game7', Game: 'Rugby' },
        { Id: 'Game8', Game: 'Snooker' },
        { Id: 'Game9', Game: 'Tennis' }
    ],
    fields: { text: 'Game', value: 'Id' }
}
}
}
</script>
<style>
@import ".../node_modules/@syncfusion/ej2-base/styles/material.css";
@import ".../node_modules/@syncfusion/ej2-inputs/styles/material.css";
@import ".../node_modules/@syncfusion/ej2-vue-
dropdowns/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/drop-down-list/accessibility-cs1" %}

### Ensuring accessibility

The DropDownList component's accessibility levels are ensured through an [accessibility-checker](#) and [axe-core](#) software tools during automated testing.

The accessibility compliance of the DropDownList component is shown in the following sample. Open the [sample](#) in a new window to evaluate the accessibility of the DropDownList component with accessibility tools.

{% previewsample "https://ej2.syncfusion.com/accessibility/drop-down-list.html" %}

See also

- [Accessibility in Syncfusion Vue components](#)

### Two way binding in Vue Drop down list component

Two-way binding can be achieved by using the `v-model` directive in Vue. In the following sample, when you change the value in one DropDownList component, v-model will automatically update the value in the other DropDownList.

The following example demonstrates how to set the **two-way-binding** in the DropDownList.

#### APP.VUE

```

<template>
<div id="app">
  <div id="wrapper1">
    <ejs-dropdownlist id='first':dataSource='sportsData'
:fields='fields' :placeholder="waterMark" v-model="value" ></ejs-
dropdownlist>
  </div>
  <div id="wrapper2">
    <ejs-dropdownlist id='second' :dataSource='sportsData'
:fields='fields' :placeholder="waterMark" v-model="value" ></ejs-
dropdownlist>
  </div>

```

```

</div>
</template>
<script>
import Vue from 'vue';
import { DropDownListPlugin } from '@syncfusion/ej2-vue-dropdowns';
Vue.use(DropDownListPlugin);
export default {
  name: 'app',
  data () {
    return {
      waterMark : 'Select a game',
      value: null,
      sportsData: [
        { Id: 'Game1', Game: 'Badminton' },
        { Id: 'Game2', Game: 'Basketball' },
        { Id: 'Game3', Game: 'Cricket' },
        { Id: 'Game4', Game: 'Football' },
        { Id: 'Game5', Game: 'Golf' },
        { Id: 'Game6', Game: 'Hockey' },
        { Id: 'Game7', Game: 'Rugby' },
        { Id: 'Game8', Game: 'Snooker' }
      ],
      fields: { value: 'Game' }
    }
  }
}
</script>
<style>
@import "../../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../../node_modules/@syncfusion/ej2-inputs/styles/material.css";
@import "../../node_modules/@syncfusion/ej2-vue-dropdowns/styles/material.css";
#wrapper1{
  min-width: 250px;
  float: left;
  margin-left: 100px;
}
#wrapper2{
  min-width: 250px;
  float: right;
  margin-right: 100px;
}
</style>

```

{% previewsample "page.domainurl/code-snippet/drop-down-list/two-way-cs1" %}

## How To

### Add item in Vue Drop down list component

You can add item in between based on item [indexLink to the Video](#). If you add new item without item index, item will be added as last item in list.

To add and remove items dynamically in Vue DropDownList component, you can check on this video:

The following example demonstrate how to add item in between in DropDownList.

### APP.VUE

```

<template>
  <div id="app">
    <div id='container' style="margin:0 auto; width:250px;">
      <br>
      <ejs-dropdownlist id='dropdownlist' ref='dropdown' :index='index'
      :dataSource='sportsData' :fields='fields' placeholder='Select a game'></ejs-
      dropdownlist>
    </div>
    <div style='padding: 50px 0'>
      <button id='first' class="e-control e-btn" v-on:click="onFirst"> add
      item (Hockey) in first</button>
    </div>
    <div style='padding-left: 50px 0'>
      <button id='between' class="e-control e-btn" v-on:click="onMiddle">
      add item (Golf) in between</button>
    </div>
    <div style='padding: 50px 0'>
      <button id='last' class="e-control e-btn" v-on:click="onLast"> add
      item (Cricket) in last</button>
    </div>
  </div>
</template>
<script>
import Vue from 'vue';
import { DropDownListPlugin } from "@syncfusion/ej2-vue-dropdowns";
Vue.use(DropDownListPlugin);
export default {
  data () {
    return {
      sportsData: [
        { Id: 'game1', Game: 'Badminton' },
        { Id: 'game2', Game: 'Football' },
        { Id: 'game3', Game: 'Tennis' }
      ],
      fields : { text: 'Game', value: 'Id' },
      index : 2
    }
  },
  methods: {
    onFirst: function (event) {
      this.$refs.dropdown.addItem({Id: 'game0', Game: 'Hockey'}, 0);
    },
    onMiddle: function (event) {
      this.$refs.dropdown.addItem({Id: 'game4', Game: 'Golf'}, 2);
    },
    onLast: function (event) {
      this.$refs.dropdown.addItem({Id: 'game5', Game: 'Cricket'});
    }
  }
}
</script>
<style>
@import "../../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../../node_modules/@syncfusion/ej2-inputs/styles/material.css";
@import "../../node_modules/@syncfusion/ej2-vue-
dropdowns/styles/material.css";
</style>

```

```
{% previewsample "page.domainurl/code-snippet/drop-down-list/how-to/valuechange-cs1" %}
```

### Cascading in Vue Drop down list component

The cascading DropDownList is a series of DropDownList, where the value of one DropDownList depends upon another's value. This can be configured by using the [change](#) event of the parent DropDownList. Within that change event handler, data has to be loaded to the child DropDownList based on the selected value of the parent DropDownList.

The following example, shows the cascade behavior of country, state, and city DropDownList. Here, the `dataBind` method is used to reflect the property changes immediately to the DropDownList.

#### APP.VUE

```
<template>
  <div id="app">
    <div id='container' style="margin:50px auto 0; width:250px;">
      <br>
      <ejs-dropdownlist id='countries' ref='countries' :query='Query1'
      :dataSource='countryData' :index='countryindex' :fields='countryfields'
      :change='onCountryChange' placeholder='Select a country'></ejs-dropdownlist>
      <div class="padding-top">
        <ejs-dropdownlist id='states' ref='states' :query='Query2'
        :dataSource='stateData' :index='stateindex' :fields='statefields'
        :enabled='stateenabled' :change='onStateChange' placeholder='Select a
        state'></ejs-dropdownlist>
      </div>
      <div class="padding-top">
        <ejs-dropdownlist id='cities' ref='cities' :query='Query3'
        :dataSource='cityData' :index='cityindex' :enabled='cityenabled'
        :fields='cityfields' placeholder='Select a city'></ejs-dropdownlist>
      </div>
    </div>
  </div>
</template>
<script>
import Vue from 'vue';
import { DropDownListPlugin } from "@syncfusion/ej2-vue-dropdowns";
Vue.use(DropDownListPlugin);
import { Query } from '@syncfusion/ej2-data';
export default {
  data () {
    return {
      countryData: [
        { CountryName: 'United States', CountryId: '1' },
        { CountryName: 'Australia', CountryId: '2' }
      ],
      stateData: [
        { StateName: 'New York', CountryId: '1', StateId: '101' },
        { StateName: 'Virginia ', CountryId: '1', StateId: '102' },
        { StateName: 'Tasmania ', CountryId: '2', StateId: '105' }
      ],
      cityData : [
        { CityName: 'Albany', StateId: '101', CityId: 201 },
        { CityName: 'Beacon ', StateId: '101', CityId: 202 },
        { CityName: 'Emporia', StateId: '102', CityId: 206 },
      ]
    }
  }
}
```

```

        { CityName: 'Hampton ', StateId: '102', CityId: 205 },
        { CityName: 'Hobart', StateId: '105', CityId: 213 },
        { CityName: 'Launceston ', StateId: '105', CityId: 214 }
    ],
    countryfields : { value: 'CountryId', text: 'CountryName' },
    statefields : { value: 'StateId', text: 'StateName' },
    cityfields : { text: 'CityName', value: 'CityId' },
    countryindex : 1,
    stateindex : 0,
    cityindex : 0,
    cityenabled : false,
    stateenabled : false,
    Query1: null,
    Query2: null,
    Query3: null
    }
},
methods: {
    onCountryChange: function(e) {
        this.Query2 = new Query().where('CountryId', 'equal',
this.$refs.countries.ej2Instances.value);
        this.stateenabled = true;
    },
    onStateChange: function(e) {
        this.Query3 = new Query().where('StateId', 'equal',
this.$refs.states.ej2Instances.value);
        this.cityenabled = true;
    }
}
}
</script>
<style>
@import ".../node_modules/@syncfusion/ej2-base/styles/material.css";
@import ".../node_modules/@syncfusion/ej2-inputs/styles/material.css";
@import ".../node_modules/@syncfusion/ej2-vue-
dropdowns/styles/material.css";
#container .padding-top {
    padding-top: 35px;
}
</style>

```

{% previewsample "page.domainurl/code-snippet/drop-down-list/how-to/cascade-cs1" %}

### Clear item in Vue Drop down list component

You can clear the selected item in the below two different ways.

By clicking on the **clear icon** which is shown in DropDownList element, you can clear the selected item in DropDownList through **interaction**. By using [showClearButton](#) property, you can enable the clear icon in DropDownList element.

Through **programmatic** you can set **null** value to anyone of the index, text or value property to clear the selected item in DropDownList.

The following example demonstrate about how to clear the selected item in DropDownList.

### APP.VUE

```

<template>
  <div id="app">
    <div id='container' style="margin:0 auto; width:250px;">
      <br>
      <ejs-dropdownlist id='dropdownlist' ref='dropdownObj'
: value='dropDownValue' :dataSource='sportsData' :showClearButton='true'
placeholder='Select a game'></ejs-dropdownlist>
    </div>
    <div style='padding: 50px'>
      <button id='button' class="e-control e-btn" v-on:click="onClick"> Set
null to value property</button>
    </div>
  </div>
</template>
<script>
import Vue from 'vue';
import { DropDownListPlugin } from "@syncfusion/ej2-vue-dropdowns";
Vue.use(DropDownListPlugin);
export default {
  data () {
    return {
      dropDownValue: null,
      sportsData: ["American Football", "Badminton", "Basketball",
"Cricket", "Football", "Golf", "Hockey", "Rugby", "Snooker", "Tennis"]
    }
  },
  methods: {
    onClick: function (event) {
      this.$refs.dropdownObj.ej2Instances.value = null;
      this.$refs.dropdownObj.dataBind();
    }
  }
}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-inputs/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-
dropdowns/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/drop-down-list/how-to/value-cs1" %}

### Close popup in Vue Drop down list component

By using the `hidePopup` method in `DropDownList`, you can close the popup on scroll when triggered the windows scroll event.

The following example demonstrate about how to close the popup on scroll.

#### APP.VUE

```

<template>
  <div id="app">
    <div style = 'padding: 50px'>
      <h4> You can close the opened popup by scroll the page.</h4>
    </div>
  </div>

```

```

<div id='container' style="margin:0 auto; width:250px;">
  <br>
  <ejs-dropdownlist id='dropdownlist' popupHeight="220px"
:dataSource='sportsData' placeholder='Select a game'></ejs-dropdownlist>
</div>
</div>
</template>
<script>
import Vue from 'vue';
import { DropDownListPlugin } from "@syncfusion/ej2-vue-dropdowns";
Vue.use(DropDownListPlugin);
export default {
  data () {
    return {
      sportsData: ['Badminton', 'Cricket', 'Football', 'Golf', 'Tennis']
    }
  }
}
document.onscroll = () => {
  var dropObj = document.getElementById("dropdownlist"); //to get
  dropdown list object
  dropObj.ej2_instances[0].hidePopup(); // hide the popup using
  hidePopup method
};
</script>
<style>
@import "../../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../../node_modules/@syncfusion/ej2-inputs/styles/material.css";
@import "../../node_modules/@syncfusion/ej2-vue-
dropdowns/styles/material.css";
body {
  height: 500px;
}
</style>

```

{% previewsample "page.domainurl/code-snippet/drop-down-list/how-to/scroll-cs1" %}

### Group header in Vue Drop down list component

The following example demonstrate about how to disable the Fixed group header in DropDownList through CSS by using visibility attribute.

#### APP.VUE

```

<template>
  <div id="app">
    <div id='container' style="margin:20px auto 0; width:250px;">
      <br>
      <ejs-dropdownlist id='dropdownlist' popupHeight='200px'
placeholder='Select a vegetable' :fields='fields'
:dataSource='dataSource'></ejs-dropdownlist>
    </div>
  </div>
</template>
<script>
import Vue from 'vue';
import { DropDownListPlugin } from "@syncfusion/ej2-vue-dropdowns";

```

```

Vue.use(DropDownListPlugin);
export default {
  data () {
    var vegetableData = [
      { Vegetable: 'Cabbage', Category: 'Leafy and Salad', Id: 'item1' },
      { Vegetable: 'Spinach', Category: 'Leafy and Salad', Id: 'item2' },
      { Vegetable: 'Wheat grass', Category: 'Leafy and Salad', Id: 'item3' },
    ],
    { Vegetable: 'Yarrow', Category: 'Leafy and Salad', Id: 'item4' },
    { Vegetable: 'Pumpkins', Category: 'Leafy and Salad', Id: 'item5' },
    { Vegetable: 'Chickpea', Category: 'Beans', Id: 'item6' },
    { Vegetable: 'Green bean', Category: 'Beans', Id: 'item7' },
    { Vegetable: 'Horse gram', Category: 'Beans', Id: 'item8' },
    { Vegetable: 'Garlic', Category: 'Bulb and Stem', Id: 'item9' },
    { Vegetable: 'Nopal', Category: 'Bulb and Stem', Id: 'item10' },
    { Vegetable: 'Onion', Category: 'Bulb and Stem', Id: 'item11' }
  ];
  return {
    dataSource : vegetableData,
    fields : { groupBy: 'Category', text: 'Vegetable', value: 'Id' }
  }
}
</script>
<style>
@import "../../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../../node_modules/@syncfusion/ej2-inputs/styles/material.css";
@import "../../node_modules/@syncfusion/ej2-vue-
dropdowns/styles/material.css";
.e-ddl.e-dropdownbase.e-fixed-head {
  visibility: hidden;
}
</style>

```

{% previewsample "page.domainurl/code-snippet/drop-down-list/how-to/header-cs1" %}

### Highlight filtering in Vue Drop down list component

By using the **highlightSearch** method, you can highlight the matched character in DropDownList filtering.

The following example demonstrates about how to highlight the matched character in filtering.

#### APP.VUE

```

<template>
  <div id="app">
    <div id='container' style="margin:20px auto 0; width:250px;">
      <br>
      <ejs-dropdownlist id='dropdownlist' placeholder='Select a customer'
      popupHeight='250px' sortOrder="Ascending" :query='query'
      :allowFiltering='allowFiltering' :filtering='filtering'
      :dataSource='dataSource' :fields='fields'></ejs-dropdownlist>
    </div>
  </div>
</template>
<script>

```



```

import Vue from 'vue';
import { DropDownListPlugin } from "@syncfusion/ej2-vue-dropdowns";
Vue.use(DropDownListPlugin);
import { DataManager, Query, ODataV4Adaptor, Predicate } from "@syncfusion/ej2-data";
import { DropDownList, FilteringEventArgs, highlightSearch } from '@syncfusion/ej2-dropdowns';
export default {
  data () {
    return {
      dataSource : new DataManager({
        url:
'https://services.odata.org/V4/Northwind/Northwind.svc/Customers',
        adaptor: new ODataV4Adaptor,
        crossDomain: true
      }),
      query : new Query().select(['ContactName', 'CustomerID']).take(7),
      fields : { text: 'ContactName', value: 'CustomerID' },
      allowFiltering: true,
    }
  },
  methods: {
    filtering: function(e) {
      var searchData = new DataManager({
        url:
'https://services.odata.org/V4/Northwind/Northwind.svc/Customers',
        adaptor: new ODataV4Adaptor,
        crossDomain: true
      });
      var text = e.text;
      var query = new Query().select(['ContactName', 'CustomerID']);
      //frame the query based on search string with filter type.
      query = (e.text !== '') ? query.where('ContactName',
'startswith', e.text, true) : query;
      //pass the filter data source, filter query to updateData
      method.
      e.updateData(searchData, query);
      setTimeout(() => {
        var popupElement =
document.getElementById('dropdownlist_popup');
        // get the list from popup element
        var lists = popupElement.querySelector('ul');
        // pass the element, text, ignore case and filter type as
argument to highlightSearch method.
        highlightSearch(lists, text, true, 'StartsWith');
      }, 300);
    }
  }
}
</script>
<style>
@import "../../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../../node_modules/@syncfusion/ej2-inputs/styles/material.css";
@import "../../node_modules/@syncfusion/ej2-vue-
dropdowns/styles/material.css";
</style>

```

```
{% previewsample "page.domainurl/code-snippet/drop-down-list/how-to/highlight-cs1" %}
```

### Icons support in Vue Drop down list component

You can render **icons** to the list items by mapping the [iconCss](#) field. This `iconCss` field create a span in the list item with mapped class name to allow styling as per your need.

In the following sample, icon classes are mapped with `iconCss` field.

#### APP.VUE

```
<template>
  <div id="app">
    <div id='container' style="margin:50px auto 0; width:250px;">
      <br>
      <ejs-dropdownlist id='dropdownlist' :dataSource='sortFormatData'
:fields='fields' placeholder='Select a format'></ejs-dropdownlist>
    </div>
  </div>
</template>
<script>
import Vue from 'vue';
import { DropDownListPlugin } from "@syncfusion/ej2-vue-dropdowns";
Vue.use(DropDownListPlugin);
export default {
  data () {
    return {
      sortFormatData: [
        { Class: 'sort', Type: 'Sort A to Z', Id: '1' },
        { Class: 'filter', Type: 'Filter', Id: '2' },
        { Class: 'clear', Type: 'Clear', Id: '3' }
      ],
      fields : { text: 'Type', iconCss: 'Class', value: 'Id' }
    }
  }
}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-inputs/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-dropdowns/styles/material.css";
.e-list-icon {
  line-height: 1.3;
  padding-right: 10px;
  text-indent: 5px;
}
.sort:before {
  content: '\e890';
  font-family: 'e-icons';
  font-size: 15px;
}
.filter:before {
  content: '\e7ee';
  font-family: 'e-icons';
  font-size: 15px;
  opacity: 0.78;
}
```

```

}
.clear:before {
  content: '\e7fc';
  font-family: 'e-icons';
  font-size: 15px;
}
</style>

```

{% previewsample "page.domainurl/code-snippet/drop-down-list/how-to/icons-cs1" %}

### Incremental search in Vue Drop down list component

DropDownList supports incremental search, by default. You can search the list item by focusing the DropDownList and typing the characters in it. The closely matched items are selected sequentially.

If the same key is searched once again, the next matched item is selected.

### Modify data in Vue Drop down list component

When binding the remote data source, by using the [actionComplete](#) event, you can modify the result data before passing it to DropDownList.

The following sample demonstrate how to modify the result data.

#### APP.VUE

```

<template>
  <div id="app">
    <div id='container' style="margin:50px auto 0; width:250px;">
      <br>
      <ejs-dropdownlist id='dropdownlist' placeholder='Select a customer'
      :actionComplete='actionComplete' :dataSource='dataSource' :query='query'
      :fields='fields'></ejs-dropdownlist>
    </div>
  </div>
</template>
<script>
import Vue from 'vue';
import { DropDownListPlugin } from "@syncfusion/ej2-vue-dropdowns";
Vue.use(DropDownListPlugin);
import { DataManager, Query, ODataV4Adaptor } from "@syncfusion/ej2-data";
export default {
  data () {
    return {
      query : new Query().from('Customers').select(['ContactName',
      'CustomerID']).take(6),
      dataSource : new DataManager({
        url: 'https://services.odata.org/V4/Northwind/Northwind.svc/',
        adaptor: new ODataV4Adaptor,
        crossDomain: true
      }),
      fields: { text: 'ContactName', value: 'CustomerID' }
    }
  },
  methods: {
    actionComplete: function (e) {
      // initially result contains 6 items
      console.log("Before modified the result: " + e.result.length);
    }
  }
}

```

```

        // remove first 2 items from result.
        e.result.splice(0, 2);
        // now displays the result count is 4.
        console.log("After modified the result: " + e.result.length);
    }
}
}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-inputs/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-
dropdowns/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/drop-down-list/how-to/modify-cs1" %}

### Multiple cascading in Vue Drop down list component

The following example demonstrate about how to preselect the list items in multiple cascading DropDownList.

#### APP.VUE

```

<template>
  <div id="app">
    <div id='container' style="margin:50px auto 0; width:250px;">
      <br>
      <ejs-dropdownlist id='countries' :dataSource='countryData'
      :index='countryindex' :fields='countryfields' :change='onCountryChange'
      placeholder='Select a country'></ejs-dropdownlist>
      <div class="padding-top">
        <ejs-dropdownlist id='states' :dataSource='stateData'
        :index='stateindex' :fields='statefields' :stateenabled='stateenabled'
        :change='onStateChange' placeholder='Select a state'></ejs-dropdownlist>
      </div>
      <div class="padding-top">
        <ejs-dropdownlist id='cities' :dataSource='cityData'
        :index='cityindex' :enabled='cityenabled' :fields='cityfields'
        placeholder='Select a city'></ejs-dropdownlist>
      </div>
    </div>
  </div>
</template>
<script>
import Vue from 'vue';
import { DropDownListPlugin } from "@syncfusion/ej2-vue-dropdowns";
Vue.use(DropDownListPlugin);
import { Query } from '@syncfusion/ej2-data';
export default {
  data () {
    return {
      countryData: [
        { CountryName: 'United States', CountryId: '1' },
        { CountryName: 'Australia', CountryId: '2' }
      ],
      stateData: [

```

```

        { StateName: 'New York', CountryId: '1', StateId: '101' },
        { StateName: 'Virginia ', CountryId: '1', StateId: '102' },
        { StateName: 'Tasmania ', CountryId: '2', StateId: '105' }
    ],
    cityData : [
        { CityName: 'Albany', StateId: '101', CityId: 201 },
        { CityName: 'Beacon ', StateId: '101', CityId: 202 },
        { CityName: 'Emporia', StateId: '102', CityId: 206 },
        { CityName: 'Hampton ', StateId: '102', CityId: 205 },
        { CityName: 'Hobart', StateId: '105', CityId: 213 },
        { CityName: 'Launceston ', StateId: '105', CityId: 214 }
    ]
    countryfields : { value: 'CountryId', text: 'CountryName' }
    statefields : { value: 'StateId', text: 'StateName' },
    cityfields : { text: 'CityName', value: 'CityId' },
    countryindex : 1,
    stateindex : 0,
    cityindex : 0,
    cityenabled : false,
    stateenabled : false,
    }
},
methods: {
    onCountryChange: function(e) {
        var countryObj =
document.getElementById('countries').ej2_instances[0];
        var stateObj =
document.getElementById('states').ej2_instances[0];
        var cityObj =
document.getElementById('cities').ej2_instances[0];
        //Query the data source based on country DropDownList selected
        value
        stateObj.query = new Query().where('CountryId', 'equal',
countryObj.value);
        // enable the state DropDownList
        stateObj.enabled = true;
        //clear the existing selection.
        stateObj.text = null;
        // bind the property changes to state DropDownList
        stateObj.dataBind();
        //clear the existing selection in city DropDownList
        cityObj.text = null;
        //disabe the city DropDownList
        cityObj.enabled = false;
        //bind the property cahnges to City DropDownList
        cityObj.dataBind();
    },
    onStateChange: function(e) {
        var stateObj =
document.getElementById('states').ej2_instances[0];
        var cityObj =
document.getElementById('cities').ej2_instances[0];
        // Query the data source based on state DropDownList selected
        value
        cityObj.query = new Query().where('StateId', 'equal',
stateObj.value);
        // enable the city DropDownList

```

```

        cityObj.enabled = true;
        //clear the existing selection
        cityObj.text = null;
        // bind the property change to city DropDownList
        cityObj.dataBind();
    }
}
}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-inputs/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-
dropdowns/styles/material.css";
    #container .padding-top {
        padding-top: 35px;
    }
</style>

```

{% previewsample "page.domainurl/code-snippet/drop-down-list/how-to/preselect-cs1" %}

### Remote data bind in Vue Drop down list component

Before component rendering, you can get the total items count by using [actionComplete](#) &#160;event with its result arguments.

After rendering this component, you can get the total items count by using [getItems](#) method.

The following example demonstrate how to get the total items count.

### APP.VUE

```

<template>
  <div id="app">
    <div id='container' style="margin:50px auto 0; width:250px;">
      <br>
      <ejs-dropdownlist id='dropdownlist' ref='dropdown'
placeholder='Select a customer' :actionComplete='actionComplete'
:dataSource='dataSource' :query='query' :fields='fields'></ejs-dropdownlist>
    </div>
    <div style='padding: 50px 0'>
      <button id='first' class="e-control e-btn" v-on:click="getItems"> Get
items</button>
    </div>
    <p id='event'> </p>
  </div>
</template>
<script>
import Vue from 'vue';
import { DropDownListPlugin } from "@syncfusion/ej2-vue-dropdowns";
Vue.use(DropDownListPlugin);
import { DataManager, Query, ODataV4Adaptor } from "@syncfusion/ej2-data";
export default {
  data () {
    return {
      query : new Query().from('Customers').select(['ContactName',
'CustomerID']).take(6),
      dataSource : new DataManager({

```

```

        url: 'https://services.odata.org/V4/Northwind/Northwind.svc/',
        adaptor: new ODataV4Adaptor,
        crossDomain: true
    )),
    fields: { text: 'ContactName', value: 'CustomerID' }
}
},
methods: {
    actionComplete: function (e) {
        // initially result contains 6 items
        console.log("Before modified the result: " + e.result.length);
        // remove first 2 items from result.
        e.result.splice(0, 2);
        // now displays the result count is 4.
        console.log("After modified the result: " + e.result.length);
    },
    getItem: function(e) {
        // get items count using getItem method
        console.log("Total items count: " +
this.$refs.dropdown.getItem().length);
        var element = document.createElement('p');
        element.innerText = "Total items count: " +
this.$refs.dropdown.getItem().length;
        document.getElementById('event').append(element);
    }
}
}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-inputs/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-
dropdowns/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/drop-down-list/how-to/getItems-cs1" %}

### Remove item in Vue Drop down list component

The following example demonstrate about how to remove an item from DropDownList.

#### APP.VUE

```

<template>
  <div id="app">
    <div id='container' style="margin:0 auto; width:250px;">
      <br>
      <ejs-dropdownlist id='dropdownlist' ref="dropdownObj"
:dataSource='sportsData' :fields='fields' placeholder='Select a game'></ejs-
dropdownlist>
    </div>
    <div style='padding: 50px 0'>
      <button id='first' class="e-control e-btn" v-on:click="remove"> Remove
0th item</button>
    </div>
  </div>
</template>

```

```

<script>
import Vue from 'vue';
import { DropDownListPlugin } from "@syncfusion/ej2-vue-dropdowns";
Vue.use(DropDownListPlugin);
export default {
  data () {
    return {
      sportsData: [
        { Id: 'game1', Game: 'Badminton' },
        { Id: 'game2', Game: 'Football' },
        { Id: 'game3', Game: 'Tennis' }
      ],
      fields : { text: 'Game', value: 'Id' }
    }
  },
  methods: {
    remove: function (event) {
      if(this.$refs.dropdownObj.getItems().length > 1) {
        this.$refs.dropdownObj.getItems()[0].remove();
        this.$refs.dropdownObj.dataSource.splice(0, 1);
      } else {
        this.sportsData = [];
      }
    }
  }
}
</script>
<style>
@import "../../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../../node_modules/@syncfusion/ej2-inputs/styles/material.css";
@import "../../node_modules/@syncfusion/ej2-vue-
dropdowns/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/drop-down-list/how-to/valuechange-cs2" %}

[Search on filtering in Vue Drop down list component](#)

The following example demonstrates about how to set limit the search result on filtering.

#### APP.VUE

```

<template>
  <div id="app">
    <div id='container' style="margin:20px auto 0; width:250px;">
      <br>
      <ejs-dropdownlist id='dropdownlist' placeholder='Select a customer'
        popupHeight='250px' sortOrder="Ascending" :query='query'
        :allowFiltering='allowFiltering' :filtering='filtering'
        :dataSource='dataSource' :fields='fields'></ejs-dropdownlist>
    </div>
  </div>
</template>
<script>
import Vue from 'vue';
import { DropDownListPlugin } from "@syncfusion/ej2-vue-dropdowns";
Vue.use(DropDownListPlugin);

```



```

import { DataManager, Query, ODataV4Adaptor, Predicate } from "@syncfusion/ej2-data";
export default {
  data () {
    return {
      dataSource : new DataManager({
        url:
        'https://services.odata.org/V4/Northwind/Northwind.svc/Customers',
        adaptor: new ODataV4Adaptor,
        crossDomain: true
      }),
      query : new Query().select(['ContactName', 'CustomerID']).take(7),
      fields : { text: 'ContactName', value: 'CustomerID' },
      allowFiltering: true,
    }
  },
  methods: {
    filtering: function(e) {
      var searchData = new DataManager({
        url:
        'https://services.odata.org/V4/Northwind/Northwind.svc/Customers',
        adaptor: new ODataV4Adaptor,
        crossDomain: true
      });
      // set limit as 4 to search result
      var query = new Query().select(['ContactName',
      'CustomerID']).take(4);
      query = (e.text !== '') ? query.where('ContactName',
      'startswith', e.text, true) : query;
      e.updateData(searchData, query);
    }
  }
}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-inputs/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-
dropdowns/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/drop-down-list/how-to/limit-cs1" %}

### Tooltip in Vue Drop down list component

You can achieve this behavior by using **ej2-tooltip** component. When the mouse hover on the DropDownList option that tooltip display some details related to hovered list item.

### APP.VUE

```

<template>
  <div id="app">
    <div id='container' style="margin:50px auto 0; width:250px;">
      <br>
      <ejs-dropdownlist id='Countries' ref='ddlObj'
      :created='onDropdownCreate' :dataSource='countryData' :fields='fields'
      :close='onClose' placeholder='Select a country'>

```

```

        </ejs-dropdownlist>
    </div>
</div>
</template>
<script>
import Vue from 'vue';
import { DropDownListPlugin } from "@syncfusion/ej2-vue-dropdowns";
import { TooltipPlugin } from "@syncfusion/ej2-vue-popups";
import { ButtonPlugin } from "@syncfusion/ej2-vue-buttons";
Vue.use(TooltipPlugin);
Vue.use(ButtonPlugin);
Vue.use(DropDownListPlugin);
import { Tooltip } from "@syncfusion/ej2-vue-popups";
export default { data () {
    return {
        countryData: [
            { id: '1', text: 'Australia', content: 'National sports is Cricket' },
            { id: '2', text: 'Bhutan', content: 'National sports is Archery' },
            { id: '3', text: 'China', content: 'National sports is Table Tennis' },
            { id: '4', text: 'Cuba', content: 'National sports is Baseball' },
            { id: '5', text: 'India', content: 'National sports is Hockey' },
            { id: '6', text: 'Spain', content: 'National sports is Football' },
            { id: '7', text: 'United States', content: 'National sports is Baseball' }
        ],
        fields : { text: 'text', value: 'id' },
        tooltip: Tooltip
    },
    methods: {
        onClose: function(e) {
            this.tooltip.close();
        },
        onBeforeRender : function(args) {
            var result = this.$refs.ddlObj.dataSource;
            var i;
            for (i = 0; i < result.length; i++) {
                if (result[i].text === args.target.textContent) {
                    this.tooltip.content = result[i].content;
                    this.tooltip.dataBind();
                    break;
                }
            }
        },
        onDropdownCreate: function(args) {
            this.tooltip = new Tooltip({
                // default content of tooltip
                content: 'Loading...',
                // set target element to tooltip
                target: '.e-list-item',
                // set position of tooltip
                position: 'top center',
                // bind beforeRender event
                beforeRender: this.onBeforeRender
            });
        }
    }
}

```

```

        this.tooltip.appendTo('body');
    }
}
</script>
<style>
@import "../../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../../node_modules/@syncfusion/ej2-inputs/styles/material.css";
@import "../../node_modules/@syncfusion/ej2-popups/styles/material.css";
@import "../../node_modules/@syncfusion/ej2-vue-
dropdowns/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/drop-down-list/how-to/tooltip-cs1" %}

### Value change in Vue Drop down list component

You can check about whether value change happened by manual or programmatic by using [change](#) event argument that argument name is `isInteracted`.

The following example demonstrate, how to check whether value change happened by manual or programmatic.

#### APP.VUE

```

<template>
  <div id="app">
    <div id='container' style="margin:0 auto; width:250px;">
      <br>
      <ejs-dropdownlist id='dropdownlist' ref='dropdown'
      :dataSource='sportsData' :fields='fields' :change='onChange'
      placeholder='Select a game'></ejs-dropdownlist>
    </div>
    <div style='margin: 50px'>
      <button id='button' class="e-control e-btn" v-on:click="onClick"> Set
      value dynamically</button>
    </div>
    <p style='margin: 5px' id='event'> </p>
  </div>
</template>
<script>
import Vue from 'vue';
import { DropDownListPlugin } from "@syncfusion/ej2-vue-dropdowns";
Vue.use(DropDownListPlugin);
export default {
  data () {
    return {
      sportsData: [
        { Id: 'game1', Game: 'Badminton' },
        { Id: 'game2', Game: 'Football' },
        { Id: 'game3', Game: 'Tennis' }
      ],
      fields : { text: 'Game', value: 'Id' }
    }
  },
  methods: {
    onClick: function (event) {

```

```
        this.$refs.dropdown.ej2Instances.value = 'game3';
    },
    onChange : function(args) {
        let element = document.createElement('p');
        if (args.isInteracted) {
            element.innerText = 'Changes happened by Interaction';
        } else {
            element.innerText = 'Changes happened by programmatic';
        }
        document.getElementById('event').append(element);
    }
}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-inputs/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-
dropdowns/styles/material.css";
</style>
```

{% previewsample "page.domainurl/code-snippet/drop-down-list/how-to/change-cs1" %}

Value support in Vue Drop down list component

yes, value for each list items should be unique.

## Dropdown Tree

### Getting Started with the Vue Drop down tree Component in Vue 2

This article provides a step-by-step guide for setting up a Vue 2 project using [Vue-CLI](#) and integrating the Syncfusion Vue Drop down tree component using the [Composition API](#) / [Options API](#).

Prerequisites

[System requirements for Syncfusion Vue UI components](#)

Setting up the Vue 2 project

To generate a Vue 2 project using Vue-CLI, use the [vue create](#) command. Follow these steps to install Vue CLI and create a new project:

```
`bash
```

```
npm install -g @vue/cli
```

```
vue create quickstart
```

```
cd quickstart
```

```
npm run serve
```

```
,
```

or

```
`bash
```

```
yarn global add @vue/cli
```

```
vue create quickstart
```

```
cd quickstart
```

```
yarn run serve
```

```
,
```

When creating a new project, choose the option **Default ([Vue 2] babel, eslint)** from the menu.

```
E:\samples>vue create quickstart

Vue CLI v5.0.8
? Please pick a preset:
  Default ([Vue 3] babel, eslint)
> Default ([Vue 2] babel, eslint)
  Manually select features
```

Once the **quickstart** project is set up with default settings, proceed to add Syncfusion components to the project.

#### Add Syncfusion Vue packages

Syncfusion packages are available at [npmjs.com](https://www.npmjs.com). To use Vue components, install the required npm package.

This article uses the [Vue Drop down tree component](#) as an example. Install the **@syncfusion/ej2-vue-dropdowns** package by running the following command:

```
`bash
```

```
npm install @syncfusion/ej2-vue-dropdowns --save
```

```
,
```

```
or
```

```
`bash
```

```
yarn add @syncfusion/ej2-vue-dropdowns
```

```
,
```

#### Import Syncfusion CSS styles

You can import themes for the Syncfusion Vue component in various ways, such as using CSS or SASS styles from npm packages, CDN, [CRG](#) and [Theme Studio](#). Refer to [themes topic](#) to know more about built-in themes and different ways to refer to themes in a Vue project.

In this article, the **Material** theme is applied using CSS styles, which are available in installed packages. The necessary **Material** CSS styles for the Drop down tree component and its dependents were imported into the `<style>` section of **src/App.vue** file.

#### ~/SRC/APP.VUE

```
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-inputs/styles/material.css";
@import "../node_modules/@syncfusion/ej2-navigations/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-dropdowns/styles/material.css";
```

```
</style>
```

### Add Syncfusion Vue component

Follow the below steps to add the Vue Drop down tree component using **Composition API** or **Options API**:

1\ First, import and register the Drop down tree component in the **script** section of the **src/App.vue** file. If you are using the **Composition API**, you should add the **setup** attribute to the **script** tag to indicate that Vue will be using the **Composition API**.

#### **COMPOSITION API (~SRC/APP.VUE)**

```
<script setup>
import { DropDownTreeComponent as EjsDropdowntree } from "@syncfusion/ej2-vue-dropdowns";
</script>
```

#### **OPTIONS API (~SRC/APP.VUE)**

```
<script>
import { DropDownTreeComponent } from "@syncfusion/ej2-vue-dropdowns";
export default {
  components: {
    'ejs-dropdowntree': DropDownTreeComponent
  }
}
</script>
```

2\ In the **template** section, define the Drop down tree component with the **fields** property.

#### **~/SRC/APP.VUE**

```
<template>
<div id="app">
<div id='container' style="margin:50px auto 0; width:250px;">
<br>
<ejs-dropdowntree id='dropdowntree' :fields='fields'></ejs-dropdowntree>
</div>
</div>
</template>
```

### Binding data source

The Dropdown Tree component can load the data either from local data sources or remote data services. This can be done using the **dataSource** property that is a member of the **fields** property. The **dataSource** property supports array of JavaScript objects and DataManager. Here, an array of JSON values is passed to the Dropdown Tree component.

#### **COMPOSITION API (~SRC/APP.VUE)**

```
<template>
<div id="app">
<div id='container' style="margin:50px auto 0; width:250px;">
<br>
```

```

<ejs-dropdowntree id='dropdowntree' :fields='fields'></ejs-dropdowntree>
</div>
</div>
</template>
<script setup>
import { DropDownTreeComponent as EjsDropdowntree } from "@syncfusion/ej2-vue-dropdowns";
var data = [
{
  nodeId: '01', nodeText: 'Music',
  nodeChild: [
    { nodeId: '01-01', nodeText: 'Gouttes.mp3' }
  ]
},
{
  nodeId: '02', nodeText: 'Videos', expanded: true,
  nodeChild: [
    { nodeId: '02-01', nodeText: 'Naturals.mp4' },
    { nodeId: '02-02', nodeText: 'Wild.mpeg' },
  ]
},
{
  nodeId: '03', nodeText: 'Documents',
  nodeChild: [
    { nodeId: '03-01', nodeText: 'Environment Pollution.docx' },
    { nodeId: '03-02', nodeText: 'Global Water, Sanitation, & Hygiene.docx' },
    { nodeId: '03-03', nodeText: 'Global Warming.ppt' },
    { nodeId: '03-04', nodeText: 'Social Network.pdf' },
    { nodeId: '03-05', nodeText: 'Youth Empowerment.pdf' },
  ]
}
];
const fields = { dataSource: data, value: 'nodeId', text: 'nodeText', child: 'nodeChild' };
</script>

```

### OPTIONS API (~SRC/APP.VUE)

```

<template>
<div id="app">
<div id='container' style="margin:50px auto 0; width:250px;">
<br>
<ejs-dropdowntree id='dropdowntree' :fields='fields'></ejs-dropdowntree>
</div>
</div>
</template>
<script>
import { DropDownTreeComponent } from "@syncfusion/ej2-vue-dropdowns";
var data = [
{
  nodeId: '01', nodeText: 'Music',
  nodeChild: [
    { nodeId: '01-01', nodeText: 'Gouttes.mp3' }
  ]
},
{
  nodeId: '02', nodeText: 'Videos', expanded: true,

```

```

nodeChild: [
  { nodeId: '02-01', nodeText: 'Naturals.mp4' },
  { nodeId: '02-02', nodeText: 'Wild.mpeg' },
],
},
{
  nodeId: '03', nodeText: 'Documents',
  nodeChild: [
    { nodeId: '03-01', nodeText: 'Environment Pollution.docx' },
    { nodeId: '03-02', nodeText: 'Global Water, Sanitation, & Hygiene.docx' },
    { nodeId: '03-03', nodeText: 'Global Warming.ppt' },
    { nodeId: '03-04', nodeText: 'Social Network.pdf' },
    { nodeId: '03-05', nodeText: 'Youth Empowerment.pdf' },
  ]
}
];
export default {
  components: {
    'ejs-dropdowntree': DropDownTreeComponent
  },
  data () {
    return {
      fields: { dataSource: data, value: 'nodeId', text: 'nodeText', child:
        'nodeChild' }
    }
  }
}
</script>

```

Here is the summarized code for the above steps in the **src/App.vue** file:

#### **COMPOSITION API (~SRC/APP.VUE)**

```

<template>
  <div id="app">
    <div id='container' style="margin:50px auto 0; width:250px;">
      <br>
      <ejs-dropdowntree id='dropdowntree' :fields='fields'></ejs-
dropdowntree>
    </div>
  </div>
</template>
<script setup>
import { DropDownTreeComponent as EjsDropdowntree } from "@syncfusion/ej2-
vue-dropdowns";
var data = [
  {
    nodeId: '01', nodeText: 'Music',
    nodeChild: [
      { nodeId: '01-01', nodeText: 'Gouttes.mp3' }
    ]
  },
  {
    nodeId: '02', nodeText: 'Videos', expanded: true,
    nodeChild: [
      { nodeId: '02-01', nodeText: 'Naturals.mp4' },
      { nodeId: '02-02', nodeText: 'Wild.mpeg' },

```



```

    ]
  },
  {
    nodeId: '03', nodeText: 'Documents',
    nodeChild: [
      { nodeId: '03-01', nodeText: 'Environment Pollution.docx' },
      { nodeId: '03-02', nodeText: 'Global Water, Sanitation, &
Hygiene.docx' },
      { nodeId: '03-03', nodeText: 'Global Warming.ppt' },
      { nodeId: '03-04', nodeText: 'Social Network.pdf' },
      { nodeId: '03-05', nodeText: 'Youth Empowerment.pdf' },
    ]
  }
];
const fields = { dataSource: data, value: 'nodeId', text: 'nodeText', child:
'nodeChild' };
</script>
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-inputs/styles/material.css";
@import "../node_modules/@syncfusion/ej2-navigations/styles/material.css";
@import "../node_modules/@syncfusion/ej2-buttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-popups/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-dropdowns/styles/material.css";
</style>

```

### OPTIONS API (~SRC/APP.VUE)

```

<template>
  <div id="app">
    <div id='container' style="margin:50px auto 0; width:250px;">
      <br>
      <ejs-dropdowntree id='dropdowntree' :fields='fields'></ejs-
dropdowntree>
    </div>
  </div>
</template>
<script>
import { DropDownTreeComponent } from "@syncfusion/ej2-vue-dropdowns";
var data = [
  {
    nodeId: '01', nodeText: 'Music',
    nodeChild: [
      { nodeId: '01-01', nodeText: 'Gouttes.mp3' }
    ]
  },
  {
    nodeId: '02', nodeText: 'Videos', expanded: true,
    nodeChild: [
      { nodeId: '02-01', nodeText: 'Naturals.mp4' },
      { nodeId: '02-02', nodeText: 'Wild.mpeg' },
    ]
  },
  {
    nodeId: '03', nodeText: 'Documents',
    nodeChild: [
      { nodeId: '03-01', nodeText: 'Environment Pollution.docx' },

```

```

        { nodeId: '03-02', nodeText: 'Global Water, Sanitation, &
Hygiene.docx' },
        { nodeId: '03-03', nodeText: 'Global Warming.ppt' },
        { nodeId: '03-04', nodeText: 'Social Network.pdf' },
        { nodeId: '03-05', nodeText: 'Youth Empowerment.pdf' },
      ]
    }
  });
  export default {
    components: {
      'ejs-dropdowntree': DropDownTreeComponent
    },
    data () {
      return {
        fields: { dataSource: data, value: 'nodeId', text: 'nodeText', child:
'nodeChild' }
      }
    }
  }
</script>
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-inputs/styles/material.css";
@import "../node_modules/@syncfusion/ej2-navigations/styles/material.css";
@import "../node_modules/@syncfusion/ej2-buttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-popups/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-dropdowns/styles/material.css";
</style>

```

### Run the project

To run the project, use the following command:

```
`bash
```

```
npm run serve
```

```
,
```

or

```
`bash
```

```
yarn run serve
```

```
,
```

```
{% previewsample "page.domainurl/code-snippet/drop-down-tree/getting-started/getting-started-cs1"
%}
```

### Getting Started with the Vue Dropdown Tree Component in Vue 3

This article provides a step-by-step guide for setting up a [Vite](#) project with a JavaScript environment and integrating the Syncfusion Vue Dropdown Tree component using the [Composition API](#) / [Options API](#).

The **Composition API** is a new feature introduced in Vue.js 3 that provides an alternative way to organize and reuse component logic. It allows developers to write components as functions that use smaller, reusable functions called composition functions to manage their properties and behavior.

The **Options API** is the traditional way of writing Vue.js components, where the component logic is organized into a series of options that define the component's properties and behavior. These options include data, methods, computed properties, watchers, lifecycle hooks, and more.

#### Prerequisites

##### [System requirements for Syncfusion Vue UI components](#)

#### Set up the Vite project

A recommended approach for beginning with Vue is to scaffold a project using [Vite](#). To create a new Vite project, use one of the commands that are specific to either NPM or Yarn.

```
`bash
```

```
npm create vite@latest
```

```
`
```

or

```
`bash
```

```
yarn create vite
```

```
`
```

Using one of the above commands will lead you to set up additional configurations for the project as below:

1. Define the project name: We can specify the name of the project directly. Let's specify the name of the project as **my-project** for this article.

```
`bash
```

```
? Project name: » my-project
```

```
`
```

2. Select **Vue** as the framework. It will create a Vue 3 project.

```
`bash
```

```
? Select a framework: » - Use arrow-keys. Return to submit.
```

```
Vanilla
```

```
Vue
```

```
React
```

```
Preact
```

```
Lit
```

```
Svelte
```

```
Others
```

```
`
```

3. Choose **JavaScript** as the framework variant to build this Vite project using JavaScript and Vue.

```
`bash
```

? Select a variant: » - Use arrow-keys. Return to submit.

JavaScript

TypeScript

Customize with create-vue ↗

Nuxt ↗

,

4. Upon completing the aforementioned steps to create the `my-project`, run the following command to install its dependencies:

```
`bash
```

```
cd my-project
```

```
npm install
```

,

or

```
`bash
```

```
cd my-project
```

```
yarn install
```

,

Now that `my-project` is ready to run with default settings, let's add Syncfusion components to the project.

#### Add Syncfusion Vue packages

Syncfusion Vue component packages are available at [npmjs.com](https://www.npmjs.com). To use Syncfusion Vue components in the project, install the corresponding npm package.

This article uses the [Vue Dropdown Tree component](#) as an example. To use the Vue Dropdown Tree component in the project, the `@syncfusion/ej2-vue-navigations` package needs to be installed using the following command:

```
`bash
```

```
npm install @syncfusion/ej2-vue-dropdowns --save
```

,

or

```
`bash
```

```
yarn add @syncfusion/ej2-vue-dropdowns
```

,

#### Import Syncfusion CSS styles

You can import themes for the Syncfusion Vue component in various ways, such as using CSS or SASS styles from npm packages, CDN, [CRG](#) and [Theme Studio](#). Refer to [themes topic](#) to know more about built-in themes and different ways to refer to themes in a Vue project.

In this article, **Material** theme is applied using CSS styles, which are available in installed packages. The necessary **Material** CSS styles for the Dropdown Tree component and its dependents were imported into the `<style>` section of **src/App.vue** file.

#### ~/SRC/APP.VUE

```
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-inputs/styles/material.css";
@import "../node_modules/@syncfusion/ej2-navigations/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-dropdowns/styles/material.css";
</style>
```

#### Add Syncfusion Vue component

Follow the below steps to add the Vue Dropdown Tree component using **Composition API** or **Options API**:

1.First, import and register the Dropdown Tree component in the **script** section of the **src/App.vue** file. If you are using the **Composition API**, you should add the **setup** attribute to the **script** tag to indicate that Vue will be using the Composition API.

#### COMPOSITION API (~/SRC/APP.VUE)

```
<script setup>
import { DropDownTreeComponent as EjsDropdowntree } from "@syncfusion/ej2-vue-dropdowns";
</script>
```

#### OPTIONS API (~/SRC/APP.VUE)

```
<script>
import { DropDownTreeComponent } from "@syncfusion/ej2-vue-dropdowns";
//Component registration
export default {
  name: "App",
  components: {
    "ejs-dropdowntree": DropDownTreeComponent
  }
}
</script>
```

2.Add the component definition in template section.

#### ~/SRC/APP.VUE

```
<template>
<ejs-dropdowntree id='dropdowntree' :fields='fields'></ejs-dropdowntree>
</template>
```

Here is the summarized code for the above steps in the **src/App.vue** file:

#### COMPOSITION API (~/SRC/APP.VUE)

```
<template>
```

```

<ejs-dropdowntree id='dropdowntree' :fields='fields'></ejs-dropdowntree>
</template>
<script setup>
import { DropDownTreeComponent as EjsDropdowntree } from "@syncfusion/ej2-vue-dropdowns";
const data = [
  {
    nodeId: '01', nodeText: 'Music',
    nodeChild: [
      { nodeId: '01-01', nodeText: 'Gouttes.mp3' }
    ]
  },
  {
    nodeId: '02', nodeText: 'Videos', expanded: true,
    nodeChild: [
      { nodeId: '02-01', nodeText: 'Naturals.mp4' },
      { nodeId: '02-02', nodeText: 'Wild.mpeg' },
    ]
  },
  {
    nodeId: '03', nodeText: 'Documents',
    nodeChild: [
      { nodeId: '03-01', nodeText: 'Environment Pollution.docx' },
      { nodeId: '03-02', nodeText: 'Global Water, Sanitation, & Hygiene.docx' },
      { nodeId: '03-03', nodeText: 'Global Warming.ppt' },
      { nodeId: '03-04', nodeText: 'Social Network.pdf' },
      { nodeId: '03-05', nodeText: 'Youth Empowerment.pdf' },
    ]
  }
];
const fields = { dataSource: data, value: 'nodeId', text: 'nodeText', child: 'nodeChild' };
</script>
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-inputs/styles/material.css";
@import "../node_modules/@syncfusion/ej2-navigations/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-dropdowns/styles/material.css";
</style>

```

### OPTIONS API (~SRC/APP.VUE)

```

<template>
<ejs-dropdowntree id='dropdowntree' :fields='fields'></ejs-dropdowntree>
</template>
<script>
import { DropDownTreeComponent } from "@syncfusion/ej2-vue-dropdowns";
const data = [
  {
    nodeId: '01', nodeText: 'Music',
    nodeChild: [
      { nodeId: '01-01', nodeText: 'Gouttes.mp3' }
    ]
  },
  {
    nodeId: '02', nodeText: 'Videos', expanded: true,
    nodeChild: [

```

```
{ nodeId: '02-01', nodeText: 'Naturals.mp4' },
{ nodeId: '02-02', nodeText: 'Wild.mpeg' },
],
},
{
  nodeId: '03', nodeText: 'Documents',
  nodeChild: [
    { nodeId: '03-01', nodeText: 'Environment Pollution.docx' },
    { nodeId: '03-02', nodeText: 'Global Water, Sanitation, & Hygiene.docx' },
    { nodeId: '03-03', nodeText: 'Global Warming.ppt' },
    { nodeId: '03-04', nodeText: 'Social Network.pdf' },
    { nodeId: '03-05', nodeText: 'Youth Empowerment.pdf' },
  ]
}
];
//Component registration
export default {
  name: "App",
  components: {
    "ejs-dropdowntree": DropDownTreeComponent
  },
  data() {
    return {
      fields: { dataSource: data, value: 'nodeId', text: 'nodeText', child:
        'nodeChild' }
    };
  }
}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-inputs/styles/material.css";
@import "../node_modules/@syncfusion/ej2-navigations/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-dropdowns/styles/material.css";
</style>
```

### Run the project

To run the project, use the following command:

```
`bash
```

```
npm run dev
```

```
`
```

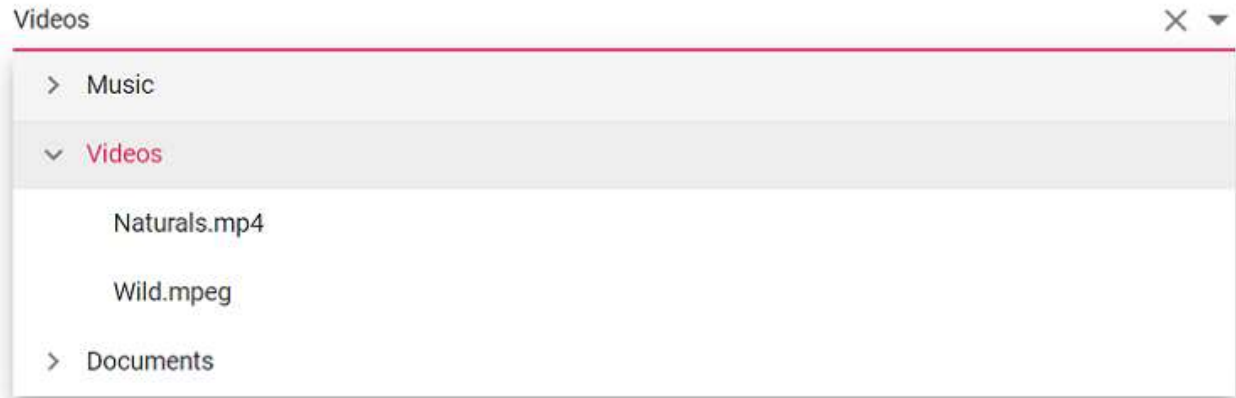
or

```
`bash
```

```
yarn run dev
```

```
`
```

The output will appear as follows:



For migrating from Vue 2 to Vue 3, refer to the [migration](#) documentation.

See also

- [Getting Started with Vue UI Components using Composition API and TypeScript](#)
- [Getting Started with Vue UI Components using Options API and TypeScript](#)

### Data binding in Vue Drop down tree component

The Dropdown Tree component provides an option to load the data either from local data sources or from remote data services. This can be done through `dataSource` property that is a member of the `fields` property. The `dataSource` property supports array of JavaScript objects and `DataManager`. It also supports different kinds of data services such as OData, OData V4, Web API, URL, and JSON with the help of `DataManager` adaptors.

Dropdown Tree has `load on demand` (Lazy load) option. It reduces the bandwidth size when consuming the huge data. By default, the `loadOnDemand` is set to false. By enabling this property, it loads first level items initially, and when parent item is expanded, loads the child items based on the `parentValue/child` member.

#### Local data

To bind local data to the Dropdown Tree, you can assign a JavaScript object array to the `dataSource` property.

The Dropdown Tree component requires three fields (Value, text, and parentValue) to render local data source. When mapper fields are not specified, it takes the default values as the mapping fields. Local data source can also be provided as an instance of the `DataManager`. It supports two kinds of local data binding methods.

- Hierarchical data
- Self-referential data

#### Hierarchical data

Dropdown Tree can be populated with the hierarchical data source that contains nested array of JSON objects. You can directly map the hierarchical data and the field members with corresponding key values from the hierarchical data to the `fields` property.



In the following example, **code**, **name**, and **countries** columns from the hierarchical data have been mapped to **value**, **text**, and **child** fields, respectively.

#### APP.VUE

```
<template>
  <div id="app">
    <div id='container' style="margin:50px auto 0; width:250px;">
      <br>
      <ejs-dropdowntree id='dropdowntree' :fields='fields'></ejs-
dropdowntree>
    </div>
  </div>
</template>
<script>
import Vue from 'vue';
import { DropDownTreePlugin } from "@syncfusion/ej2-vue-dropdowns";
Vue.use(DropDownTreePlugin);
var data = [
  {
    code: 'AF', name: 'Africa', countries: [
      { code: 'NGA', name: 'Nigeria' },
      { code: 'EGY', name: 'Egypt' },
      { code: 'ZAF', name: 'South Africa' }
    ]
  },
  {
    code: 'AS', name: 'Asia', expanded: true, countries: [
      { code: 'CHN', name: 'China' },
      { code: 'IND', name: 'India', selected: true },
      { code: 'JPN', name: 'Japan' }
    ]
  },
  {
    code: 'EU', name: 'Europe', countries: [
      { code: 'DNK', name: 'Denmark' },
      { code: 'FIN', name: 'Finland' },
      { code: 'AUT', name: 'Austria' }
    ]
  },
  {
    code: 'NA', name: 'North America', countries: [
      { code: 'USA', name: 'United States of America' },
      { code: 'CUB', name: 'Cuba' },
      { code: 'MEX', name: 'Mexico' }
    ]
  },
  {
    code: 'SA', name: 'South America', countries: [
      { code: 'BRA', name: 'Brazil' },
      { code: 'COL', name: 'Colombia' },
      { code: 'ARG', name: 'Argentina' }
    ]
  },
  {
    code: 'OC', name: 'Oceania', countries: [
      { code: 'AUS', name: 'Australia' },

```

```

        { code: 'NZL', name: 'New Zealand' },
        { code: 'WSM', name: 'Samoa' }
      ],
    },
    {
      code: 'AN', name: 'Antarctica', countries: [
        { code: 'BVT', name: 'Bouvet Island' },
        { code: 'ATF', name: 'French Southern Lands' }
      ]
    },
  ],
];
export default {
  data () {
    return {
      fields: { dataSource: data, value: 'code', text: 'name', child:
'countries' }
    }
  }
}
</script>
<style>
@import "../../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../../node_modules/@syncfusion/ej2-inputs/styles/material.css";
@import "../../node_modules/@syncfusion/ej2-
navigations/styles/material.css";
@import "../../node_modules/@syncfusion/ej2-buttons/styles/material.css";
@import "../../node_modules/@syncfusion/ej2-popups/styles/material.css";
@import "../../node_modules/@syncfusion/ej2-vue-
dropdowns/styles/material.css";
</style>

```

```
{% previewsample "page.domainurl/code-snippet/drop-down-tree/data-binding/hierarchial-data-cs1"
%}
```

### Self-referential data

Dropdown Tree can be populated from the self-referential data structure that contains array of JSON objects with `parentValue` mapping.

You can directly assign the self-referential data and map all the field members with corresponding key values from self-referential data to the `fields` property.

To render the root level items, specify the `parentValue` as null or no need to specify the `parentValue` in the `dataSource`.

In the following example, **id**, **pid**, **hasChild**, and **name** columns from self-referential data have been mapped to **value**, **parentValue**, **hasChildren**, and **text** fields, respectively.

### APP.VUE

```

<template>
  <div id="app">
    <div id='container' style="margin:50px auto 0; width:250px;">
      <br>
      <ejs-dropdowntree id='dropdowntree' :fields='fields'></ejs-
dropdowntree>
    </div>
  </div>

```

```

</div>
</template>
<script>
import Vue from 'vue';
import { DropDownTreePlugin } from "@syncfusion/ej2-vue-dropdowns";
Vue.use(DropDownTreePlugin);
var data = [
  { id: 1, name: 'Discover Music', hasChild: true, expanded: true },
  { id: 2, pid: 1, name: 'Hot Singles' },
  { id: 3, pid: 1, name: 'Rising Artists' },
  { id: 4, pid: 1, name: 'Live Music' },
  { id: 5, pid: 1, name: 'Best of 2017 So Far' },
  { id: 7, name: 'Sales and Events', hasChild: true },
  { id: 8, pid: 7, name: '100 Albums' },
  { id: 9, pid: 7, name: 'Hip-Hop and R&B Sale' },
  { id: 10, pid: 7, name: 'CD Deals' },
  { id: 11, name: 'Categories', hasChild: true },
  { id: 12, pid: 11, name: 'Songs' },
  { id: 13, pid: 11, name: 'Bestselling Albums' },
  { id: 14, pid: 11, name: 'New Releases' },
  { id: 15, pid: 11, name: 'Bestselling Songs' },
  { id: 16, name: 'MP3 Albums', hasChild: true },
  { id: 17, pid: 16, name: 'Rock' },
  { id: 18, pid: 16, name: 'Gospel' },
  { id: 19, pid: 16, name: 'Latin Music' },
  { id: 20, pid: 16, name: 'Jazz' },
  { id: 21, name: 'More in Music', hasChild: true },
  { id: 22, pid: 21, name: 'Music Trade-In' },
  { id: 23, pid: 21, name: 'Redeem a Gift Card' },
  { id: 24, pid: 21, name: 'Band T-Shirts' },
];
export default {
  data () {
    return {
      fields: { dataSource: data, value: 'id', text: 'name',
parentValue: "pid", hasChildren: 'hasChild' }
    }
  }
}
</script>
<style>
@import "../../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../../node_modules/@syncfusion/ej2-inputs/styles/material.css";
@import "../../node_modules/@syncfusion/ej2-
navigations/styles/material.css";
@import "../../node_modules/@syncfusion/ej2-buttons/styles/material.css";
@import "../../node_modules/@syncfusion/ej2-popups/styles/material.css";
@import "../../node_modules/@syncfusion/ej2-vue-
dropdowns/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/drop-down-tree/data-binding/self-referential-data-cs1" %}

### Remote data

Dropdown Tree can also be populated from a remote data service with the help of the **DataManager** component and **Query** property.

It supports different kinds of data services such as OData, OData V4, Web API, URL, and JSON with the help of **DataManager** adaptors.

You can assign service data as an instance of **DataManager** to the **dataSource**. To interact with remote data source, you must provide the endpoint **url**.

The **DataManager** that acts as an interface between the service endpoint and the Dropdown Tree requires the following information to interact with service endpoint properly.

- **DataManager->url**: Defines the service endpoint to fetch data.
- **DataManager->adaptor**: Defines the adaptor option. By default, **ODataAdaptor** is used for remote binding.

Adaptor is responsible for processing response and request from/to the service endpoint. The **@syncfusion/ej2-data** package provides some pre-defined adaptors designed to interact with service endpoints. They are,

- **UrlAdaptor**: Used to interact with remote services. This is the base adaptor for all remote based adaptors.
- **ODataAdaptor**: Used to interact with OData endpoints.
- **ODataV4Adaptor**: Used to interact with OData V4 endpoints.
- **WebApiAdaptor**: Used to interact with Web API created under OData standards.
- **WebMethodAdaptor**: Used to interact with web methods.

In the following example, **ODataV4Adaptor** is used to fetch data from the remote services. The **EmployeeID**, **FirstName**, and **EmployeeID** columns from the Employees table have been mapped to **value**, **text**, and **hasChildren** fields respectively for first level items.

The **OrderID**, **EmployeeID**, and **ShipName** columns from the orders table have been mapped to **value**, **parentValue**, and **text** fields respectively for second level items.

### APP.VUE

```
<template>
  <div id="app">
    <div id='container' style="margin:50px auto 0; width:250px;">
      <br>
      <ejs-dropdowntree id='dropdowntree' :fields='fields'></ejs-
dropdowntree>
    </div>
  </div>
</template>
<script>
import Vue from 'vue';
import { DropDownTreePlugin } from "@syncfusion/ej2-vue-dropdowns";
Vue.use(DropDownTreePlugin);
import { DataManager, Query, ODataV4Adaptor } from "@syncfusion/ej2-data";
export default {
```

```

data () {
  var remotedata = new DataManager({
    url: 'https://services.odata.org/V4/Northwind/Northwind.svc',
    adaptor: new ODataV4Adaptor,
    crossDomain: true,
  });
  return {
    fields: { dataSource: remotedata, query: new
Query().from('Employees').select('EmployeeID,FirstName,Title').take(5),
value: 'EmployeeID', text: 'FirstName', hasChildren: 'EmployeeID',
child: { dataSource: remotedata, query: new
Query().from('Orders').select('OrderID,EmployeeID,ShipName').take(5), value:
'OrderID', parentValue: 'EmployeeID', text: 'ShipName'}
}
}
}
}
</script>
<style>
@import "../../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../../node_modules/@syncfusion/ej2-inputs/styles/material.css";
@import "../../node_modules/@syncfusion/ej2-
navigations/styles/material.css";
@import "../../node_modules/@syncfusion/ej2-buttons/styles/material.css";
@import "../../node_modules/@syncfusion/ej2-popups/styles/material.css";
@import "../../node_modules/@syncfusion/ej2-vue-
dropdowns/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/drop-down-tree/data-binding/remote-data-cs1" %}

### Prevent Node selection

You can prevent the selection of individual tree node by using the [selectable](#) property. The tree node selection is not allowed while disable this property.

The [selectable](#) property is disabled and the selection is prevented for parent nodes in below sample.

### APP.VUE

```

<template>
  <div id="app">
    <div id='container' style="margin:50px auto 0; width:250px;">
      <br>
      <ejs-dropdowntree id='dropdowntree' :fields='fields'></ejs-
dropdowntree>
    </div>
  </div>
</template>
<script>
import Vue from 'vue';
import { DropDownTreePlugin } from "@syncfusion/ej2-vue-dropdowns";
Vue.use(DropDownTreePlugin);
var data = [
  { id: 1, name: 'Discover Music', hasChild: true, expanded: true,
selectable: false },
  { id: 2, pid: 1, name: 'Hot Singles' },

```

```

    { id: 3, pid: 1, name: 'Rising Artists' },
    { id: 4, pid: 1, name: 'Live Music' },
    { id: 6, pid: 1, name: 'Best of 2017 So Far' },
    { id: 7, name: 'Sales and Events', hasChild: true, selectable: false
  },
  { id: 8, pid: 7, name: '100 Albums' },
  { id: 9, pid: 7, name: 'Hip-Hop and R&B Sale' },
  { id: 10, pid: 7, name: 'CD Deals' },
  { id: 11, name: 'Categories', hasChild: true, selectable: false },
  { id: 12, pid: 11, name: 'Songs' },
  { id: 13, pid: 11, name: 'Bestselling Albums' },
  { id: 14, pid: 11, name: 'New Releases' },
  { id: 15, pid: 11, name: 'Bestselling Songs' },
  { id: 16, name: 'MP3 Albums', hasChild: true, selectable: false },
  { id: 17, pid: 16, name: 'Rock' },
  { id: 18, pid: 16, name: 'Gospel' },
  { id: 19, pid: 16, name: 'Latin Music' },
  { id: 20, pid: 16, name: 'Jazz' },
  { id: 21, name: 'More in Music', hasChild: true, selectable: false
},
  { id: 22, pid: 21, name: 'Music Trade-In' },
  { id: 23, pid: 21, name: 'Redeem a Gift Card' },
  { id: 24, pid: 21, name: 'Band T-Shirts' },
];
export default {
  data () {
    return {
      fields: { dataSource: data, value: 'id', text: 'name',
parentValue:"pid", hasChildren: 'hasChild', selectable: 'selectable' }
    }
  }
}
</script>
<style>
@import "../../../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../../../node_modules/@syncfusion/ej2-inputs/styles/material.css";
@import "../../../node_modules/@syncfusion/ej2-
navigations/styles/material.css";
@import "../../../node_modules/@syncfusion/ej2-buttons/styles/material.css";
@import "../../../node_modules/@syncfusion/ej2-popups/styles/material.css";
@import "../../../node_modules/@syncfusion/ej2-vue-
dropdowns/styles/material.css";
</style>

```

```
{% previewsample "page.domainurl/code-snippet/drop-down-tree/data-binding/prevent-node-cs1" %}
```

## Templates in Vue Drop down tree component

The Dropdown Tree provides support to customize each list item, header, and footer elements. It uses the Essential JS 2 Template engine to compile and render the elements properly.

### Item template

The content of each list item within the Dropdown Tree can be customized with the help of [itemTemplate](#) property.

In the following sample, the Dropdown Tree list items are customized with employee information such as **name** and **job** using the **itemTemplate** property.

The template expression should be provided inside the `{% raw %}{{...}}{% endraw %}` interpolation syntax.

### APP.VUE

```
<template>
  <div id="app">
    <div id='container' style="margin:50px auto 0; width:250px;"><br>
      <ejs-dropdowntree id='dropdowntree' :fields='fields'
placeholder='Select an employee':itemTemplate='itemTemplate' ></ejs-
dropdowntree>
    </div>
  </div>
</template>
<script>
import Vue from 'vue';
import { DropDownTreePlugin } from "@syncfusion/ej2-vue-dropdowns";
Vue.use(DropDownTreePlugin);
var data = [
  { "id": 1, "name": "Steven Buchanan", "job": "General Manager",
"hasChild": true, "expanded": true },
  { "id": 2, "pid": 1, "name": "Laura Callahan", "job": "Product Manager",
"hasChild": true },
  { "id": 3, "pid": 2, "name": "Andrew Fuller", "job": "Team Lead",
"hasChild": true },
  { "id": 4, "pid": 3, "name": "Anne Dodsworth", "job": "Developer" },
  { "id": 10, "pid": 3, "name": "Lilly", "job": "Developer", "status":
"online" },
  { "id": 5, "pid": 1, "name": "Nancy Davolio", "job": "Product Manager",
"hasChild": true },
  { "id": 6, "pid": 5, "name": "Michael Suyama", "job": "Team Lead",
"hasChild": true },
  { "id": 7, "pid": 6, "name": "Robert King", "job": "Developer" },
  { "id": 11, "pid": 6, "name": "Mary", "job": "Developer" },
  { "id": 9, "pid": 1, "name": "Janet Leverling", "job": "HR" }
];
var itemVue = Vue.component("itemTemplate", {
  template: `<span><span class="ename">{{data.name}} - </span><span
class="ejjob">{{data.job}}</span></span>`,
  data() {return {data: {}};
  }
});
export default {
  data () {
    return {
      itemTemplate : function() {
        return {template: itemVue};
      },
      fields: { dataSource: data, value: 'id', text: 'name',
parentValue:"pid", hasChildren: 'hasChild' },
    }
  }
}
</script>
```

```

<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-inputs/styles/material.css";
@import "../node_modules/@syncfusion/ej2-navigations/styles/material.css";
@import "../node_modules/@syncfusion/ej2-buttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-popups/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-dropdowns/styles/material.css";
.ejob{
    opacity: .60;
}
</style>

```

{% previewsample "page.domainurl/code-snippet/drop-down-tree/templates/item-template-cs1" %}

### Header template

The header element is shown statically at the top of the popup list items within the Dropdown Tree. A custom element can be placed as a header element using the [headerTemplate](#) property.

In the following sample, the header is customized with the custom element.

### APP.VUE

```

<template>
  <div id="app">
    <div id='container' style="margin:50px auto 0; width:250px;"><br>
      <ejs-dropdowntree id='dropdowntree' :fields='fields'
placeholder='Select an employee':headerTemplate='headerTemplate' ></ejs-
dropdowntree>
    </div>
  </div>
</template>
<script>
import Vue from 'vue';
import { DropDownTreePlugin } from "@syncfusion/ej2-vue-dropdowns";
Vue.use(DropDownTreePlugin);
var data = [
  { "id": 1, "name": "Steven Buchanan", "job": "General Manager",
"hasChild": true, "expanded": true },
  { "id": 2, "pid": 1, "name": "Laura Callahan", "job": "Product Manager",
"hasChild": true },
  { "id": 3, "pid": 2, "name": "Andrew Fuller", "job": "Team Lead",
"hasChild": true },
  { "id": 4, "pid": 3, "name": "Anne Dodsworth", "job": "Developer" },
  { "id": 10, "pid": 3, "name": "Lilly", "job": "Developer", "status":
"online" },
  { "id": 5, "pid": 1, "name": "Nancy Davolio", "job": "Product Manager",
"hasChild": true },
  { "id": 6, "pid": 5, "name": "Michael Suyama", "job": "Team Lead",
"hasChild": true },
  { "id": 7, "pid": 6, "name": "Robert King", "job": "Developer " },
  { "id": 11, "pid": 6, "name": "Mary", "job": "Developer " },
  { "id": 9, "pid": 1, "name": "Janet Leverling", "job": "HR" }
];
var headerVue = Vue.component("headerTemplate", {
  template: `<span class='head'>Employee List</span>`,
  data() {

```



```

    return {
      data: {}
    };
  }
});
export default {
  data () {
    return {
      fields: { dataSource: data, value: 'id', text: 'name',
parentValue: "pid", hasChildren: 'hasChild' },
      headerTemplate : function(e) {
        return {
          template: headerVue
        };
      }
    }
  }
}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-inputs/styles/material.css";
@import "../node_modules/@syncfusion/ej2-navigations/styles/material.css";
@import "../node_modules/@syncfusion/ej2-buttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-popups/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-dropdowns/styles/material.css";
.head {
  height: 40px;
  font-size: 15px;
  font-weight: 600;
  border-bottom: 1px solid #e0e0e0;
}
</style>

```

{% previewsample "page.domainurl/code-snippet/drop-down-tree/templates/header-template-cs1" %}

### Footer template

The Dropdown Tree has options to show a footer element at the bottom of the list items in the popup list. Here, you can place any custom element as a footer element using the [footerTemplate](#) property.

In the following sample, the footer element displays the total number of employees present in the Dropdown Tree.

### APP.VUE

```

<template>
  <div id="app">
    <div id='container' style="margin:50px auto 0; width:250px;"><br>
      <ejs-dropdowntree id='dropdowntree' :fields='fields'
placeholder='Select an employee':footerTemplate='footerTemplate' ></ejs-
dropdowntree>
    </div>
  </div>
</template>
<script>
import Vue from 'vue';

```

```

import { DropDownTreePlugin } from "@syncfusion/ej2-vue-dropdowns";
Vue.use(DropDownTreePlugin);
var data = [
  { "id": 1, "name": "Steven Buchanan", "job": "General Manager",
    "hasChild": true, "expanded": true },
  { "id": 2, "pid": 1, "name": "Laura Callahan", "job": "Product Manager",
    "hasChild": true },
  { "id": 3, "pid": 2, "name": "Andrew Fuller", "job": "Team Lead",
    "hasChild": true },
  { "id": 4, "pid": 3, "name": "Anne Dodsworth", "job": "Developer" },
  { "id": 10, "pid": 3, "name": "Lilly", "job": "Developer", "status":
    "online" },
  { "id": 5, "pid": 1, "name": "Nancy Davolio", "job": "Product Manager",
    "hasChild": true },
  { "id": 6, "pid": 5, "name": "Michael Suyama", "job": "Team Lead",
    "hasChild": true },
  { "id": 7, "pid": 6, "name": "Robert King", "job": "Developer" },
  { "id": 11, "pid": 6, "name": "Mary", "job": "Developer" },
  { "id": 9, "pid": 1, "name": "Janet Leverling", "job": "HR" }
];
var footerVue = Vue.component("footerTemplate", {
  template: `<span class='foot'> Total number of employees: 10</span>`,
  data() {
    return {
      data: {}
    };
  }
});
export default {
  data () {
    return {
      fields: { dataSource: data, value: 'id', text: 'name',
        parentValue: "pid", hasChildren: 'hasChild' },
      footerTemplate : function(e) {
        return {
          template: footerVue
        }
      }
    }
  }
}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-inputs/styles/material.css";
@import "../node_modules/@syncfusion/ej2-navigations/styles/material.css";
@import "../node_modules/@syncfusion/ej2-buttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-popups/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-dropdowns/styles/material.css";
.foot {
  text-indent: 1.2em;
  display: block;
  font-size: 14px;
  line-height: 40px;
  border-top: 1px solid #e0e0e0;
  font-weight: bold;
}

```

```
.custom .e-ddt-footer{
  border-top: 1px solid #e0e0e0;
}
</style>
```

{% previewsample "page.domainurl/code-snippet/drop-down-tree/templates/footer-template-cs1" %}

### No records template

The Dropdown Tree is supports to display custom design in the popup list content using the [noRecordsTemplate](#) property when no matches found on search.

In the following sample, popup list content displays the notification of no data available.

### APP.VUE

```
<template>
  <div id="app">
    <div id='container' style="margin:50px auto 0; width:250px;"><br>
      <ejs-dropdowntree id='dropdowntree' :fields='fields'
placeholder='Select an employee':noRecordsTemplate='noRecordsTemplate'
></ejs-dropdowntree>
    </div>
  </div>
</template>
<script>
import Vue from 'vue';
import { DropDownTreePlugin } from "@syncfusion/ej2-vue-dropdowns";
Vue.use(DropDownTreePlugin);
var data = [ ];
var noRecordsVue = Vue.component("noRecordsTemplate", {
  template: `<span class='norecord'> NO DATA AVAILABLE</span>`,
  data() {
    return {
      data: {}
    };
  }
});
export default {
  data () {
    return {
      noRecordsTemplate : function (e) {
        return {
          template: noRecordsVue
        }
      },
      fields: { dataSource: data, value: 'id', text: 'name',
parentValue:"pid", hasChildren: 'hasChild' },
    }
  }
}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-inputs/styles/material.css";
@import "../node_modules/@syncfusion/ej2-navigations/styles/material.css";
@import "../node_modules/@syncfusion/ej2-buttons/styles/material.css";
```

```
@import "../node_modules/@syncfusion/ej2-popups/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-dropdowns/styles/material.css";
</style>
```

```
{% previewsample "page.domainurl/code-snippet/drop-down-tree/templates/noRecords-template-cs1"
%}
```

### Action failure template

The Dropdown Tree provides an option to custom design the popup list content using [actionFailureTemplate](#) property, when the data fetch request fails at the remote server.

In the following sample, when the data fetch request fails, the Dropdown Tree displays the notification.

#### APP.VUE

```
<template>
  <div id="app">
    <div id='container' style="margin:50px auto 0; width:250px;">
      <br>
      <ejs-dropdownntree id='dropdownntree' :fields='fields'
placeholder='Select an
employee':actionFailureTemplate='actionFailureTemplate' ></ejs-dropdownntree>
    </div>
  </div>
</template>
<script>
import Vue from 'vue';
import { DropDownTreePlugin } from "@syncfusion/ej2-vue-dropdowns";
import { DataManager, Query, ODataV4Adaptor } from "@syncfusion/ej2-data";
Vue.use(DropDownTreePlugin);
var remoteData = new DataManager({
  url: 'https://services.odata.org/V4/Northwind/Northwind.svs',
  adaptor: new ODataV4Adaptor,
  crossDomain: true,
});
var failureVue = Vue.component("failureTemplate", {
  template: `<span class='action-failure'> Data fetch request fails</span>`,
  data() {
    return {
      data: {}
    };
  }
});
export default Vue.extend({
  data: function() {
    return {
      actionFailureTemplate : function(e) {
        return {
          template: failureVue
        }
      },
      fields: { dataSource: remoteData, value: 'id', text: 'name',
parentValue:"pid", hasChildren: 'hasChild' },
      height: '200px'
    };
  },
});
```

```
});
</script>
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-inputs/styles/material.css";
@import "../node_modules/@syncfusion/ej2-navigations/styles/material.css";
@import "../node_modules/@syncfusion/ej2-buttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-popups/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-dropdowns/styles/material.css";
</style>
```

```
{% previewsample "page.domainurl/code-snippet/drop-down-tree/templates/actionFailure-template-cs1" %}
```

### Custom template to show selected items in input

In Dropdown Tree, while selecting more than one items via checkbox or multi selection support, all the selected items will be displayed in the input. Instead of displaying all the selected item text, the custom template can be displayed by setting the the [mode](#) property as **Custom** and [customTemplate](#) property.

When the **mode** property is set as **Custom**, the Dropdown Tree displays the default template value (**\${value.length} item(s) selected**) like **1 item(s) selected** or **2 item(s) selected**. The default template can be customized by setting **customTemplate** property.

In the following sample, the Dropdown Tree is rendered with default value of the **customTemplate** property like **"1 item(s) selected or 2 item(s) selected"**.

### APP.VUE

```
<template>
  <div id="app">
    <div id='container' style="margin:50px auto 0; width:250px;"><br>
      <ejs-dropdowntree id='dropdowntree' ref="ddtreeObj" :fields='fields'
:customTemplate='customTemplate' :showCheckBox='true'
:treeSettings='treeSettings' mode= 'Custom' :placeholder='waterMark'
:popupHeight='height'></ejs-dropdowntree>
    </div>
  </div>
</template>
<script>
import Vue from "vue";
import { DropDownTreePlugin } from "@syncfusion/ej2-vue-dropdowns";
Vue.use(DropDownTreePlugin);
var data = [
  { "id": 1, "name": "Steven Buchanan", "job": "General Manager",
"hasChild": true, "expanded": true },
  { "id": 2, "pid": 1, "name": "Laura Callahan", "job": "Product Manager",
"hasChild": true },
  { "id": 3, "pid": 2, "name": "Andrew Fuller", "job": "Team Lead",
"hasChild": true },
  { "id": 4, "pid": 3, "name": "Anne Dodsworth", "job": "Developer" },
  { "id": 10, "pid": 3, "name": "Lilly", "job": "Developer", "status":
"online" },
  { "id": 5, "pid": 1, "name": "Nancy Davolio", "job": "Product Manager",
"hasChild": true },
  { "id": 6, "pid": 5, "name": "Michael Suyama", "job": "Team Lead",
"hasChild": true },
```

```

    { "id": 7, "pid": 6, "name": "Robert King", "job": "Developer " },
    { "id": 11, "pid": 6, "name": "Mary", "job": "Developer " },
    { "id": 9, "pid": 1, "name": "Janet Leverling", "job": "HR" }
  ];
  export default Vue.extend({
    data: function() {
      var temp = this;
      return {
        fields: { dataSource: data, value: 'id', text: 'name', parentValue:
        'pid', hasChildren: 'hasChild' },
        height: '200px',
        waterMark: 'Select an employee',
        treeSettings: { autoCheck: true },
        customTemplate: '${value.length} item(s) selected'
      };
    }
  });
</script>
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-inputs/styles/material.css";
@import "../node_modules/@syncfusion/ej2-navigations/styles/material.css";
@import "../node_modules/@syncfusion/ej2-buttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-popups/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-dropdowns/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/drop-down-tree/templates/custom-template-mode-cs1" %}

In the following sample, the Dropdown Tree is rendered with custom value of the **customTemplate** property like **Selected items count: 2**.

#### APP.VUE

```

<template>
  <div id="app">
    <div id='container' style="margin:50px auto 0; width:250px;"><br>
      <ejs-dropdowntree id='dropdowntree' ref="ddtreeObj" :fields='fields'
      :customTemplate='customTemplate' :showCheckBox='true'
      :treeSettings='treeSettings' mode= 'Custom' :placeholder='waterMark'
      :popupHeight='height'></ejs-dropdowntree>
    </div>
  </div>
</template>
<script>
import Vue from "vue";
import { DropDownTreePlugin } from "@syncfusion/ej2-vue-dropdowns";
Vue.use(DropDownTreePlugin);
var data = [
  { "id": 1, "name": "Steven Buchanan", "job": "General Manager",
  "hasChild": true, "expanded": true },
  { "id": 2, "pid": 1, "name": "Laura Callahan", "job": "Product Manager",
  "hasChild": true },
  { "id": 3, "pid": 2, "name": "Andrew Fuller", "job": "Team Lead",
  "hasChild": true },

```

```

    { "id": 4, "pid": 3, "name": "Anne Dodsworth", "job": "Developer" },
    { "id": 10, "pid": 3, "name": "Lilly", "job": "Developer", "status":
"online" },
    { "id": 5, "pid": 1, "name": "Nancy Davolio", "job": "Product Manager",
"hasChild": true },
    { "id": 6, "pid": 5, "name": "Michael Suyama", "job": "Team Lead",
"hasChild": true },
    { "id": 7, "pid": 6, "name": "Robert King", "job": "Developer " },
    { "id": 11, "pid": 6, "name": "Mary", "job": "Developer " },
    { "id": 9, "pid": 1, "name": "Janet Leverling", "job": "HR" }
  ];
  export default Vue.extend({
    data: function() {
      var temp = this;
      return {
        fields: { dataSource: data, value: 'id', text: 'name', parentValue:
'pid', hasChildren: 'hasChild' },
        height: '200px',
        waterMark: 'Select an employee',
        treeSettings: { autoCheck: true },
        customTemplate: 'Selected item(s) count: ${value.length}'
      };
    }
  });
</script>
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-inputs/styles/material.css";
@import "../node_modules/@syncfusion/ej2-navigations/styles/material.css";
@import "../node_modules/@syncfusion/ej2-buttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-popups/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-dropdowns/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/drop-down-tree/templates/custom-template-cs1" %}

### Checkbox in Vue Drop down tree component

The Dropdown Tree component allows you to check more than one item from the tree without affecting the UI's appearance by enabling the `showCheckBox` property. When this property is enabled, checkbox appears before each item text in the popup.

In the following example, the `showCheckBox` property is enabled.

#### APP.VUE

```

<template>
  <div id="app">
    <div id='container' style="margin:50px auto 0; width:250px;">
      <br>
      <ejs-dropdowntree id='dropdowntree' :fields='fields'
:showCheckBox='true'></ejs-dropdowntree>
    </div>
  </div>
</template>
<script>
import Vue from 'vue';

```

```

import { DropDownTreePlugin } from "@syncfusion/ej2-vue-dropdowns";
Vue.use(DropDownTreePlugin);
var data = [
  { id: 1, name: 'Discover Music', hasChild: true, expanded: true },
  { id: 2, pid: 1, name: 'Hot Singles' },
  { id: 3, pid: 1, name: 'Rising Artists' },
  { id: 4, pid: 1, name: 'Live Music' },
  { id: 6, pid: 1, name: 'Best of 2017 So Far' },
  { id: 7, name: 'Sales and Events', hasChild: true },
  { id: 8, pid: 7, name: '100 Albums - $5 Each' },
  { id: 9, pid: 7, name: 'Hip-Hop and R&B Sale' },
  { id: 10, pid: 7, name: 'CD Deals' },
  { id: 11, name: 'Categories', hasChild: true },
  { id: 12, pid: 11, name: 'Songs' },
  { id: 13, pid: 11, name: 'Bestselling Albums' },
  { id: 14, pid: 11, name: 'New Releases' },
  { id: 15, pid: 11, name: 'Bestselling Songs' },
  { id: 16, name: 'MP3 Albums', hasChild: true },
  { id: 17, pid: 16, name: 'Rock' },
  { id: 18, pid: 16, name: 'Gospel' },
  { id: 19, pid: 16, name: 'Latin Music' },
  { id: 20, pid: 16, name: 'Jazz' },
  { id: 21, name: 'More in Music', hasChild: true },
  { id: 22, pid: 21, name: 'Music Trade-In' },
  { id: 23, pid: 21, name: 'Redeem a Gift Card' },
  { id: 24, pid: 21, name: 'Band T-Shirts' },
];
export default {
  data () {
    return {
      fields: { dataSource: data, value: 'id', text: 'name',
parentValue:"pid", hasChildren: 'hasChild' }
    }
  }
}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-inputs/styles/material.css";
@import "../node_modules/@syncfusion/ej2-
navigations/styles/material.css";
@import "../node_modules/@syncfusion/ej2-buttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-popups/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-
dropdowns/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/drop-down-tree/checkboxes-cs1" %}

### Auto Check

By default, the checkbox state of the parent and child items in the Dropdown Tree will not be dependent over each other. If you need dependent checked state, then enable the `autoCheck` property which is a member of `treeSettings` property.



- If one or more child items are not in the checked state, then the parent item will be in the intermediate state.
- If all the child items are checked, then the parent item will also be in the checked state.
- If a parent item is checked, then all the child items will also be changed to the checked state.

In the following example, the `autoCheck` property is enabled.

#### APP.VUE

```
<template>
  <div id="app">
    <div id='container' style="margin:50px auto 0; width:250px;">
      <br>
      <ejs-dropdowntree id='dropdowntree' :fields='fields'
:showCheckBox='true' :treeSettings='treeSettings'></ejs-dropdowntree>
    </div>
  </div>
</template>
<script>
import Vue from 'vue';
import { DropDownTreePlugin } from "@syncfusion/ej2-vue-dropdowns";
Vue.use(DropDownTreePlugin);
var data = [
  { id: 1, name: 'Discover Music', hasChild: true, expanded: true },
  { id: 2, pid: 1, name: 'Hot Singles' },
  { id: 3, pid: 1, name: 'Rising Artists' },
  { id: 4, pid: 1, name: 'Live Music' },
  { id: 6, pid: 1, name: 'Best of 2017 So Far' },
  { id: 7, name: 'Sales and Events', hasChild: true },
  { id: 8, pid: 7, name: '100 Albums - $5 Each' },
  { id: 9, pid: 7, name: 'Hip-Hop and R&B Sale' },
  { id: 10, pid: 7, name: 'CD Deals' },
  { id: 11, name: 'Categories', hasChild: true },
  { id: 12, pid: 11, name: 'Songs' },
  { id: 13, pid: 11, name: 'Bestselling Albums' },
  { id: 14, pid: 11, name: 'New Releases' },
  { id: 15, pid: 11, name: 'Bestselling Songs' },
  { id: 16, name: 'MP3 Albums', hasChild: true },
  { id: 17, pid: 16, name: 'Rock' },
  { id: 18, pid: 16, name: 'Gospel' },
  { id: 19, pid: 16, name: 'Latin Music' },
  { id: 20, pid: 16, name: 'Jazz' },
  { id: 21, name: 'More in Music', hasChild: true },
  { id: 22, pid: 21, name: 'Music Trade-In' },
  { id: 23, pid: 21, name: 'Redeem a Gift Card' },
  { id: 24, pid: 21, name: 'Band T-Shirts' },
];
export default {
  data () {
    return {
      fields: { dataSource: data, value: 'id', text: 'name',
parentValue:"pid", hasChildren: 'hasChild' },
      treeSettings: { autoCheck: true }
    }
  }
}
```

```

</script>
<style>
@import ".../node_modules/@syncfusion/ej2-base/styles/material.css";
@import ".../node_modules/@syncfusion/ej2-inputs/styles/material.css";
@import ".../node_modules/@syncfusion/ej2-
navigations/styles/material.css";
@import ".../node_modules/@syncfusion/ej2-buttons/styles/material.css";
@import ".../node_modules/@syncfusion/ej2-popups/styles/material.css";
@import ".../node_modules/@syncfusion/ej2-vue-
dropdowns/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/drop-down-tree/auto-check-cs1" %}

### Select All

The Dropdown Tree component has in-built support to select all the tree items using Select All options in the header.

When the `showSelectAll` property is set to true, a checkbox will be displayed in the popup header that allows you to select or deselect all the tree items in the popup.

By default, `Select All` and `unSelect All` text values will be showcased along with the checkbox in the popup header to indicate the action to be performed on checking or unchecking the checkbox. You can customize these name attributes by using `selectAllText` and `unSelectAllText` properties respectively.

### APP.VUE

```

<template>
  <div id="app">
    <div id='container' style="margin:50px auto 0; width:250px;">
      <br>
      <ejs-dropdowntree id='dropdowntree' :fields='fields'
      :showCheckBox='true' :showSelectAll='true' :selectAllText='selectText'
      :unSelectAllText='unSelectText'></ejs-dropdowntree>
    </div>
  </div>
</template>
<script>
import Vue from 'vue';
import { DropDownTreePlugin } from "@syncfusion/ej2-vue-dropdowns";
Vue.use(DropDownTreePlugin);
var data = [
  { id: 1, name: 'Discover Music', hasChild: true, expanded: true },
  { id: 2, pid: 1, name: 'Hot Singles' },
  { id: 3, pid: 1, name: 'Rising Artists' },
  { id: 4, pid: 1, name: 'Live Music' },
  { id: 6, pid: 1, name: 'Best of 2017 So Far' },
  { id: 7, name: 'Sales and Events', hasChild: true },
  { id: 8, pid: 7, name: '100 Albums - $5 Each' },
  { id: 9, pid: 7, name: 'Hip-Hop and R&B Sale' },
  { id: 10, pid: 7, name: 'CD Deals' },
  { id: 11, name: 'Categories', hasChild: true },
  { id: 12, pid: 11, name: 'Songs' },
  { id: 13, pid: 11, name: 'Bestselling Albums' },
  { id: 14, pid: 11, name: 'New Releases' },
  { id: 15, pid: 11, name: 'Bestselling Songs' },

```

```

    { id: 16, name: 'MP3 Albums', hasChild: true },
    { id: 17, pid: 16, name: 'Rock' },
    { id: 18, pid: 16, name: 'Gospel' },
    { id: 19, pid: 16, name: 'Latin Music' },
    { id: 20, pid: 16, name: 'Jazz' },
    { id: 21, name: 'More in Music', hasChild: true },
    { id: 22, pid: 21, name: 'Music Trade-In' },
    { id: 23, pid: 21, name: 'Redeem a Gift Card' },
    { id: 24, pid: 21, name: 'Band T-Shirts' },
  ];
  export default {
    data () {
      return {
        fields: { dataSource: data, value: 'id', text: 'name',
parentValue:"pid", hasChildren: 'hasChild' },
        selectText: 'Check All',
        unSelectText: 'UnCheck All'
      }
    }
  }
</script>
<style>
@import "../../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../../node_modules/@syncfusion/ej2-inputs/styles/material.css";
@import "../../node_modules/@syncfusion/ej2-
navigations/styles/material.css";
@import "../../node_modules/@syncfusion/ej2-buttons/styles/material.css";
@import "../../node_modules/@syncfusion/ej2-popups/styles/material.css";
@import "../../node_modules/@syncfusion/ej2-vue-
dropdowns/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/drop-down-tree/select-all-cs1" %}

### Localization in Vue Drop down tree component

The [Localization](#) library allows you to localize default text content of the Dropdown Tree and it can be localized to any culture by defining the texts and messages of the Dropdown Tree in the corresponding culture. The default locale of the Dropdown Tree is en (English). The following table represents the default texts and messages of the Dropdown Tree in en culture.

KEY	Text/Message
----	----
noRecordsTemplate	No records found
actionFailureTemplate	Request failed
overflowCountTemplate	+\$ {count} more..
totalCountTemplate	\$ {count} selected

### Accessibility in Vue Dropdown Tree component

The Dropdown Tree component followed the accessibility guidelines and standards, including [ADA, Section 508](#), [WCAG 2.2](#) standards, and [WCAG roles](#) that are commonly used to evaluate accessibility.

The accessibility compliance for the Dropdown Tree component is outlined below.

| Accessibility Criteria | Compatibility |

| -- | -- |

| [WCAG 2.2](#) Support |  |

| [Section 508](#) Support |  |

| Screen Reader Support |  |

| Right-To-Left Support |  |

| Color Contrast |  |

| Mobile Device Support |  |

| Keyboard Navigation Support |  |

| [Accessibility Checker](#) Validation |  |

| [Axe-core](#) Accessibility Validation |  |

<style>

.post .post-content img {

display: inline-block;

margin: 0.5em 0;

}

</style>

<div> - All features of the component meet the requirement.</div>

<div> - Some features of the component do not meet the requirement.</div>

<div> - The component does not meet the requirement.</div>

### WAI-ARIA attributes

The Dropdown Tree component followed the [WAI-ARIA](#) patterns to meet the accessibility. The following ARIA attributes are used in the Dropdown Tree component:

| Attributes | Purpose |

| --- | --- |

- | `role=checkbox` | All list items are contained within the element. |
- | `aria-disabled` | Indicates element is perceivable but disabled. |
- | `aria-owns` | This attribute contains the ID of the popup list to indicate popup as a child element. |
- | `aria-haspopup` | Indicates whether the Dropdown Tree input element has a popup list or not. |
- | `aria-expanded` | Indicates the state of the popup list for Dropdown Tree and the parent node's expansion status for TreeView. |
- | `aria-activedescendent` | This attribute holds the ID of the active list item to focus its descendant child element. |
- | `aria-labelledby` | This attribute points to the element(s) labeling the element it's applied to. |
- | `aria-describedby` | This attribute points to the element(s) describing the one it's set on. |
- | `role=tree` | All tree nodes are contained within the element. |
- | `role=treeitem` | Specifies the role of each tree node in a selectable TreeView and its containment within the tree. |
- | `role=group` | Specifies the role of each parent node container in the TreeView. |
- | `role=checkbox` | Indicates checkbox control along with treeitem element. |
- | `aria-multiselectable` | Indicates whether the TreeView enables multiple selection or not. |
- | `aria-selected` | Indicates the selected node. |
- | `aria-level` | Indicates the level of node in TreeView. |
- | `aria-checked` | Indicates the current checked state of TreeView checkbox. |
- | `aria-label` | Indicates the contextual message for the TreeView checkbox and Dropdown Tree. |
- | `aria-activedescendant` | Identifies the currently active element when focusing on the TreeView. |

### Keyboard interaction

The Dropdown Tree component followed the [keyboard interaction](#) guideline, making it easy for people who use assistive technologies (AT) and those who completely rely on keyboard navigation. The following keyboard shortcuts are supported by the Dropdown Tree component.

| Interaction Keys | Description |

|-----|-----|

| **Alt + Down** | Opens the popup. |

| **Alt + Up** | Closes the popup. |

| **Esc(Escape)** | Closes the popup when it is in an open state. |

| **Arrow Up** | Goes to the previous item in the popup. |

| **Arrow Down** | Goes to the next item in the popup. |

| **Arrow Right** | Expands the current item in the popup. |

| Arrow Left | Collapses the current item in the popup. |

| Home | Goes to the first item in the popup. |

| End | Goes to the last item in the popup. |

| Enter | Selects the focused item in the popup. |

| Space | Checks the current item in the popup. |

### Ensuring accessibility

The Dropdown Tree component's accessibility levels are ensured through an [accessibility-checker](#) and [axe-core](#) software tools during automated testing.

The accessibility compliance of the Dropdown Tree component is shown in the following sample. Open the [sample](#) in a new window to evaluate the accessibility of the Dropdown Tree component with accessibility tools.

```
{% previewsample "https://ej2.syncfusion.com/accessibility/drop-down-tree.html" %}
```

See also

- [Accessibility in Syncfusion Vue components](#)

## FileManager

### Getting Started with the Vue File manager Component in Vue 2

This article provides a step-by-step guide for setting up a Vue 2 project using [Vue-CLI](#) and integrating the Syncfusion Vue Filemanager component

#### Prerequisites

[System requirements for Syncfusion Vue UI components](#)

#### Dependencies

The following are the dependencies required to use the File Manager component in your application:

```
`js
|-- @syncfusion/ej2-vue-filemanager
|-- @syncfusion/ej2-vue-base
|-- @syncfusion/ej2-vue-grids
|-- @syncfusion/ej2-vue-navigations
|-- @syncfusion/ej2-vue-inputs
|-- @syncfusion/ej2-vue-lists
|-- @syncfusion/ej2-vue-popups
|-- @syncfusion/ej2-vue-buttons
|-- @syncfusion/ej2-base
|-- @syncfusion/ej2-data
|-- @syncfusion/ej2-grids
```

```
|-- @syncfusion/ej2-filemanager  
|-- @syncfusion/ej2-navigations  
|-- @syncfusion/ej2-inputs  
|-- @syncfusion/ej2-lists  
|-- @syncfusion/ej2-popups  
|-- @syncfusion/ej2-buttons  
`
```

### Setting up the Vue 2 project

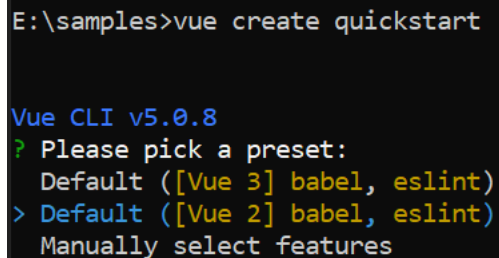
To generate a Vue 2 project using Vue-CLI, use the [vue create](#) command. Follow these steps to install Vue CLI and create a new project:

```
`bash  
npm install -g @vue/cli  
vue create quickstart  
cd quickstart  
npm run serve  
`
```

or

```
`bash  
yarn global add @vue/cli  
vue create quickstart  
cd quickstart  
yarn run serve  
`
```

When creating a new project, choose the option **Default ([Vue 2] babel, eslint)** from the menu.



```
E:\samples>vue create quickstart  
  
Vue CLI v5.0.8  
? Please pick a preset:  
  Default ([Vue 3] babel, eslint)  
> Default ([Vue 2] babel, eslint)  
  Manually select features
```

Once the **quickstart** project is set up with default settings, proceed to add Syncfusion components to the project.

### Add Syncfusion Vue packages

Syncfusion packages are available at [npmjs.com](https://www.npmjs.com). To use Vue components, install the required npm package.

This article uses the [Vue File manager component](#) as an example. Install the `@syncfusion/ej2-vue-filemanager` package by running the following command:

```
`bash
```

```
npm install @syncfusion/ej2-vue-filemanager --save
```

```
,
```

or

```
`bash
```

```
yarn add @syncfusion/ej2-vue-filemanager
```

```
,
```

### Import Syncfusion CSS styles

You can import themes for the Syncfusion Vue component in various ways, such as using CSS or SASS styles from npm packages, CDN, [CRG](#) and [Theme Studio](#). Refer to [themes topic](#) to know more about built-in themes and different ways to refer to themes in a Vue project.

In this article, the `Material` theme is applied using CSS styles, which are available in installed packages. The necessary `Material` CSS styles for the File manager component and its dependents were imported into the `<style>` section of `src/App.vue` file.

#### ~/SRC/APP.VUE

```
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-icons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-inputs/styles/material.css";
@import "../node_modules/@syncfusion/ej2-popups/styles/material.css";
@import "../node_modules/@syncfusion/ej2-buttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-splitbuttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-navigations/styles/material.css";
@import "../node_modules/@syncfusion/ej2-layouts/styles/material.css";
@import "../node_modules/@syncfusion/ej2-grids/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-filemanager/styles/material.css";
</style>
```

Note: If you want to refer the combined component styles, please make use of our [CRG](#) (Custom Resource Generator) in your application.

### Add Syncfusion Vue component

Follow the below steps to add the Vue File manager component:

1\ First, import and register the File manager component in the `script` section of the `src/App.vue` file.

#### ~/SRC/APP.VUE

```
<script>
```



```
import { FileManagerComponent } from "@syncfusion/ej2-vue-filemanager";
export default {
  components: {
    'ejs-filemanager': FileManagerComponent
  }
}
</script>
```

2\ In the **template** section, define the File manager component with [ajaxSettings](#) property.

#### ~/SRC/APP.VUE

```
<template>
<div id="app">
<ejs-filemanager id="file-manager" :ajaxSettings="ajaxSettings">
</ejs-filemanager>
</div>
</template>
```

3\ Declare the value for the **ajaxSettings** property in the **script** section.

#### ~/SRC/APP.VUE

```
<script>
data () {
  return {
    ajaxSettings:
    {
      url: "https://ej2-aspcore-
service.azurewebsites.net/api/FileManager/FileOperations"
    }
  }
}
</script>
```

Here is the summarized code for the above steps in the **src/App.vue** file:

#### ~/SRC/APP.VUE

```
<template>
  <div id="app">
    <ejs-filemanager id="file-manager" :ajaxSettings="ajaxSettings">
    </ejs-filemanager>
  </div>
</template>
<script>
import { FileManagerComponent } from "@syncfusion/ej2-vue-filemanager";
export default {
  components: {
    'ejs-filemanager': FileManagerComponent
  },
  data () {
    return {
      ajaxSettings:
      {

```

```

        url: "https://ej2-aspcore-
service.azurewebsites.net/api/FileManager/FileOperations"
    }
}
}
}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-icons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-inputs/styles/material.css";
@import "../node_modules/@syncfusion/ej2-popups/styles/material.css";
@import "../node_modules/@syncfusion/ej2-buttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-splitbuttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-navigations/styles/material.css";
@import "../node_modules/@syncfusion/ej2-layouts/styles/material.css";
@import "../node_modules/@syncfusion/ej2-grids/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-
filemanager/styles/material.css";
</style>

```

### Run the project

To run the project, use the following command:

```
`bash
```

```
npm run serve
```

```
,
```

or

```
`bash
```

```
yarn run serve
```

```
,
```

```
{% previewsample "page.domainurl/code-snippet/file-manager/getting-started-cs1" %}
```

### File Download support

To perform the download operation, initialize the `downloadUrl` property in a [ajaxSettings](#) of File Manager component.

#### ~/SRC/APP.VUE

```

<script>
import { FileManagerComponent } from "@syncfusion/ej2-vue-filemanager";
export default {
  components: {
    'ejs-filemanager': FileManagerComponent
  },
  data () {
    return {
      ajaxSettings:
      {
        url: "https://ej2-aspcore-
service.azurewebsites.net/api/FileManager/FileOperations",

```

```
downloadUrl: "https://ej2-aspcore-  
service.azurewebsites.net/api/FileManager/Download"  
}  
}  
}  
}  
</script>
```

### File Upload support

To perform the upload operation, initialize the `uploadUrl` property in a [ajaxSettings](#) of File Manager Component.

#### ~/SRC/APP.VUE

```
<script>  
import { FileManagerComponent } from "@syncfusion/ej2-vue-filemanager";  
export default {  
  components: {  
    'ejs-filemanager': FileManagerComponent  
  },  
  data () {  
    return {  
      ajaxSettings: {  
        url: "https://ej2-aspcore-  
service.azurewebsites.net/api/FileManager/FileOperations",  
        uploadUrl: "https://ej2-aspcore-  
service.azurewebsites.net/api/FileManager/Upload"  
      }  
    }  
  }  
}  
</script>
```

### Image Preview support

To perform the image preview support in the File Manager component, need to initialize the `getImageUrl` property in a [ajaxSettings](#) of File Manager component.

#### APP.VUE

```
<template>  
  <div id="app">  
    <ejs-filemanager id="file-manager" :ajaxSettings="ajaxSettings">  
    </ejs-filemanager>  
  </div>  
</template>  
<script>  
import { FileManagerComponent } from "@syncfusion/ej2-vue-filemanager";  
export default {  
  components: {  
    'ejs-filemanager': FileManagerComponent  
  },  
  data () {  
    return {  
      ajaxSettings:  

```

```

        {
            url: "https://ej2-aspcore-
service.azurewebsites.net/api/FileManager/FileOperations",
            getImageUrl: "https://ej2-aspcore-
service.azurewebsites.net/api/FileManager/GetImage"
        }
    }
}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-icons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-inputs/styles/material.css";
@import "../node_modules/@syncfusion/ej2-popups/styles/material.css";
@import "../node_modules/@syncfusion/ej2-buttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-splitbuttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-navigations/styles/material.css";
@import "../node_modules/@syncfusion/ej2-layouts/styles/material.css";
@import "../node_modules/@syncfusion/ej2-grids/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-
filemanager/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/file-manager/getting-started-cs2" %}

### Injecting feature modules

Basically, the file manager component contains large-icons view for displaying the files and folders, a breadcrumbbar for navigation and context menu for performing operations. However, these basic functionalities can be extended by using the additional feature modules like detailsview, toolbar, navigation pane, and context menu to change the layout and to simplify the navigation and file operations within the file system. The above modules can be injected using `provide`.

The following example shows you the File Manager with all feature modules.

### APP.VUE

```

<template>
<div id="app">
    <div class="wrapper">
        <ejs-filemanager id="overview_filemanager" :ajaxSettings="ajaxSettings"
:vue="view">
            </ejs-filemanager>
        </div>
    </div>
</template>
<script>
import { FileManagerComponent, DetailsView, NavigationPane, Toolbar } from
"@syncfusion/ej2-vue-filemanager";
export default {
    components: {
        'ejs-filemanager': FileManagerComponent
    },
    data () {
        return {
            ajaxSettings:

```

```

        {
            url: "https://ej2-aspcore-
service.azurewebsites.net/api/FileManager/FileOperations",
            getImageUrl: "https://ej2-aspcore-
service.azurewebsites.net/api/FileManager/GetImage",
            uploadUrl: "https://ej2-aspcore-
service.azurewebsites.net/api/FileManager/Upload",
            downloadUrl: "https://ej2-aspcore-
service.azurewebsites.net/api/FileManager/Download"
        },
        // Initial view of File Manager is set to details view
        view: "Details"
    },
    //Injecting additional modules in FileManager
    provide: {
        filemanager: [DetailsView, NavigationPane, Toolbar]
    }
}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-icons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-inputs/styles/material.css";
@import "../node_modules/@syncfusion/ej2-popups/styles/material.css";
@import "../node_modules/@syncfusion/ej2-buttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-splitbuttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-navigations/styles/material.css";
@import "../node_modules/@syncfusion/ej2-layouts/styles/material.css";
@import "../node_modules/@syncfusion/ej2-grids/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-
filemanager/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/file-manager/overview-cs1" %}

**Note:** The appearance of the File Manager can be customized by using [cssClass](#) property. This adds a css class to the root of the File Manager which can be used to add new styles or override existing styles to the File Manager.

### Switching initial view of the File Manager

The initial view of the File Manager can be changed to details or largeicons view with the help of [view](#) property. By default, the File Manager will be rendered in large icons view. When the File Manager is initially rendered, [created](#) will be triggered. This event can be utilized for performing operations once the File Manager has been successfully created.

### APP.VUE

```

<template>
    <div id="app">
        <ejs-filemanager id="file-manager" :view="view" :created="onCreate"
:ajaxSettings="ajaxSettings">
            </ejs-filemanager>
        </div>
    </template>
</script>

```

```

import { FileManagerComponent, DetailsView, NavigationPane, Toolbar } from
"@syncfusion/ej2-vue-filemanager";
export default {
  components: {
    'ejs-filemanager': FileManagerComponent
  },
  data () {
    return {
      ajaxSettings: {
        url: "https://ej2-aspcore-
service.azurewebsites.net/api/FileManager/FileOperations",
        getImageUrl: "https://ej2-aspcore-
service.azurewebsites.net/api/FileManager/GetImage",
        uploadUrl: "https://ej2-aspcore-
service.azurewebsites.net/api/FileManager/Upload",
        downloadUrl: "https://ej2-aspcore-
service.azurewebsites.net/api/FileManager/Download"
      },
      // Initial view of File Manager is set to details view
      view: "Details"
    };
  },
  provide: {
    filemanager: [DetailsView, NavigationPane, Toolbar]
  },
  methods: {
    // File Manager's created event function
    onCreate() {
      console.log("File Manager has been created successfully");
    }
  }
}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-icons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-inputs/styles/material.css";
@import "../node_modules/@syncfusion/ej2-popups/styles/material.css";
@import "../node_modules/@syncfusion/ej2-buttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-splitbuttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-navigations/styles/material.css";
@import "../node_modules/@syncfusion/ej2-layouts/styles/material.css";
@import "../node_modules/@syncfusion/ej2-grids/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-
filemanager/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/file-manager/view-cs1" %}

### Maintaining component state on page reload

The File Manager supports maintaining the component state on page reload. This can be achieved by enabling [enablePersistence](#) property which maintains the following,

- Previous view of the File Manager - [View](#)

- Previous path of the File Manager - [Path](#)
- Previous selected items of the File Manager - [SelectedItems](#)

For every operation in File Manager, ajax request will be sent to the server which then processes the request and sends back the response. When the ajax request is success, [success](#) event will be triggered and [failure](#) event will be triggered if the request gets failed.

#### APP.VUE

```
<template>
  <div id="app">
    <ejs-filemanager id="file-manager"
      :enablePersistence="enablePersistence" :ajaxSettings="ajaxSettings"
      :success="onAjaxSuccess" :failure="onAjaxFailure">
      </ejs-filemanager>
    </div>
  </template>
<script>
import { FileManagerComponent, DetailsView, NavigationPane, Toolbar } from
"@syncfusion/ej2-vue-filemanager";
export default {
  components: {
    'ejs-filemanager': FileManagerComponent
  },
  data () {
    return {
      ajaxSettings:
        {
          url: "https://ej2-aspcore-
service.azurewebsites.net/api/FileManager/FileOperations",
          getImageUrl: "https://ej2-aspcore-
service.azurewebsites.net/api/FileManager/GetImage",
          uploadUrl: "https://ej2-aspcore-
service.azurewebsites.net/api/FileManager/Upload",
          downloadUrl: "https://ej2-aspcore-
service.azurewebsites.net/api/FileManager/Download"
        },
      enablePersistence: true
    };
  },
  provide: {
    filemanager: [DetailsView, NavigationPane, Toolbar]
  },
  methods: {
    // File Manager's file onSuccess function
    onAjaxSuccess() {
      console.log("Ajax request successful");
    },
    // File Manager's file onError function
    onAjaxFailure() {
      console.log("Ajax request has failed");
    }
  }
}
</script>
<style>
```

```
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-icons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-inputs/styles/material.css";
@import "../node_modules/@syncfusion/ej2-popups/styles/material.css";
@import "../node_modules/@syncfusion/ej2-buttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-splitbuttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-navigations/styles/material.css";
@import "../node_modules/@syncfusion/ej2-layouts/styles/material.css";
@import "../node_modules/@syncfusion/ej2-grids/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-filemanager/styles/material.css";
</style>
```

{% previewsample "page.domainurl/code-snippet/file-manager/persistence-cs1" %}

**Note:** The files of the current folder opened in the File manager can be refreshed programatically by calling [refreshFiles](#) method.

Rendering component in right-to-left direction

It is possible to render the File Manager in right-to-left direction by setting the [enableRtl](#) API to true.

#### APP.VUE

```
<template>
  <div id="app">
    <ejs-filemanager id="file-manager" :enableRtl="enableRtl"
    :ajaxSettings="ajaxSettings">
      </ejs-filemanager>
    </div>
  </template>
<script>
import { FileManagerComponent, DetailsView, NavigationPane, Toolbar } from
"@syncfusion/ej2-vue-filemanager";
export default {
  components: {
    'ejs-filemanager': FileManagerComponent
  },
  data () {
    return {
      ajaxSettings: {
        url: "https://ej2-aspcore-
service.azurewebsites.net/api/FileManager/FileOperations",
        getImageUrl: "https://ej2-aspcore-
service.azurewebsites.net/api/FileManager/GetImage",
        uploadUrl: "https://ej2-aspcore-
service.azurewebsites.net/api/FileManager/Upload",
        downloadUrl: "https://ej2-aspcore-
service.azurewebsites.net/api/FileManager/Download"
      },
      enableRtl: true
    };
  },
  provide: {
    filemanager: [DetailsView, NavigationPane, Toolbar]
  }
}
```



```

}
</script>
<style>
@import
"https://ej2.syncfusion.com/vue/documentation/node_modules/@syncfusion/ej2-
base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-icons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-inputs/styles/material.css";
@import "../node_modules/@syncfusion/ej2-popups/styles/material.css";
@import "../node_modules/@syncfusion/ej2-buttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-splitbuttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-navigations/styles/material.css";
@import "../node_modules/@syncfusion/ej2-layouts/styles/material.css";
@import "../node_modules/@syncfusion/ej2-grids/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-
filemanager/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/file-manager/rtl-cs1" %}

### Specifying the current path of the File Manager

The current path of the File Manager can be specified initially or dynamically using the [path](#) property.

The following code snippet demonstrates specifying the current path in File Manager on rendering.

#### ~/SRC/APP.VUE

```

<template>
<div id="app">
<ejs-filemanager id="file-manager" :path="path"
:ajaxSettings="ajaxSettings">
</ejs-filemanager>
</div>
</template>
<script>
import { FileManagerComponent, DetailsView, NavigationPane, Toolbar } from
"@syncfusion/ej2-vue-filemanager";
export default {
components: {
'ejs-filemanager': FileManagerComponent
},
data () {
return {
ajaxSettings:
{
url: "https://ej2-aspcore-
service.azurewebsites.net/api/FileManager/FileOperations",
getImageUrl: "https://ej2-aspcore-
service.azurewebsites.net/api/FileManager/GetImage",
uploadUrl: "https://ej2-aspcore-
service.azurewebsites.net/api/FileManager/Upload",
downloadUrl: "https://ej2-aspcore-
service.azurewebsites.net/api/FileManager/Download"
},
// Initial view of File Manager is set to details view
view: "Details",
// Specify the required current path

```

```
path: '/Food'
};
},
provide: {
  filemanager: [DetailsView, NavigationPane, Toolbar]
}
}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-icons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-inputs/styles/material.css";
@import "../node_modules/@syncfusion/ej2-popups/styles/material.css";
@import "../node_modules/@syncfusion/ej2-buttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-splitbuttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-navigations/styles/material.css";
@import "../node_modules/@syncfusion/ej2-layouts/styles/material.css";
@import "../node_modules/@syncfusion/ej2-grids/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-filemanager/styles/material.css";
</style>
```

**Note:** You can refer to our [Vue File Manager](#) feature tour page for its groundbreaking feature representations. You can also explore our [Vue File Manager example](#) that shows you how to render the File Manager in Vue.

## Getting Started with the Vue File Manager Component in Vue 3

This article provides a step-by-step guide for setting up a [Vite](#) project with a JavaScript environment and integrating the Syncfusion Vue File Manager component using the [Composition API](#) / [Options API](#).

The **Composition API** is a new feature introduced in Vue.js 3 that provides an alternative way to organize and reuse component logic. It allows developers to write components as functions that use smaller, reusable functions called composition functions to manage their properties and behavior.

The **Options API** is the traditional way of writing Vue.js components, where the component logic is organized into a series of options that define the component's properties and behavior. These options include data, methods, computed properties, watchers, lifecycle hooks, and more.

### Prerequisites

#### [System requirements for Syncfusion Vue UI components](#)

#### Set up the Vite project

A recommended approach for beginning with Vue is to scaffold a project using [Vite](#). To create a new Vite project, use one of the commands that are specific to either NPM or Yarn.

```
`bash
```

```
npm create vite@latest
```

```
`
```

or

```
`bash
```

```
yarn create vite
```

`

Using one of the above commands will lead you to set up additional configurations for the project as below:

1. Define the project name: We can specify the name of the project directly. Let's specify the name of the project as `my-project` for this article.

`bash

? Project name: » my-project

`

2. Select `Vue` as the framework. It will create a Vue 3 project.

`bash

? Select a framework: » - Use arrow-keys. Return to submit.

Vanilla

Vue

React

Preact

Lit

Svelte

Others

`

3. Choose `JavaScript` as the framework variant to build this Vite project using JavaScript and Vue.

`bash

? Select a variant: » - Use arrow-keys. Return to submit.

JavaScript

TypeScript

Customize with create-vue ↗

Nuxt ↗

`

4. Upon completing the aforementioned steps to create the `my-project`, run the following command to install its dependencies:

`bash

cd my-project

npm install

`

or

```
`bash
cd my-project
yarn install
`
```

Now that `my-project` is ready to run with default settings, let's add Syncfusion components to the project.

### Add Syncfusion Vue packages

Syncfusion Vue component packages are available at [npmjs.com](https://www.npmjs.com). To use Syncfusion Vue components in the project, install the corresponding npm package.

This article uses the [Vue File Manager component](#) as an example. To use the Vue File Manager component in the project, the `@syncfusion/ej2-vue-navigations` package needs to be installed using the following command:

```
`bash
npm install @syncfusion/ej2-vue-navigations --save
`
```

or

```
`bash
yarn add @syncfusion/ej2-vue-navigations
`
```

### Import Syncfusion CSS styles

You can import themes for the Syncfusion Vue component in various ways, such as using CSS or SASS styles from npm packages, CDN, [CRG](#) and [Theme Studio](#). Refer to [themes topic](#) to know more about built-in themes and different ways to refer to themes in a Vue project.

In this article, `Material` theme is applied using CSS styles, which are available in installed packages. The necessary `Material` CSS styles for the File Manager component and its dependents were imported into the `<style>` section of `src/App.vue` file.

### ~/SRC/APP.VUE

```
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-icons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-inputs/styles/material.css";
@import "../node_modules/@syncfusion/ej2-popups/styles/material.css";
@import "../node_modules/@syncfusion/ej2-buttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-splitbuttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-navigations/styles/material.css";
@import "../node_modules/@syncfusion/ej2-layouts/styles/material.css";
@import "../node_modules/@syncfusion/ej2-grids/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-
filemanager/styles/material.css";
</style>
```

### Add Syncfusion Vue component

Follow the below steps to add the Vue File Manager component using **Composition API** or **Options API**:

1.First, import and register the File Manager component in the **script** section of the **src/App.vue** file. If you are using the **Composition API**, you should add the **setup** attribute to the **script** tag to indicate that Vue will be using the **Composition API**.

#### **COMPOSITION API (~/SRC/APP.VUE)**

```
<script setup>
import { FileManagerComponent as EjsFilemanager, DetailsView,
  NavigationPane, Toolbar } from "@syncfusion/ej2-vue-filemanager";
</script>
```

#### **OPTIONS API (~/SRC/APP.VUE)**

```
<script>
import { FileManagerComponent, DetailsView, NavigationPane, Toolbar } from
"@syncfusion/ej2-vue-filemanager";
//Component registration
export default {
  name: "App",
  components: {
    "ejs-filemanager": FileManagerComponent
  }
}
</script>
```

Note: By default, LargeIcons view will be initialized in the File Manager. If Grid view is required then the DetailsView module needs to be injected using **provide**.

2.Add the component definition in template section.

#### **~/SRC/APP.VUE**

```
<template>
<ejs-filemanager id="file-manager" :ajaxSettings="ajaxSettings"></ejs-
filemanager>
</template>
```

Here is the summarized code for the above steps in the **src/App.vue** file:

#### **COMPOSITION API (~/SRC/APP.VUE)**

```
<template>
<ejs-filemanager id="file-manager" :ajaxSettings="ajaxSettings"></ejs-
filemanager>
</template>
<script setup>
import { FileManagerComponent as EjsFilemanager, DetailsView,
  NavigationPane, Toolbar } from "@syncfusion/ej2-vue-filemanager";
import { provide } from "vue";
const filemanager = [DetailsView, NavigationPane, Toolbar];
provide('filemanager', filemanager);
```

```
const ajaxSettings = {
url: "https://ej2-aspcore-
service.azurewebsites.net/api/FileManager/FileOperations",
getImageUrl: "https://ej2-aspcore-
service.azurewebsites.net/api/FileManager/GetImage",
uploadUrl: "https://ej2-aspcore-
service.azurewebsites.net/api/FileManager/Upload",
downloadUrl: "https://ej2-aspcore-
service.azurewebsites.net/api/FileManager/Download"
};
</script>
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-icons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-inputs/styles/material.css";
@import "../node_modules/@syncfusion/ej2-popups/styles/material.css";
@import "../node_modules/@syncfusion/ej2-buttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-splitbuttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-navigations/styles/material.css";
@import "../node_modules/@syncfusion/ej2-layouts/styles/material.css";
@import "../node_modules/@syncfusion/ej2-grids/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-
filemanager/styles/material.css";
</style>
```

### OPTIONS API (~SRC/APP.VUE)

```
<template>
<ejs-filemanager id="file-manager" :ajaxSettings="ajaxSettings"></ejs-
filemanager>
</template>
<script>
import { FileManagerComponent, DetailsView, NavigationPane, Toolbar } from
"@syncfusion/ej2-vue-filemanager";
//Component registration
export default {
name: "App",
components: {
"ejs-filemanager": FileManagerComponent
},
data() {
return {
ajaxSettings:
{
url: "https://ej2-aspcore-
service.azurewebsites.net/api/FileManager/FileOperations",
getImageUrl: "https://ej2-aspcore-
service.azurewebsites.net/api/FileManager/GetImage",
uploadUrl: "https://ej2-aspcore-
service.azurewebsites.net/api/FileManager/Upload",
downloadUrl: "https://ej2-aspcore-
service.azurewebsites.net/api/FileManager/Download"
},
};
},
provide: {
```

```
filemanager: [DetailsView, NavigationPane, Toolbar]
}
}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-icons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-inputs/styles/material.css";
@import "../node_modules/@syncfusion/ej2-popups/styles/material.css";
@import "../node_modules/@syncfusion/ej2-buttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-splitbuttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-navigations/styles/material.css";
@import "../node_modules/@syncfusion/ej2-layouts/styles/material.css";
@import "../node_modules/@syncfusion/ej2-grids/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-
filemanager/styles/material.css";
</style>
```

### Run the project

To run the project, use the following command:

```
`bash
```

```
npm run dev
```

```
`
```

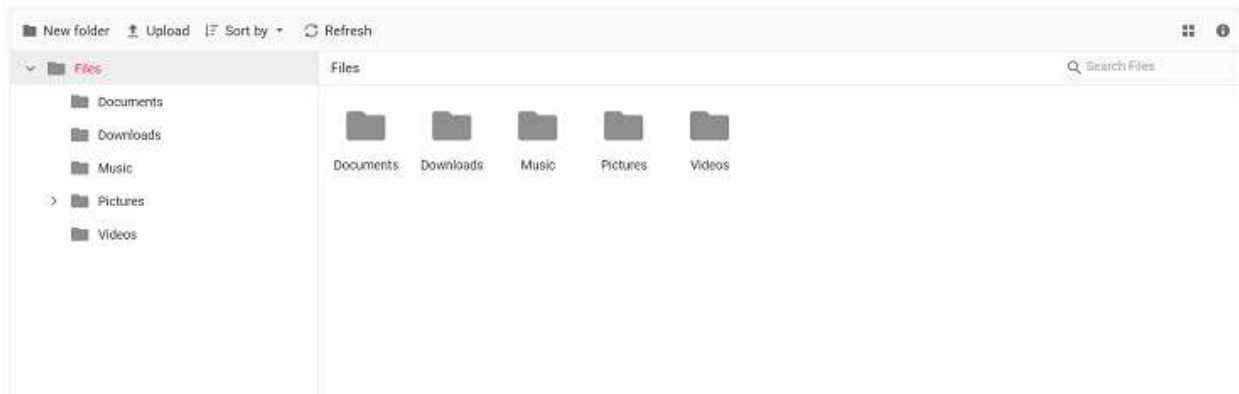
or

```
`bash
```

```
yarn run dev
```

```
`
```

The output will appear as follows:



For migrating from Vue 2 to Vue 3, refer to the [migration](#) documentation.

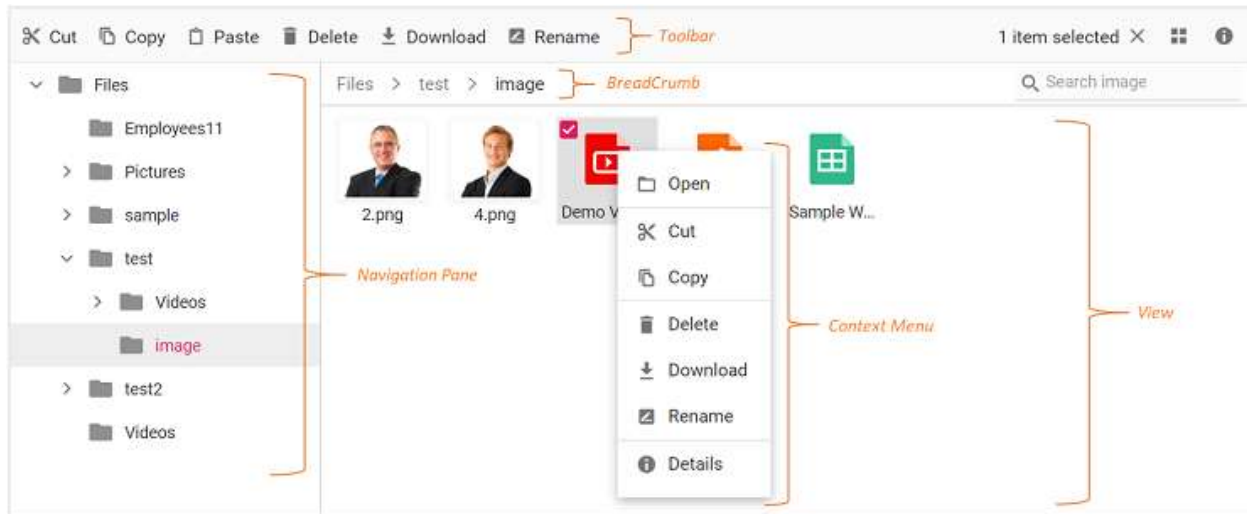
See also

- [Getting Started with Vue UI Components using Composition API and TypeScript](#)
- [Getting Started with Vue UI Components using Options API and TypeScript](#)

### User interface in Vue File manager component

The file manager UI is comprised of several sections like view, toolbar, breadcrumb, context menu, and so on. The UI of the file manager is enhanced with injectable modules like **Details View** for browsing files and folders in a grid, **Navigation Pane** for folder navigation, and **Toolbar** for file operations. The file manager with all feature modules have the following sections in its UI.

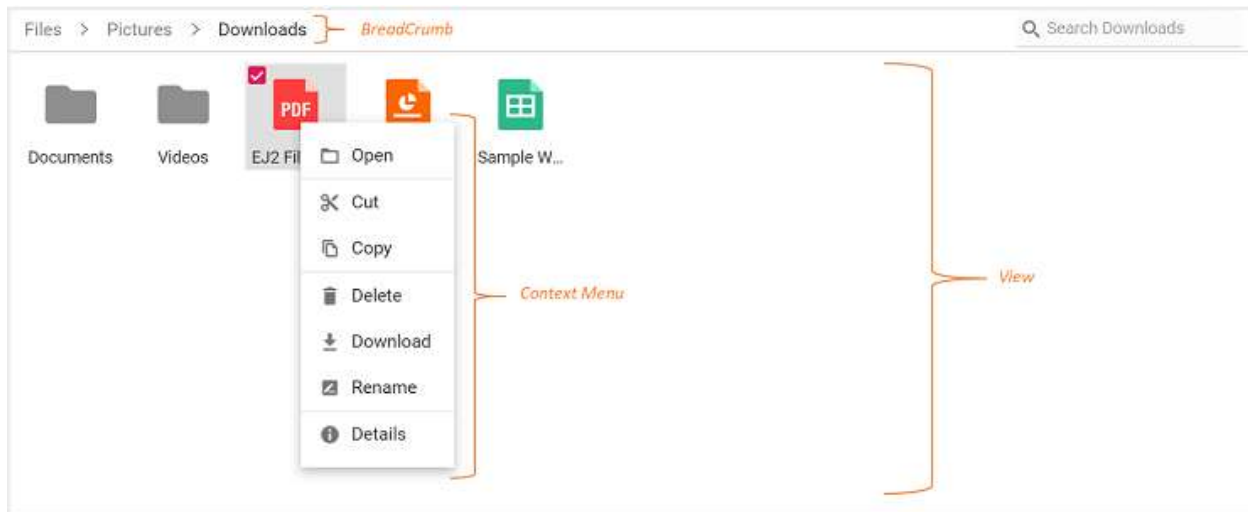
- [View](#) (Large Icons view for browsing files and folders),
- [Toolbar](#) (For direct access to file operations),
- [Navigation Pane](#) (For easy navigation between folders),
- [Breadcrumb](#) (For parent folder navigations),
- [Context Menu](#) (For accessing file operations).



The basic file manager is a light weight component with all the basic functions. The basic file manager have the following sections in its UI to browse files and folders and manage them with file operations.

- [View](#) (Large Icons view for browsing files and folders),
- [Breadcrumb](#) (For parent folder navigations),
- [Context Menu](#) (For accessing file operations).



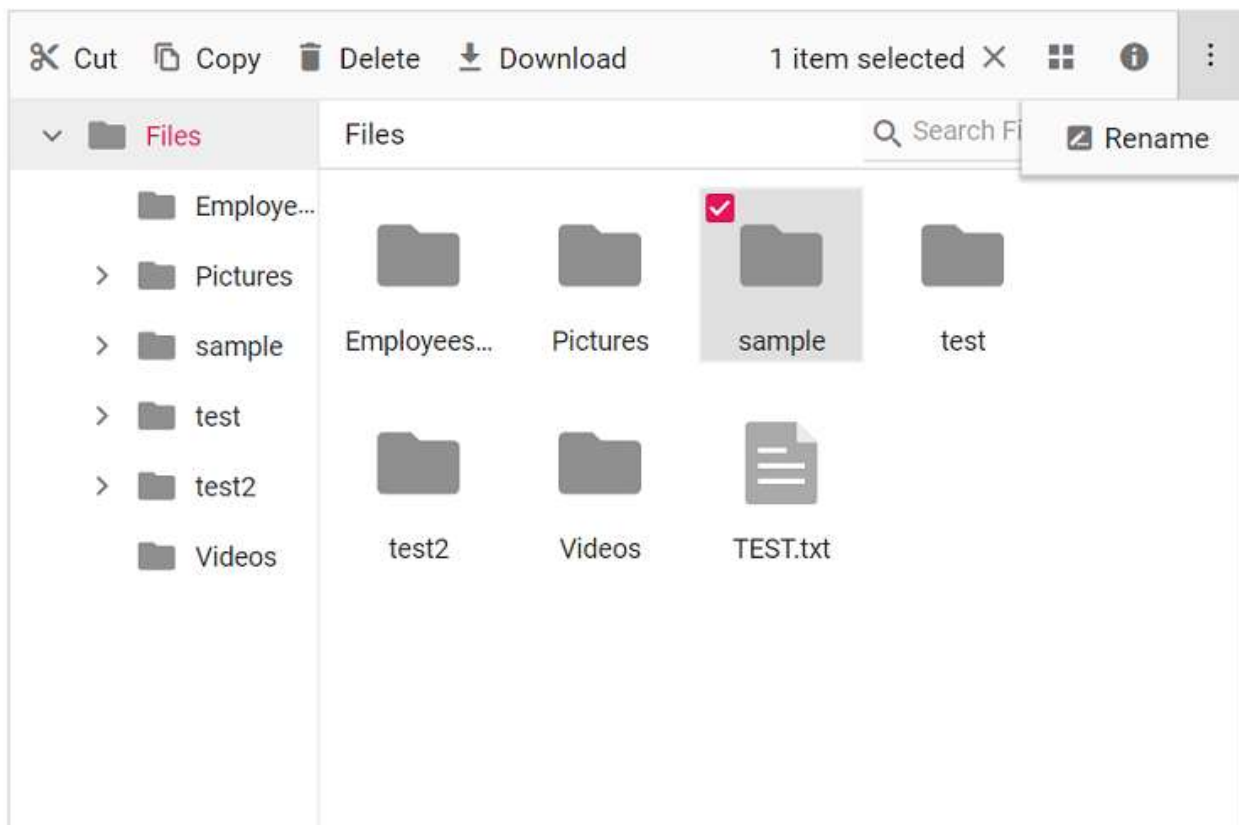


### Toolbar

The **Toolbar** provides easy access to the file operations using different buttons and it is presented at the top of the file manager.

If the toolbar items exceed the size of the toolbar, then the exceeding toolbar size will be moved to toolbar popup with a dropdown button at the end of toolbar.

*\*Refer [Toolbar](#) section in file operations to know more about the buttons present in toolbar\*.*



### Files and folders navigation

The file manager provides navigation between files and folders using the following two options.

- [Navigation Pane](#)
- [Breadcrumb](#)

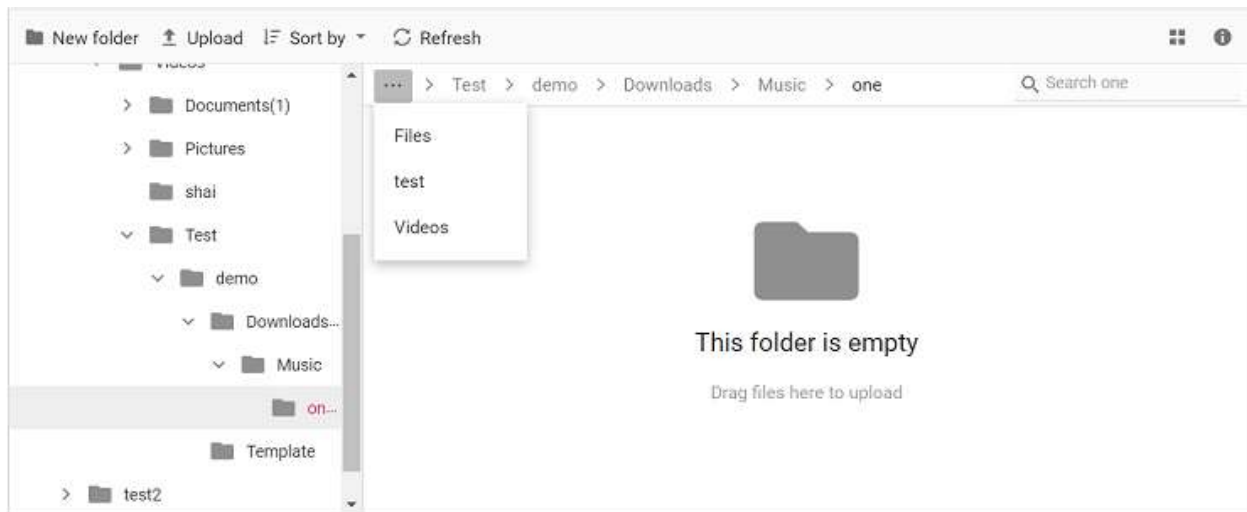
### Navigation pane

The navigation pane is an injectable module so, it should be injected before rendering the file manager to use its functionality. It displays the folder hierarchy of the file system and provides easy navigation to the desired folder. Using [navigationPaneSettings](#) minimum and maximum width of the navigation pane can be changed. The navigation pane can be shown or hidden using the `visible` option in the [navigationPaneSettings](#).

### BreadCrumb

The file manager provides breadcrumb for navigating to the parent folders. The breadcrumb the in file manager is responsible for resizing.

Whenever the path length exceeds the breadcrumb length, a dropdown button will be added at the starting of the breadcrumb to hold the parent folders adjacent to root.



### View

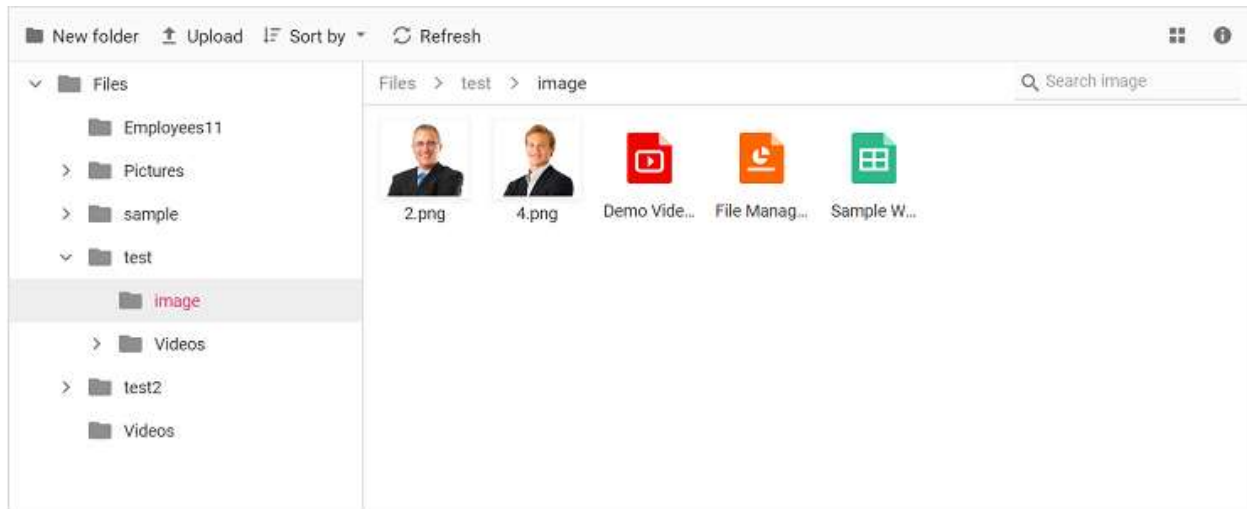
View is the section where the files and folders are displayed for the user to browse. The file manager has two types of views to display the files and folders.

- [Large Icons View](#)
- [Details View](#)

The **large icons view** is the default starting view in the file manager. The view can be changed by using the [toolbar](#) view button or by using the view menu in [context menu](#). The [view](#) API can also be used to change the initial view of the file manager.

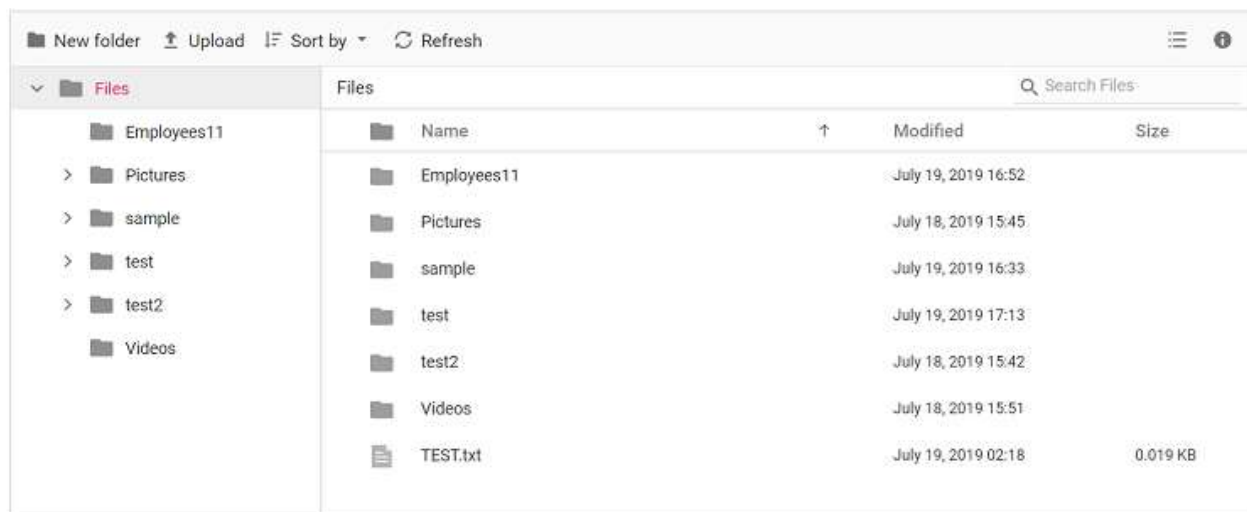
### Large icons view

In the large icons view, the thumbnail icons will be shown in a larger size, which displays the data in a form that best suits their content. For image and video type files, a **preview** will be displayed. Extension thumbnails will be displayed for other type files.



### Details view

Details view is an injectable module in the file manager so, it should be injected before rendering the file manager to avail its functionality. In the details view, the files are displayed in a sorted list order. This file list comprises of several columns of information about the files such as **Name**, **Date Modified**, **Type**, and **Size**. Each file has its own small icon representing the file type. Additional columns can be added using [detailsViewSettings](#) API. The details view allows you to perform sorting using column header.

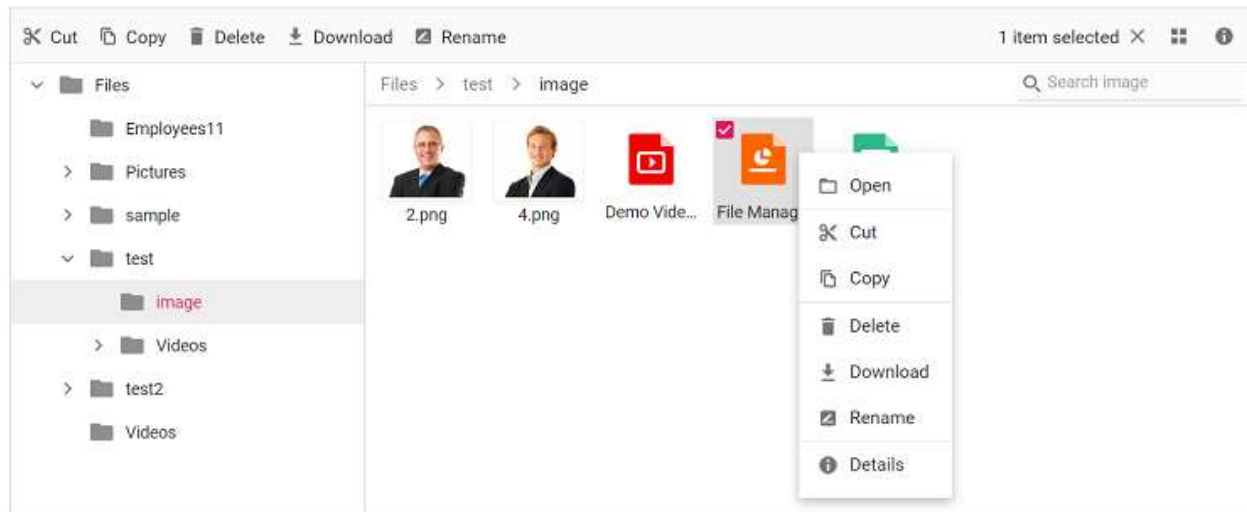


### Context menu

The context menu appears on user interaction such as right-click. The file manager is provided with context menu support to perform list of file operations with the files and folders. Context menu appears with varying menu items based on the targets such as file, folder (including navigation pane folders), and layout (empty area in view).

Context menu can be customized using the [contextMenuSettings](#), [menuOpen](#), and [menuClick](#) events.

*\*Refer [Context Menu](#) section in file operations to know more about the menu items present in context menu\*.*



### File operations in Vue File manager component

The file manager component is used to browse, manage, and organize the files and folders in a file system through a web application. All basic file operations like creating a new folder, uploading and downloading of files in the file system, and deleting and renaming of existing files and folders are available in the file manager component. Additionally, previewing of image files is also provided in the file manager component.

The following table represents the basic operations available in the file manager and their corresponding functions.

Operation Name	Function
----------------	----------

----	----
------	------

read	Read the details of files or folders available in the given path from the file system, to display the files for the user to browse the content.
------	-------------------------------------------------------------------------------------------------------------------------------------------------

create	Creates a new folder in the current path of the file system.
--------	--------------------------------------------------------------

delete	Removes the file or folder from the file server.
--------	--------------------------------------------------

rename	Rename the selected file or folder in the file system.
--------	--------------------------------------------------------

search	Searches for items matching the search string in the current and child directories.
--------	-------------------------------------------------------------------------------------

details	Gets the detail of the selected item(s) from the file server.
---------	---------------------------------------------------------------

copy	Copy the selected file or folder in the file system.
------	------------------------------------------------------

move	Cut the selected file or folder in the file server.
------	-----------------------------------------------------

upload	Upload files to the current path or directory in the file system.
--------	-------------------------------------------------------------------

download	Downloads the file from the server and the multiple files can be downloaded as ZIP files.
----------	-------------------------------------------------------------------------------------------

The *CreateFolder*, *Remove*, and *Rename* actions will be reflected in the file manager only after the successful response from the server.

### Folder Upload support

To perform the directory(folder) upload in File Manager, set [directoryUpload](#) as true within the uploadSettings property. The directory upload feature is supported for the following file service providers:

- Physical file service provider.
- Azure file service provider.
- NodeJS file service provider.
- Amazon file service provider.

In the following example, directory upload is enabled/disabled on DropDownButton selection.

#### APP.VUE

```
<template>
<div id="app">
  <div class="control-section folder-upload">
    <div class="sample-container">
      <ejs-filemanager id="filemanager" :ajaxSettings='ajaxSettings'
:created='onCreated' >
    </ejs-filemanager>
    </div>
  </div>
</div>
</template>
<script>
import Vue from "vue";
import { FileManagerPlugin, NavigationPane, Toolbar, DetailsView,
FileManagerComponent } from "@syncfusion/ej2-vue-filemanager";
import { DropDownButton, ItemModel } from "@syncfusion/ej2-splitbuttons";
Vue.use(FileManagerPlugin);
// File Manager directory upload feature sample
let hostUrl = 'https://ej2-aspcore-service.azurewebsites.net/';
export default {
  data () {
    return {
      ajaxSettings: {
        url: "https://ej2-aspcore-
service.azurewebsites.net/api/FileManager/FileOperations",
        getImageUrl: "https://ej2-aspcore-
service.azurewebsites.net/api/FileManager/GetImage",
        uploadUrl: "https://ej2-aspcore-
service.azurewebsites.net/api/FileManager/Upload",
        downloadUrl: "https://ej2-aspcore-
service.azurewebsites.net/api/FileManager/Download"
      }
    };
  },
  provide: {
    filemanager: [NavigationPane, DetailsView, Toolbar]
  },
  methods: {
    onCreated: function (args) {
```

```

        var customBtn =
document.getElementById("filemanager_tb_upload");
        customBtn.onclick = (e) => {
            e.stopPropagation();
        };
        //DropDownButton items definition
        var items = [{ text: "Folder" }, { text: "Files" }];
        var drpDownBtn = new DropDownButton({
            items: items,
            select: (args) => {
                var fileObj =
document.getElementById("filemanager").ej2_instances[0];
                if (args.item.text === "Folder") {
                    fileObj.uploadSettings.directoryUpload = true;
                } else {
                    fileObj.uploadSettings.directoryUpload = false;
                }
                setTimeout(function () {
                    var uploadBtn = document.querySelector(".e-file-
select-wrap button");
                    uploadBtn.click();
                }, 100);
            },
        }, "#filemanager_tb_upload"
    );
    },
    },
    });
</script>
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-icons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-inputs/styles/material.css";
@import "../node_modules/@syncfusion/ej2-popups/styles/material.css";
@import "../node_modules/@syncfusion/ej2-buttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-splitbuttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-navigations/styles/material.css";
@import "../node_modules/@syncfusion/ej2-layouts/styles/material.css";
@import "../node_modules/@syncfusion/ej2-grids/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-
filemanager/styles/material.css";
.folder-upload .sample-container {
    margin: 10px 10px 10px 10px;
}
</style>

```

{% previewsample "page.domainurl/code-snippet/file-manager/directory-upload-cs1" %}

#### Physical file service provider

To achieve the directory upload in the physical file service provider, use the below code snippet in **ActionResult Upload** method in the **Controllers/FileManagerController.cs** file.

```
`ts
```

```
[Route("Upload")]
```

```
public IActionResult Upload(string path, IList<IFormFile> uploadFiles, string action)
{
    FileManagerResponse uploadResponse;
    foreach (var file in uploadFiles)
    {
        var folders = (file.FileName).Split('/');
        // checking the folder upload
        if (folders.Length > 1)
        {
            for (var i = 0; i < folders.Length - 1; i++)
            {
                string newDirectoryPath = Path.Combine(this.basePath + path, folders[i]);
                if (!Directory.Exists(newDirectoryPath))
                {
                    this.operation.ToCamelCase(this.operation.Create(path, folders[i]));
                }
                path += folders[i] + "/";
            }
        }
    }
    uploadResponse = operation.Upload(path, uploadFiles, action, null);
    if (uploadResponse.Error != null)
    {
        Response.Clear();
        Response.ContentType = "application/json; charset=utf-8";
        Response.StatusCode = Convert.ToInt32(uploadResponse.Error.Code);
        Response.HttpContext.Features.Get<IHttpResponseFeature>().ReasonPhrase =
            uploadResponse.Error.Message;
    }
    return Content("");
}
```

Refer to the [GitHub](#) for more details

And also add the below code snippet in `FileManagerResponse Upload` method in `Models/PhysicalFileProvider.cs` file.

```
`ts
string[] folders = name.Split('/');
string fileName = folders[folders.Length - 1];
var fullName = Path.Combine((this.contentRootPath + path), fileName);
`
```

Refer to the [GitHub](#) for more details.

#### *Azure file service provider*

For Azure file service provider, no customizations are needed for directory upload with server side and this will work with the below default upload method code.

Refer to the [GitHub](#) for more details.

#### *NodeJS file service provider*

To perform the directory upload in the NodeJS file service provider, use the below code snippet in `app.post` method in the `filesystem-server.js` file.

```
`ts
var folders = (req.body.filename).split('/');
var filepath = req.body.path;
var uploadedFileName = folders[folders.length - 1];
// checking the folder upload
if (folders.length > 1)
{
  for (var i = 0; i < folders.length - 1; i++)
  {
    var newDirectoryPath = path.join(contentRootPath + filepath, folders[i]);
    if (!fs.existsSync(newDirectoryPath)) {
      fs.mkdirSync(newDirectoryPath);
      (async () => {
        await FileManagerDirectoryContent(req, res, newDirectoryPath).then(data => {
          response = { files: data };
          response = JSON.stringify(response);
        });
      })();
    }
    filepath += folders[i] + "/";
  }
}
```



```
}  
fs.rename('./' + uploadedFileName, path.join(contentRootPath, filepath + uploadedFileName), function  
(err) {  
  if (err) {  
    if (err.code !== 'EBUSY') {  
      errorValue.message = err.message;  
      errorValue.code = err.code;  
    }  
  }  
});  
}  
`
```

Refer to the [GitHub](#) for more details.

#### *Amazon file service provider*

To perform the directory upload in the Amazon file service provider, use the below code snippet in `IActionResult AmazonS3Upload` method in the `Controllers/AmazonS3ProviderController.cs` file.

```
`ts  
foreach (var file in uploadFiles)  
{  
  var folders = (file.FileName).Split('/');  
  // checking the folder upload  
  if (folders.Length > 1)  
  {  
    for (var i = 0; i < folders.Length - 1; i++)  
    {  
      if (!this.operation.checkFileExist(path, folders[i]))  
      {  
        this.operation.ToCamelCase(this.operation.Create(path, folders[i], dataObject));  
      }  
      path += folders[i] + "/";  
    }  
  }  
}
```

Refer to the [GitHub](#) for more details.

And also add the below code snippet in `AsyncUpload` method in `Models/AmazonS3FileProvider.cs` file.

```
`ts
string[] folders = file.FileName.Split('/');
string name = folders[folders.Length - 1];
`
```

Refer to the [GitHub](#) for more details.

### File operation request and response Parameters

The default parameters available in file operation request from the file manager and the corresponding response parameters required by the file manager are listed as follows.

#### Read

The following table represents the request parameters of *read* operations.

Parameter	Type	Default	Explanation
----	----	----	----
action	String	read	Name of the file operation.
path	String	-	Relative path from which the data has to be read.
showHiddenItems	Boolean	-	Defines show or hide the hidden items.
data	FileManagerDirectoryContent	-	Details about the current path (directory).

*\*Refer [File request and response contents](#) for the contents of data\*.*

*Example:*

```
`ts
{
  action: "read",
  path: "/",
  showHiddenItems: false,
  data: []
}
`
```

The following table represents the response parameters of *read* operations.

Parameter	Type	Default	Explanation
----	----	----	----
cwd	FileManagerDirectoryContent	-	Path (Current Working Directory) details.
files	FileManagerDirectoryContent[]	-	Details of files and folders present in given path or directory.

|error|ErrorDetails|-|Error Details|

*\*Refer [File request and response contents](#) for the contents of cwd, files, and error\*.*

*Example:*

```
`ts
{
  cwd:
  {
    name:"Download",
    size:0,
    dateModified:"2019-02-28T03:48:19.8319708+00:00",
    dateCreated:"2019-02-27T17:36:15.812193+00:00",
    hasChild:false,
    isFile:false,
    type:"",
    filterPath:"//Download//"
  },
  files:[
    {
      name:"Sample Work Sheet.xlsx",
      size:6172,
      dateModified:"2019-02-27T17:23:50.9651206+00:00",
      dateCreated:"2019-02-27T17:36:15.8151955+00:00",
      hasChild:false,
      isFile:true,
      type:".xlsx",
      filterPath:"//Download//"
    }
  ],
  error:null,
  details:null
}
```

### Create

The following table represents the request parameters of *create* operations.

Parameter	Type	Default	Explanation
action	String	create	Name of the file operation.
path	String	-	Relative path in which the folder has to be created.
name	String	-	Name of the folder to be created.
data	FileManagerDirectoryContent	-	Details about the current path (directory).

*\*Refer [File request and response contents](#) for the contents of data\**

*Example:*

```
`ts
{
  action: "create",
  data: [
    {
      dateCreated: "2019-02-27T17:36:15.6571949+00:00",
      dateModified: "2019-03-12T10:17:31.8505975+00:00",
      filterPath: "/",
      hasChild: true,
      isFile: false,
      name: files,
      nodeId: "fe_tree",
      size: 0,
      type: ""
    }
  ],
  name: "Hello",
  path: "/"
}
```

The following table represents the response parameters of *create* operations.

Parameter	Type	Default	Explanation
-----	-----	-----	-----

|files|FileManagerDirectoryContent[]|-|Details of the created folder|

|error|ErrorDetails|-|Error Details|

*\*Refer [File request and response contents](#) for the contents of files and error\*.*

*Example:*

```
`ts
{
  cwd: null,
  files: [
    {
      dateCreated: "2019-03-15T10:25:05.3596171+00:00",
      dateModified: "2019-03-15T10:25:05.3596171+00:00",
      filterPath: null,
      hasChild: false,
      isFile: false,
      name: "New",
      size: 0,
      type: ""
    }
  ],
  details: null,
  error: null
}
```

### [Rename](#)

The following table represents the request parameters of *rename* operations.

Parameter	Type	Default	Explanation
----	----	----	----
action	String	rename	Name of the file operation.
path	String	-	Relative path in which the item is located.
name	String	-	Current name of the item to be renamed.
newname	String	-	New name for the item.
data	FileManagerDirectoryContent	-	Details of the item to be renamed.

*\*Refer [File request and response contents](#) for the contents of data\*.*

*Example:*

```
`ts
{
  action: "rename",
  data: [
    {
      dateCreated: "2019-03-20T05:22:34.621Z",
      dateModified: "2019-03-20T08:45:56.000Z",
      filterPath: "/Pictures/Nature/",
      hasChild: false,
      iconClass: "e-fe-image",
      isFile: true,
      name: "seaviews.jpg",
      size: 95866,
      type: ".jpg"
    }
  ],
  newname: "seaview.jpg",
  name: "seaviews.jpg",
  path: "/Pictures/Nature/"
}
```

The following table represents the response parameters of *rename* operations.

Parameter	Type	Default	Explanation
files	FileManagerDirectoryContent[]	-	Details of the renamed item.
error	ErrorDetails	-	Error Details

*\*Refer [File request and response contents](#) for the contents of files and error\*.*

*Example:*

```
`ts
{
  cwd:null,
  files:[
```

```
{
  name:"seaview.jpg",
  size:95866,
  dateModified:"2019-03-20T08:45:56+00:00",
  dateCreated:"2019-03-20T05:22:34.6214847+00:00",
  hasChild:false,
  isFile:true,
  type:".jpg",
  filterPath:"//Pictures//Nature//seaview.jpg"
},
error:null,
details:null
},
,
```

#### Delete

The following table represents the request parameters of *delete* operations.

Parameter	Type	Default	Explanation
----	----	----	----
action	String	delete	Name of the file operation.
path	String	-	Relative path where the items to be deleted are located.
names	String[]	-	List of the items to be deleted.
data	FileManagerDirectoryContent	-	Details of the item to be deleted.

*\*Refer [File request and response contents](#) for the contents of data\*.*

*Example:*

```
`ts
{
  action: "delete",
  path: "/Hello/",
  names: ["New"],
  data: []
},
,
```

The following table represents the response parameters of *delete* operations.

Parameter	Type	Default	Explanation
----	----	----	----
files	FileManagerDirectoryContent[]	-	Details about the deleted item(s).
error	ErrorDetails	-	Error Details

*\*Refer [File request and response contents](#) for the contents of files and error\*.*

*Example:*

```
`ts
{
  cwd: null,
  details: null,
  error: null,
  files: [
    {
      dateCreated: "2019-03-15T10:13:30.346309+00:00",
      dateModified: "2019-03-15T10:13:30.346309+00:00",
      filterPath: "/Hello/folder",
      hasChild: true,
      isFile: false,
      name: "folder",
      size: 0,
      type: ""
    }
  ]
}
```

#### *Details*

The following table represents the request parameters of *details* operations.

Parameter	Type	Default	Explanation
----	----	----	----
action	String	details	Name of the file operation.
path	String	-	Relative path where the items are located.
names	String[]	-	List of the items to get details.



|data|FileManagerDirectoryContent|-|Details of the selected item.|

*\*Refer [File request and response contents](#) for the contents of data\*.*

*Example:*

```
`ts
{
  action: "details",
  path: "/FileContents/",
  names: ["All Files"],
  data: []
},
```

The following table represents the response parameters of *details* operations.

Parameter	Type	Default	Explanation
details	FileManagerDirectoryContent	-	Details of the requested item(s).
error	ErrorDetails	-	Error Details

*\*Refer [File request and response contents](#) for the contents of details and error\*.*

*Example:*

```
`ts
{
  cwd:null,
  files:null,
  error:null,
  details:
  {
    name:"All Files",
    location:"//Files//FileContents//All Files",
    isFile:false,
    size:"679.8 KB",
    created:"3/8/2019 10:18:37 AM",
    modified:"3/8/2019 10:18:39 AM",
    multipleFiles:false
  }
}
```

```
}  
,
```

### Search

The following table represents the request parameters of *search* operations.

Parameter	Type	Default	Explanation
----	----	----	----
action	String	search	Name of the file operation.
path	String	-	Relative path to the directory where the files should be searched.
showHiddenItems	Boolean	-	Defines show or hide the hidden items.
caseSensitive	Boolean	-	Defines search is case sensitive or not.
searchString	String	-	String to be searched in the directory.
data	FileManagerDirectoryContent	-	Details of the searched item.

*Example:*

```
`ts  
{  
  action: "search",  
  path: "/",  
  searchString: "nature",  
  showHiddenItems: false,  
  caseSensitive: false,  
  data: []  
}  
,
```

The following table represents the response parameters of *search* operations.

Parameter	Type	Default	Explanation
----	----	----	----
cwd	FileManagerDirectoryContent	-	Path (Current Working Directory) details.
files	FileManagerDirectoryContent[]	-	Files and folders in the searched directory that matches the search input.
error	ErrorDetails	-	Error Details

*\*Refer [File request and response contents](#) for the contents of cwd, files and error.\**

*Example:*

```
`ts  
{
```

```
cwd:
{
  name:files,
  size:0,
  dateModified:"2019-03-15T10:07:00.8658158+00:00",
  dateCreated:"2019-02-27T17:36:15.6571949+00:00",
  hasChild:true,
  isFile:false,
  type:"",
  filterPath:"/"
},
files:[
{
  name:"Nature",
  size:0,
  dateModified:"2019-03-08T10:18:42.9937708+00:00",
  dateCreated:"2019-03-08T10:18:42.5907729+00:00",
  hasChild:true,
  isFile:false,
  type:"",
  filterPath:"//FileContents//Nature"
}
],
error:null,
details:null
},
,
```

### Copy

The following table represents the request parameters of *copy* operations.

Parameter	Type	Default	Explanation
----	----	----	----
action	String	copy	Name of the file operation.
path	String	-	Relative path to the directory where the files should be copied.

names	String[]	-	List of files to be copied.
targetPath	String	-	Relative path where the items to be pasted are located.
data	FileManagerDirectoryContent	-	Details of the copied item.
renameFiles	String[]	-	Details of the renamed item.

*Example:*

```
`ts
{
  action: "copy",
  path: "/",
  names: ["6.png"],
  renameFiles: ["6.png"],
  targetPath: "/Videos/"
}
```

The following table represents the response parameters of *copy* operations.

Parameter	Type	Default	Explanation
----	----	----	----
cwd	FileManagerDirectoryContent	-	Path (Current Working Directory) details.
files	FileManagerDirectoryContent[]	-	Details of copied files or folders
error	ErrorDetails	-	Error Details

*\*Refer [File request and response contents](#) for the contents of cwd, files and error.\**

*Example:*

```
`ts
{
  cwd:null,
  files:[
    {
      path:null,
      action:null,
      newName:null,
      names:null,
      name:"justin.mp4",
      size:0,
```

```
previousName:"album.mp4",
dateModified:"2019-06-21T06:58:32+00:00",
dateCreated:"2019-06-24T04:22:14.6245618+00:00",
hasChild:false,
isFile:true,
type:".mp4",
id:null,
filterPath:"/"
}
],
error:null,
details:null
}
、
```

### Move

The following table represents the request parameters of *move* operations.

Parameter	Type	Default	Explanation
----	----	----	----
action	String	move	Name of the file operation.
path	String	-	Relative path to the directory where the files should be copied.
names	String[]	-	List of files to be moved.
targetPath	String	-	Relative path where the items to be pasted are located.
data	FileManagerDirectoryContent	-	Details of the moved item.
renameFiles	String[]	-	Details of the renamed item.

### Example:

```
`ts
{
  action: "move",
  path: "/",
  names: ["6.png"],
  renameFiles: ["6.png"],
  targetPath: "/Videos/"
}
```

The following table represents the response parameters of *copy* operations.

Parameter	Type	Default	Explanation
----	----	----	----
cwd	FileManagerDirectoryContent	-	Path (Current Working Directory) details.
files	FileManagerDirectoryContent[]	-	Details of cut files or folders
error	ErrorDetails	-	Error Details

*\*Refer [File request and response contents](#) for the contents of cwd, files and error.\**

*Example:*

```
`ts
{
  cwd:null,
  files:[
    {
      path:null,
      action:null,
      newName:null,
      names:null,
      name:"justin biber.mp4",
      size:0,
      previousName:"justin biber.mp4",
      dateModified:"2019-06-21T06:58:32+00:00",
      dateCreated:"2019-06-24T04:26:49.2690476+00:00",
      hasChild:false,
      isFile:true,
      type:".mp4",
      id:null,
      filterPath:"//Videos//"
    }
  ],
  error:null,
  details:null
}
```

`

### Upload

The following table represents the request parameters of *Upload* operations.

Parameter	Type	Default	Explanation
action	String	Save	Name of the file operation.
path	String	-	Relative path to the location where the file has to be uploaded.
uploadFiles	<code>IList&lt;IFormFile&gt;</code>	-	File that are uploaded.

#### Example:

```
`ts
uploadFiles: (binary),
path: /,
action: Save,
data: {
  path:null,
  action:null,
  newName:null,
  names:null,
  name:"Downloads",
  size:0,
  previousName:null,
  dateModified:"2019-07-22T11:23:46.7153977 00:00",
  dateCreated:"2019-07-22T11:26:13.9047229 00:00",
  hasChild:false,
  isFile:false,
  type:"",
  id:null,
  filterPath:"/",
  targetPath:null,
  renameFiles:null,
  uploadFiles:null,
  caseSensitive:false,
  searchString:null,
```

```
showHiddenItems:false,
fmiconClass:null,
fmid:"fetree1",
fmpld:null,
fmselected:false,
fmicon:null,
data:null,
targetData:null,
permission:null
},
,
```

The upload response is an empty string.

#### [Download](#)

The following table represents the request parameters of *download* operations.

Parameter	Type	Default	Explanation
----	----	----	----
action	String	download	Name of the file operation
path	String	-	Relative path to location where the files to download are present.
names	String[]	-	Name list of the items to be downloaded.
data	FileManagerDirectoryContent	-	Details of the download item.

*Example:*

```
`ts
{
  action:"download",
  path:"/",
  names:["1.png"],
  data:[
    {
      path:null,
      action:null,
      newName:null,
      names:null,
      name:"1.png",
```



```
size:49792,
previousName:null,
dateModified:"2019-07-22T12:15:45.0972405+00:00",
dateCreated:"2019-07-22T12:15:45.0816042+00:00",
hasChild:false,
isFile:true,
type:".png",
id:null,
filterPath:"/",
targetPath:null,
renameFiles:null,
uploadFiles:null,
caseSensitive:false,
searchString:null,
showHiddenItems:false,
fmiconClass:"e-fe-image",
fmid:null,
fmpld:null,
fmselected:false,
fmicon:null,
data:null,
targetData:null,
permission:null,
fmcreated:"2019-07-22T12:15:45.081Z",
fmmodified:"2019-07-22T12:15:45.097Z",
fmimageUrl:"https://ej2-aspcore-service.azurewebsites.net/api/FileManager/GetImage?path=/1.png",
fmimageAttr:
{
  alt:"1.png"
},
fmhtmlAttr:
{
  class:"e-large-icon",
```

```
title:"1.png"
}
}
]
}
、
```

Downloads the requested items from the file server in response.

#### [GetImage](#)

The following table represents the request parameters of *GetImage* operations.

Parameter	Type	Default	Explanation
----	----	----	----
path	String	-	Relative path to the image file

Return the image as a file stream in response.

The request from the file manager can be customized using the [beforeSend](#) event. Additional information can be passed to the file manager in file operation response and can be used in customization.

#### [File request and response contents](#)

The following table represents the contents of *data*, *cwd*, and *files* in the file manager request and response.

Parameter	Type	Default	Explanation
----	----	----	----
name	String	-	File name
dateCreated	String	-	Date in which file was created (UTC Date string).
dateModified	String	-	Date in which file was last modified (UTC Date string).
filterPath	String	-	Relative path to the file or folder.
hasChild	Boolean	-	Defines this folder has any child folder or not.
isFile	Boolean	-	Say whether the item is file or folder.
size	Number	-	File size
type	String	-	File extension

The following table represents the contents of *error* in the file manager request and response.

Parameter	Type	Default	Explanation
----	----	----	----
code	String	-	Error code
message	String	-	Error message
fileExists	String[]	-	List of duplicate file names

The following table represents the contents of *details* in the file manager request and response.

Parameter	Type	Default	Explanation
----	----	----	----
name	String	-	File name
dateCreated	String	-	Date in which file was created (UTC Date string).
dateModified	String	-	Date in which file was last modified (UTC Date string).
filterPath	String	-	Relative path to the file or folder.
hasChild	Boolean	-	Defines this folder has any child folder or not.
isFile	Boolean	-	Say whether the item is file or folder.
size	Number	-	File size
type	String	-	File extension
multipleFiles	Boolean	-	Say whether the details are about single file or multiple files.

### Action Buttons

The file manager has several menu buttons to access the file operations. The list of menu buttons available in the file manager is given in the following table.

Menu Button	Behaviour
----	----
SortBy	Opens the sub menu to choose the sorting order and sorting parameter.
View	Opens the sub menu to choose the View.
Open	Navigates to the selected folder. Opens the preview for image files.
Refresh	Initiates the read operation for the current directory and displays the updated directory content.
NewFolder	Opens the new folder dialog box to receive the name for the new folder.
Rename	Opens the rename dialog box to receive the new name for the selected item.
Delete	Opens the delete dialog box to confirm the removal of the selected items from the file system.
Upload	Opens the upload box to select the items to upload to the file system.
Download	Downloads the selected item(s).
Details	Get details about the selected items and display them in details dialog box.
SelectAll	Selects all the files and folders displayed in the view section.

The action menu buttons are present in the toolbar and context menu. The toolbar contains the buttons based on the selected items count, while the context menu will appear with a list based on the target.

### Toolbar

The toolbar can be divided into two sections as right and left. Whenever the toolbar buttons exceed the size, the buttons present in the left section of the toolbar will be moved to the toolbar popup.

The following table provides the toolbar buttons that appear based on the selection.

<!-- markdownlint-disable MD033 -->

Selected Items Count	Left section	Right section
0 (none of the item )	<i>SortBy</i> Refresh <i>NewFolder</i> Upload	<i>View</i> Details
1 (single item selected)	<i>Delete</i> Download* Rename	<i>Selected items count</i> View* Details
>1 (multiple selection)	<i>Delete</i> Download	<i>Selected items count</i> View* Details

### Context menu

The following table provides the default context menu item and the corresponding target areas.

<!-- markdownlint-disable MD033 -->

Menu Name	Menu Items	Target
Layout	<i>SortBy</i> View <i>Refresh</i> <i>NewFolder</i> Upload Details* Select all	<i>Empty space in the view section (details view and large icon view area). Empty folder content.</i>
Folders	<i>Open</i> Delete <i>Rename</i> Downloads* Details	* Folders in treeview, details view, and large icon view.
Files	<i>Open</i> Delete <i>Rename</i> Downloads* Details	* Files in details view and large icon view.

## Views in Vue File manager component

View is the section where the files and folders are displayed for the user to browse. The [view](#) API can also be used to change the initial view of the file manager.

The file manager has two types of [views](#) to display the files and folders.

- [Largelcons View](#)
- [Details View](#)

### Largelcons View

By Default, File Manager is rendered with largeicons view. The following example demonstrate this.

#### APP.VUE

```
<template>
  <div id="app">
    <ejs-filemanager id="file-manager" :ajaxSettings="ajaxSettings">
    </ejs-filemanager>
  </div>
</template>
<script>
import Vue from "vue";
import { FileManagerPlugin, DetailsView, NavigationPane, Toolbar } from
"@syncfusion/ej2-vue-filemanager";
Vue.use(FileManagerPlugin);
```

```

export default {
  data () {
    return {
      ajaxSettings:
      {
        url: "https://ej2-aspcore-
service.azurewebsites.net/api/FileManager/FileOperations",
        getImageUrl: "https://ej2-aspcore-
service.azurewebsites.net/api/FileManager/GetImage",
        uploadUrl: "https://ej2-aspcore-
service.azurewebsites.net/api/FileManager/Upload",
        downloadUrl: "https://ej2-aspcore-
service.azurewebsites.net/api/FileManager/Download"
      },
    };
  },
  provide: {
    filemanager: [DetailsView, NavigationPane, Toolbar]
  }
}
</script>
<style>
@import
"https://ej2.syncfusion.com/vue/documentation/node_modules/@syncfusion/ej2-
base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-icons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-inputs/styles/material.css";
@import "../node_modules/@syncfusion/ej2-popups/styles/material.css";
@import "../node_modules/@syncfusion/ej2-buttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-splitbuttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-navigations/styles/material.css";
@import "../node_modules/@syncfusion/ej2-layouts/styles/material.css";
@import "../node_modules/@syncfusion/ej2-grids/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-
filemanager/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/file-manager/detailsview-cs2" %}

### Details View

Details view is an injectable module in the file manager so, it should be injected before rendering the file manager to avail its functionality. The default appearance of the file manager can be changed from largeicons to details view by using the [view](#) property. The following example demonstrate the file manager with details view.

### APP.VUE

```

<template>
  <div id="app">
    <ejs-filemanager id="file-manager" :view="view"
:ajaxSettings="ajaxSettings">
      </ejs-filemanager>
    </div>
  </template>
<script>
import Vue from "vue";

```

```

import { FileManagerPlugin, DetailsView, NavigationPane, Toolbar } from
"@syncfusion/ej2-vue-filemanager";
Vue.use(FileManagerPlugin);
export default {
  data () {
    return {
      ajaxSettings:
      {
        url: "https://ej2-aspcore-
service.azurewebsites.net/api/FileManager/FileOperations",
        getImageUrl: "https://ej2-aspcore-
service.azurewebsites.net/api/FileManager/GetImage",
        uploadUrl: "https://ej2-aspcore-
service.azurewebsites.net/api/FileManager/Upload",
        downloadUrl: "https://ej2-aspcore-
service.azurewebsites.net/api/FileManager/Download"
      },
      view : "Details"
    };
  },
  provide: {
    filemanager: [DetailsView, NavigationPane, Toolbar]
  }
}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-icons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-inputs/styles/material.css";
@import "../node_modules/@syncfusion/ej2-popups/styles/material.css";
@import "../node_modules/@syncfusion/ej2-buttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-splitbuttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-navigations/styles/material.css";
@import "../node_modules/@syncfusion/ej2-layouts/styles/material.css";
@import "../node_modules/@syncfusion/ej2-grids/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-
filemanager/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/file-manager/detailsview-cs3" %}

### Customization in Vue File manager component

The file manager component allows customizing its functionalities like, context menu, searching, uploading, toolbar using APIs. Given below are some of the functionalities that can be customized in the File Manager,

- [Context menu customization](#)
- [Details view customization](#)
- [Navigation pane customization](#)
- [Show/Hide file extension](#)
- [Show/Hide hidden items](#)
- [Show/Hide thumbnail images in large icons view](#)
- [Toolbar customization](#)
- [Upload customization](#)

- [Tooltip customization](#)

### Context menu customization

The context menu settings like, items to be displayed on files, folders and layout click and visibility can be customized using [contextMenuSettings](#) property.

#### APP.VUE

```
<template>
  <div id="app">
    <ejs-filemanager id="file-manager"
      :contextMenuSettings="contextMenuSettings" :ajaxSettings="ajaxSettings">
      </ejs-filemanager>
    </div>
  </template>
<script>
import Vue from "vue";
import { FileManagerPlugin, DetailsView, NavigationPane, Toolbar } from
"@syncfusion/ej2-vue-filemanager";
Vue.use(FileManagerPlugin);
export default {
  data () {
    return {
      ajaxSettings:
      {
        url: "https://ej2-aspcore-
service.azurewebsites.net/api/FileManager/FileOperations",
        getImageUrl: "https://ej2-aspcore-
service.azurewebsites.net/api/FileManager/GetImage",
        uploadUrl: "https://ej2-aspcore-
service.azurewebsites.net/api/FileManager/Upload",
        downloadUrl: "https://ej2-aspcore-
service.azurewebsites.net/api/FileManager/Download"
      },
      // Context Menu settings customization
      contextMenuSettings: { file: ['Open', '|', 'Details'], folder:
['Open', '|', 'Details'], layout: ['SortBy', 'View', 'Refresh', '|',
'Details', '|'], visible: true}
    };
  },
  provide: {
    filemanager: [DetailsView, NavigationPane, Toolbar]
  }
}
</script>
<style>
@import
"https://ej2.syncfusion.com/vue/documentation/node_modules/@syncfusion/ej2-
base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-icons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-inputs/styles/material.css";
@import "../node_modules/@syncfusion/ej2-popups/styles/material.css";
@import "../node_modules/@syncfusion/ej2-buttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-splitbuttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-navigations/styles/material.css";
@import "../node_modules/@syncfusion/ej2-layouts/styles/material.css";
```

```
@import "../node_modules/@syncfusion/ej2-grids/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-
filemanager/styles/material.css";
</style>
```

{% previewsample "page.domainurl/code-snippet/file-manager/contextmenu-cs1" %}

### Details view customization

The details view settings like, column width, header text, template for each field can be customized using [detailsViewSettings](#) property.

### APP.VUE

```
<template>
  <div id="app">
    <ejs-filemanager id="file-manager" :view="view"
:detailsViewSettings="detailsViewSettings" :ajaxSettings="ajaxSettings">
      </ejs-filemanager>
    </div>
  </template>
<script>
import Vue from "vue";
import { FileManagerPlugin, DetailsView, NavigationPane, Toolbar } from
"@syncfusion/ej2-vue-filemanager";
Vue.use(FileManagerPlugin);
export default {
  data () {
    return {
      ajaxSettings:
      {
        url: "https://ej2-aspcore-
service.azurewebsites.net/api/FileManager/FileOperations",
        getImageUrl: "https://ej2-aspcore-
service.azurewebsites.net/api/FileManager/GetImage",
        uploadUrl: "https://ej2-aspcore-
service.azurewebsites.net/api/FileManager/Upload",
        downloadUrl: "https://ej2-aspcore-
service.azurewebsites.net/api/FileManager/Download"
      },
      // Initial view of File Manager is set to details view
      view: "Details",
      // Details View settings customization
      detailsViewSettings: {
        columns: [
          {field: 'name', headerText: 'File Name', minWidth: 120,
width: 'auto', customAttributes: { class: 'e-fe-grid-name' },template:
'${name}'},
          {field: 'size', headerText: 'File Size',minWidth: 50,
width: '110', template: '${size}'},
          { field: '_fm_modified', headerText: 'Date
Modified',minWidth: 50, width: '190'}
        ]
      }
    };
  },
  provide: {
```



```

        filemanager: [DetailsView, NavigationPane, Toolbar]
    }
}
</script>
<style>
@import
"https://ej2.syncfusion.com/vue/documentation/node_modules/@syncfusion/ej2-
base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-icons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-inputs/styles/material.css";
@import "../node_modules/@syncfusion/ej2-popups/styles/material.css";
@import "../node_modules/@syncfusion/ej2-buttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-splitbuttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-navigations/styles/material.css";
@import "../node_modules/@syncfusion/ej2-layouts/styles/material.css";
@import "../node_modules/@syncfusion/ej2-grids/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-
filemanager/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/file-manager/detailsview-cs1" %}

### Navigation pane customization

The navigation pane settings like, minimum and maximum width and visibility can be customized using [navigationPaneSettings](#) property.

### APP.VUE

```

<template>
  <div id="app">
    <ejs-filemanager id="file-manager"
      :navigationPaneSettings="navigationPaneSettings"
      :ajaxSettings="ajaxSettings">
      </ejs-filemanager>
    </div>
  </template>
<script>
import Vue from "vue";
import { FileManagerPlugin, DetailsView, NavigationPane, Toolbar } from
"@syncfusion/ej2-vue-filemanager";
Vue.use(FileManagerPlugin);
export default {
  data () {
    return {
      ajaxSettings:
      {
        url: "https://ej2-aspcore-
service.azurewebsites.net/api/FileManager/FileOperations",
        getImageUrl: "https://ej2-aspcore-
service.azurewebsites.net/api/FileManager/GetImage",
        uploadUrl: "https://ej2-aspcore-
service.azurewebsites.net/api/FileManager/Upload",
        downloadUrl: "https://ej2-aspcore-
service.azurewebsites.net/api/FileManager/Download"
      },
      // Navigation Pane settings customization
    }
  }
}

```

```

        navigationPaneSettings: { maxWidth: '850px', minWidth: '140px',
visible: true}
    };
    },
    provide: {
        filemanager: [DetailsView, NavigationPane, Toolbar]
    }
}
</script>
<style>
@import
"https://ej2.syncfusion.com/vue/documentation/node_modules/@syncfusion/ej2-
base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-icons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-inputs/styles/material.css";
@import "../node_modules/@syncfusion/ej2-popups/styles/material.css";
@import "../node_modules/@syncfusion/ej2-buttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-splitbuttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-navigations/styles/material.css";
@import "../node_modules/@syncfusion/ej2-layouts/styles/material.css";
@import "../node_modules/@syncfusion/ej2-grids/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-
filemanager/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/file-manager/navigationpane-cs1" %}

### Show/Hide file extension

The file extensions are displayed in the File Manager by default. This can be hidden by disabling the [showFileExtension](#) property.

In File Manager [fileLoad](#) and [fileOpen](#) events are triggered before the file/folder is rendered and before the file/folder is opened respectively. These events can be utilized to perform operations before a file/folder is rendered or opened.

### APP.VUE

```

<template>
  <div id="app">
    <ejs-filemanager id="file-manager"
:showFileExtension="showFileExtension" :ajaxSettings="ajaxSettings"
:fileLoad="onBeforeFileLoad" :fileOpen="onBeforeFileOpen">
      </ejs-filemanager>
    </div>
  </template>
<script>
import Vue from "vue";
import { FileManagerPlugin, DetailsView, NavigationPane, Toolbar } from
"@syncfusion/ej2-vue-filemanager";
Vue.use(FileManagerPlugin);
export default {
  data () {
    return {
      ajaxSettings:
        {

```

```

        url: "https://ej2-aspcore-
service.azurewebsites.net/api/FileManager/FileOperations",
        getImageUrl: "https://ej2-aspcore-
service.azurewebsites.net/api/FileManager/GetImage",
        uploadUrl: "https://ej2-aspcore-
service.azurewebsites.net/api/FileManager/Upload",
        downloadUrl: "https://ej2-aspcore-
service.azurewebsites.net/api/FileManager/Download"
    },
    // Hides the file extension in File Manager
    showFileExtension: false
  };
},
provide: {
  filemanager: [DetailsView, NavigationPane, Toolbar]
},
methods: {
  // File Manager's file beforeFileLoad function
  onBeforeFileLoad: function(args) {
    console.log(args.fileDetails.name + " is loading");
  },
  // File Manager's file beforeFileOpen function
  onBeforeFileOpen: function(args) {
    console.log(args.fileDetails.name + " is opened");
  }
}
}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-icons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-inputs/styles/material.css";
@import "../node_modules/@syncfusion/ej2-popups/styles/material.css";
@import "../node_modules/@syncfusion/ej2-buttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-splitbuttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-navigations/styles/material.css";
@import "../node_modules/@syncfusion/ej2-layouts/styles/material.css";
@import "../node_modules/@syncfusion/ej2-grids/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-
filemanager/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/file-manager/fileextension-cs1" %}

### Show/Hide hidden items

The File Manager provides support to show/hide the hidden items by enabling/disabling the [showHiddenItems](#) property.

### APP.VUE

```

<template>
  <div id="app">
    <ejs-filemanager id="file-manager"
:showHiddenItems="showHiddenItems" :ajaxSettings="ajaxSettings">
      </ejs-filemanager>
    </div>

```

```

</template>
<script>
import Vue from "vue";
import { FileManagerPlugin, DetailsView, NavigationPane, Toolbar } from
"@syncfusion/ej2-vue-filemanager";
Vue.use(FileManagerPlugin);
export default {
  data () {
    return {
      ajaxSettings:
      {
        url: "https://ej2-aspcore-
service.azurewebsites.net/api/FileManager/FileOperations",
        getImageUrl: "https://ej2-aspcore-
service.azurewebsites.net/api/FileManager/GetImage",
        uploadUrl: "https://ej2-aspcore-
service.azurewebsites.net/api/FileManager/Upload",
        downloadUrl: "https://ej2-aspcore-
service.azurewebsites.net/api/FileManager/Download"
      },
      // The default value set for showHiddenItems is false
      showHiddenItems: true
    };
  },
  provide: {
    filemanager: [DetailsView, NavigationPane, Toolbar]
  }
}
</script>
<style>
@import
"https://ej2.syncfusion.com/vue/documentation/node_modules/@syncfusion/ej2-
base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-icons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-inputs/styles/material.css";
@import "../node_modules/@syncfusion/ej2-popups/styles/material.css";
@import "../node_modules/@syncfusion/ej2-buttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-splitbuttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-navigations/styles/material.css";
@import "../node_modules/@syncfusion/ej2-layouts/styles/material.css";
@import "../node_modules/@syncfusion/ej2-grids/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-
filemanager/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/file-manager/hiddenitems-cs1" %}

### Show/Hide thumbnail images in large icons view

The thumbnail images are displayed in the File Manager's large icons view by default. This can be hidden by disabling the [showThumbnail](#) property.

### APP.VUE

```

<template>
  <div id="app">

```

```

    <ejs-filemanager id="file-manager" :showThumbnail="showThumbnail"
    :ajaxSettings="ajaxSettings">
      </ejs-filemanager>
    </div>
  </template>
  <script>
    import Vue from "vue";
    import { FileManagerPlugin, DetailsView, NavigationPane, Toolbar } from
    "@syncfusion/ej2-vue-filemanager";
    Vue.use(FileManagerPlugin);
    export default {
      data () {
        return {
          ajaxSettings:
            {
              url: "https://ej2-aspcore-
service.azurewebsites.net/api/FileManager/FileOperations",
              getImageUrl: "https://ej2-aspcore-
service.azurewebsites.net/api/FileManager/GetImage",
              uploadUrl: "https://ej2-aspcore-
service.azurewebsites.net/api/FileManager/Upload",
              downloadUrl: "https://ej2-aspcore-
service.azurewebsites.net/api/FileManager/Download"
            },
          // Hides the thumbnail images in File Manager's large icons view
          showThumbnail: false
        };
      },
      provide: {
        filemanager: [DetailsView, NavigationPane, Toolbar]
      }
    }
  </script>
  <style>
    @import
    "https://ej2.syncfusion.com/vue/documentation/node_modules/@syncfusion/ej2-
base/styles/material.css";
    @import "../node_modules/@syncfusion/ej2-icons/styles/material.css";
    @import "../node_modules/@syncfusion/ej2-inputs/styles/material.css";
    @import "../node_modules/@syncfusion/ej2-popups/styles/material.css";
    @import "../node_modules/@syncfusion/ej2-buttons/styles/material.css";
    @import "../node_modules/@syncfusion/ej2-splitbuttons/styles/material.css";
    @import "../node_modules/@syncfusion/ej2-navigations/styles/material.css";
    @import "../node_modules/@syncfusion/ej2-layouts/styles/material.css";
    @import "../node_modules/@syncfusion/ej2-grids/styles/material.css";
    @import "../node_modules/@syncfusion/ej2-vue-
filemanager/styles/material.css";
  </style>

```

{% previewsample "page.domainurl/code-snippet/file-manager/disablethumbnail-cs1" %}

### Toolbar customization

The toolbar settings like, items to be displayed in toolbar and visibility can be customized using [toolbarSettings](#) property.

### APP.VUE

```

<template>
  <div id="app">
    <ejs-filemanager id="file-manager" :ajaxSettings="ajaxSettings"
:toolbarSettings="toolbarSettings">
      </ejs-filemanager>
    </div>
  </template>
<script>
import Vue from "vue";
import { FileManagerPlugin, DetailsView, NavigationPane, Toolbar } from
"@syncfusion/ej2-vue-filemanager";
Vue.use(FileManagerPlugin);
export default {
  data () {
    return {
      ajaxSettings:
      {
        url: "https://ej2-aspcore-
service.azurewebsites.net/api/FileManager/FileOperations",
        getImageUrl: "https://ej2-aspcore-
service.azurewebsites.net/api/FileManager/GetImage",
        uploadUrl: "https://ej2-aspcore-
service.azurewebsites.net/api/FileManager/Upload",
        downloadUrl: "https://ej2-aspcore-
service.azurewebsites.net/api/FileManager/Download"
      },
      // Toolbar settings customization
      toolbarSettings: { items: ['NewFolder', 'Refresh', 'View',
'Details'], visible: true }
    };
  },
  provide: {
    filemanager: [DetailsView, NavigationPane, Toolbar]
  }
}
</script>
<style>
@import
"https://ej2.syncfusion.com/vue/documentation/node_modules/@syncfusion/ej2-
base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-icons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-inputs/styles/material.css";
@import "../node_modules/@syncfusion/ej2-popups/styles/material.css";
@import "../node_modules/@syncfusion/ej2-buttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-splitbuttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-navigations/styles/material.css";
@import "../node_modules/@syncfusion/ej2-layouts/styles/material.css";
@import "../node_modules/@syncfusion/ej2-grids/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-
filemanager/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/file-manager/toolbar-cs1" %}

[See Also](#)

- [How to add new items or customize default items](#)

### Upload customization

The upload settings like, minimum and maximum file size and enabling auto upload can be customized using [uploadSettings](#) property.

### APP.VUE

```
<template>
  <div id="app">
    <ejs-filemanager id="file-manager" :view="view"
    :ajaxSettings="ajaxSettings" :uploadSettings="uploadSettings">
      </ejs-filemanager>
    </div>
  </template>
<script>
import Vue from "vue";
import { FileManagerPlugin, DetailsView, NavigationPane, Toolbar } from
"@syncfusion/ej2-vue-filemanager";
Vue.use(FileManagerPlugin);
export default {
  data () {
    return {
      ajaxSettings:
      {
        url: "https://ej2-aspcore-
service.azurewebsites.net/api/FileManager/FileOperations",
        getImageUrl: "https://ej2-aspcore-
service.azurewebsites.net/api/FileManager/GetImage",
        uploadUrl: "https://ej2-aspcore-
service.azurewebsites.net/api/FileManager/Upload",
        downloadUrl: "https://ej2-aspcore-
service.azurewebsites.net/api/FileManager/Download"
      },
      // Initial view of File Manager is set to details view
      view: "Details",
      // Upload settings customization
      uploadSettings: { maxFileSize: 233332, minFileSize: 120,
autoUpload: true}
    };
  },
  provide: {
    filemanager: [DetailsView, NavigationPane, Toolbar]
  }
}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-icons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-inputs/styles/material.css";
@import "../node_modules/@syncfusion/ej2-popups/styles/material.css";
@import "../node_modules/@syncfusion/ej2-buttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-splitbuttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-navigations/styles/material.css";
```

```
@import "../node_modules/@syncfusion/ej2-layouts/styles/material.css";
@import "../node_modules/@syncfusion/ej2-grids/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-
filemanager/styles/material.css";
</style>
```

```
{% previewsample "page.domainurl/code-snippet/file-manager/upload-cs1" %}
```

### Tooltip customization

The tooltip value can be customized by adding extra content to the title of the toolbar, navigation pane, details view and large icons of the file manager element.

### APP.VUE

```
<template>
  <div class="control-section">
    <div class="filemanagerContainer">
      <!-- Filemanager element -->
      <ejs-filemanager id="file" ref="filemanagerObj"
:ajaxSettings="ajaxSettings" :fileLoad="fileLoad">
        </ejs-filemanager>
    </div>
  </div>
</template>
<script>
import Vue from "vue";
import { getValue, select } from '@syncfusion/ej2-base';
import { FileManagerPlugin, NavigationPane, Toolbar, DetailsView,
FileManagerComponent } from "@syncfusion/ej2-vue-filemanager";
Vue.use(FileManagerPlugin);
let hostUrl = 'https://ej2-aspcore-service.azurewebsites.net/';
export default {
  data () {
    return {
      ajaxSettings: {
        url: hostUrl + 'api/FileManager/FileOperations',
        getImageUrl: hostUrl + 'api/FileManager/GetImage',
        uploadUrl: hostUrl + 'api/FileManager/Upload',
        downloadUrl: hostUrl + 'api/FileManager/Download'
      }
    };
  },
  provide: {
    filemanager: [DetailsView, NavigationPane, Toolbar]
  },
  methods:{
    // File Manager's fileOpen event function
    fileLoad: function(args) {
      const target = args.element;
      if (args.module==='DetailsView') {
        const element = select('[title]', args.element);
        const title = getValue('name', args.fileDetails) +
          '\n' + getValue('dateModified', args.fileDetails);
        element.setAttribute('title', title);
      } else if (args.module==='LargeIconsView') {
        const title= getValue('name', args.fileDetails) +
```



```

        '\n' + getValue('dateModified', args.fileDetails);
        target.setAttribute('title', title);
    }
}
}
}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-icons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-inputs/styles/material.css";
@import "../node_modules/@syncfusion/ej2-popups/styles/material.css";
@import "../node_modules/@syncfusion/ej2-buttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-splitbuttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-navigations/styles/material.css";
@import "../node_modules/@syncfusion/ej2-layouts/styles/material.css";
@import "../node_modules/@syncfusion/ej2-grids/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-
filemanager/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/file-manager/tooltip-cs1" %}

### Multiple selection in Vue File manager component

The file manager allows you to select multiple files by enabling the [allowMultiSelection](#) property (enabled by default). The multiple selection can be done by pressing the **Ctrl** key or **Shift** key and selecting the files. The check box can also be used to do multiple selection. **Ctrl + A** can be used to select all files in the current directory. The [fileSelect](#) event will be triggered when the items of file manager control is selected or unselected.

#### APP.VUE

```

<template>
  <div id="app">
    <ejs-filemanager id="file-manager" :fileSelect="onFileSelect"
    :allowMultiSelection="allowMultiSelection" :ajaxSettings="ajaxSettings">
    </ejs-filemanager>
  </div>
</template>
<script>
import Vue from "vue";
import { FileManagerPlugin, DetailsView, NavigationPane, Toolbar } from
"@syncfusion/ej2-vue-filemanager";
Vue.use(FileManagerPlugin);
export default {
  data () {
    return {
      ajaxSettings:
      {
        url: "https://ej2-aspcore-
service.azurewebsites.net/api/FileManager/FileOperations",
        getImageUrl: "https://ej2-aspcore-
service.azurewebsites.net/api/FileManager/GetImage",
        uploadUrl: "https://ej2-aspcore-
service.azurewebsites.net/api/FileManager/Upload",

```

```

        downloadUrl: "https://ej2-aspcore-
service.azurewebsites.net/api/FileManager/Download"
    },
    // allowMultiSelection is true by default
    allowMultiSelection: true
  };
},
provide: {
  filemanager: [DetailsView, NavigationPane, Toolbar]
},
methods: {
  // File Manager's file select event function
  onFileSelect: function(args){
    console.log(args.fileDetails.name + " has been " + args.action +
"ed");
  }
}
});
</script>
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-icons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-inputs/styles/material.css";
@import "../node_modules/@syncfusion/ej2-popups/styles/material.css";
@import "../node_modules/@syncfusion/ej2-buttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-splitbuttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-navigations/styles/material.css";
@import "../node_modules/@syncfusion/ej2-layouts/styles/material.css";
@import "../node_modules/@syncfusion/ej2-grids/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-
filemanager/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/file-manager/multiselect-cs1" %}

Note: The File Manager has support to select files and folders initially or dynamically by specifying their names in [selectedItems](#) property.

### Drag and drop in Vue File manager component

The file manager allows files or folders to be moved from one folder to another by using the [allowDragAndDrop](#) property. It also supports uploading a file by dragging it from Windows Explorer to FileManager control. You can enable or disable this support by using the [allowDragAndDrop](#) property of file manager.

The event triggered in drag and drop support are

- [fileDragStart](#) - Triggers when the file/folder dragging is started.
- [fileDragging](#) - Triggers while dragging the file/folder.
- [fileDragStop](#) - Triggers when the file/folder is about to be dropped at the target.
- [fileDropped](#) - Triggers when the file/folder is dropped.

### APP.VUE

```
<template>
```

```

    <div id="app">
      <ejs-filemanager id="file-manager"
:allowDragAndDrop="allowDragAndDrop" :fileDragStart="onFileDragStart"
:fileDragStop="onFileDragStop" :fileDragging="onFileDragging"
:fileDropped="onFileDropped" :ajaxSettings="ajaxSettings">
        </ejs-filemanager>
      </div>
</template>
<script>
import Vue from "vue";
import { FileManagerPlugin, DetailsView, NavigationPane, Toolbar } from
"@syncfusion/ej2-vue-filemanager";
Vue.use(FileManagerPlugin);
export default {
  data () {
    return {
      ajaxSettings:
      {
        url: "https://ej2-aspcore-
service.azurewebsites.net/api/FileManager/FileOperations",
        getImageUrl: "https://ej2-aspcore-
service.azurewebsites.net/api/FileManager/GetImage",
        uploadUrl: "https://ej2-aspcore-
service.azurewebsites.net/api/FileManager/Upload",
        downloadUrl: "https://ej2-aspcore-
service.azurewebsites.net/api/FileManager/Download"
      },
      allowDragAndDrop: true
    };
  },
  provide: {
    filemanager: [DetailsView, NavigationPane, Toolbar]
  },
  methods: {
    // File Manager's file Drag start event function
    onFileDragStart: function(args){
      console.log("File Drag Start");
    },
    // File Manager's file Drag stop event function
    onFileDragStop: function(args){
      console.log("File Drag Stop");
    },
    // File Manager's file Dragging event function
    onFileDragging: function(args){
      console.log("File Dragging");
    },
    // File Manager's file Dropped event function
    onFileDropped: function(args){
      console.log("File Dropped");
    }
  }
}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-icons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-inputs/styles/material.css";

```

```
@import "../node_modules/@syncfusion/ej2-popups/styles/material.css";
@import "../node_modules/@syncfusion/ej2-buttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-splitbuttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-navigations/styles/material.css";
@import "../node_modules/@syncfusion/ej2-layouts/styles/material.css";
@import "../node_modules/@syncfusion/ej2-grids/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-filemanager/styles/material.css";
</style>
```

```
{% previewsample "page.domainurl/code-snippet/file-manager/drag-and-drop-cs1" %}
```

### File system provider in Vue File manager component

The file system provider allows the File Manager component to manage the files and folders in a physical or cloud-based file system. It provides the methods for performing various file actions like creating a new folder, copying and moving of files or folders, deleting uploading, and downloading the files or folders in the file system.

The following file providers are added in Syncfusion EJ2 File Manager component.

- [ASP.NET Core file system provider](#)
- [ASP.NET MVC 5 file system provider](#)
- [ASP.NET Core Azure cloud file system Provider](#)
- [ASP.NET MVC 5 Azure cloud file system Provider](#)
- [ASP.NET Core Amazon S3 cloud file provider](#)
- [ASP.NET MVC Amazon S3 cloud file provider](#)
- [File Transfer Protocol file system provider](#)
- [SQL database file system provider](#)
- [NodeJS file system provider](#)
- [Google Drive file system provider](#)
- [Firebase Realtime Database file system provider](#)
- [IBM Cloud Object Storage provider](#)

### ASP.NET Core file system provider

The ASP.NET Core file system provider allows the users to access and manage the physical file system. To get started, we need to clone the [ej2-aspcore-file-provider](#) using the following command.

```
`ts
```

```
git clone https://github.com/SyncfusionExamples/ej2-aspcore-file-provider ej2-aspcore-file-provider
cd ej2-aspcore-file-provider
```

After cloning, open the project in Visual Studio and restore the NuGet packages. Now, we need to set the root directory of the physical file system in the FileManager controller.

After setting the root directory of the file system, build and run the project. Now, the project will be hosted in `http://localhost:{port}` and mapping the ajaxSettings property of the FileManager component to the appropriate controller methods allows to manage the files in the physical file system.

```
<template>
<div id="app">
<ejs-filemanager id="file-manager" :ajaxSettings="ajaxSettings">
</ejs-filemanager>
</div>
</template>
<script>
import Vue from "vue";

import { FileManagerPlugin, DetailsView, NavigationPane, Toolbar } from "@syncfusion/ej2-vue-filemanager";

Vue.use(FileManagerPlugin);

export default {
  data () {
    return {
      // Initializing File Manager ASP.NET Core service.
      ajaxSettings:
      { // Replace the hosted port number in the place of "{port}"
        url: "http://localhost:{port}/api/FileManager/FileOperations",
        getImageUrl: "http://localhost:{port}/api/FileManager/GetImage",
        uploadUrl: "http://localhost:{port}/api/FileManager/Upload",
        downloadUrl: "http://localhost:{port}/api/FileManager/Download"
      }
    };
  },
  provide: {
    filemanager: [DetailsView, NavigationPane, Toolbar]
  }
}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-icons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-inputs/styles/material.css";
```

```

@import "../node_modules/@syncfusion/ej2-popups/styles/material.css";
@import "../node_modules/@syncfusion/ej2-buttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-splitbuttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-navigations/styles/material.css";
@import "../node_modules/@syncfusion/ej2-layouts/styles/material.css";
@import "../node_modules/@syncfusion/ej2-grids/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-filemanager/styles/material.css";
</style>
`

```

Note: To learn more about file actions that can be performed with ASP.NET Core file system provider, refer to this [link](#)

#### ASP.NET MVC 5 file system provider

The ASP.NET MVC5 file system provider allows the users to access and manage the physical file system. To get started, we need to clone the [ej2-aspmvc-file-provider](#) using the following command.

```

`ts
git clone https://github.com/SyncfusionExamples/ej2-aspmvc-file-provider ej2-aspmvc-file-provider
cd ej2-aspmvc-file-provider
`

```

After cloning, open the project in Visual Studio and restore the NuGet packages. Now, we need to set the root directory of the physical file system in the FileManager controller using the Root Folder method.

After setting the root directory of the file system, build and run the project. Now, the project will be hosted in `http://localhost:{port}` and mapping the ajaxSettings property of the FileManager component to the appropriate controller methods allows to manage the files in the physical file system.

```

<template>
<div id="app">
<ejs-filemanager id="file-manager" :ajaxSettings="ajaxSettings">
</ejs-filemanager>
</div>
</template>
<script>
import Vue from "vue";

import { FileManagerPlugin, DetailsView, NavigationPane, Toolbar } from "@syncfusion/ej2-vue-filemanager";

```

```
Vue.use(FileManagerPlugin);
export default {
  data () {
    return {
      // Initializing File Manager ASP.NET MVC service.
      ajaxSettings:
      {
        // Replace the hosted port number in the place of "{port}"
        url: "http://localhost:{port}/FileManager/FileOperations",
        getImageUrl: "http://localhost:{port}/FileManager/GetImage",
        uploadUrl: "http://localhost:{port}/FileManager/Upload",
        downloadUrl: "http://localhost:{port}/FileManager/Download"
      }
    };
  },
  provide: {
    filemanager: [DetailsView, NavigationPane, Toolbar]
  }
}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-icons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-inputs/styles/material.css";
@import "../node_modules/@syncfusion/ej2-popups/styles/material.css";
@import "../node_modules/@syncfusion/ej2-buttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-splitbuttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-navigations/styles/material.css";
@import "../node_modules/@syncfusion/ej2-layouts/styles/material.css";
@import "../node_modules/@syncfusion/ej2-grids/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-filemanager/styles/material.css";
</style>
`
```

**Note:** To learn more about file actions that can be performed with ASP.NET MVC 5 file system provider, refer to this [link](#)

#### ASP.NET Core Azure cloud file system Provider

In ASP.NET Core, Azure file system provider allows the users to access and manage the blobs in the Azure blob storage. To get started, we need to clone the [azure-aspcore-file-provider](#) using the following command

```
`ts
git clone https://github.com/SyncfusionExamples/azure-aspcore-file-provider azure-aspcore-file-provider
`
```

After cloning, open the project in Visual Studio and restore the NuGet packages. Now, we need to register the Azure storage by passing details like name, password, and blob name to the Register Azure method in the FileManager controller.

```
`ts
void RegisterAzure(string accountName, string accountKey, string blobName)
`
```

Then, set the blob container and the root blob directory by passing the corresponding URLs as parameters in the setBlobContainer method as follows.

```
`ts
void setBlobContainer( Blob-contatiner-url, Root-blob-directory-url)
`
```

**Note:** Also, assign the same *blobPath URL* and *filePath URL* in [AzureFileOperations](#) and [AzureUpload](#) methods in the FileManager controller to determine the original path of the Azure blob.

After setting the blob container references, just build and run the project. Now, the project will be hosted in `http://localhost:{port}` and just mapping the **ajaxSettings** property of the FileManager component to the appropriate controller methods allows to manage the Azure blob storage.

```
<template>
<div id="app">
<ejs-filemanager id="file-manager" :ajaxSettings="ajaxSettings">
</ejs-filemanager>
</div>
</template>
<script>
import Vue from "vue";
import { FileManagerPlugin, DetailsView, NavigationPane, Toolbar } from "@syncfusion/ej2-vue-filemanager";
```



```
Vue.use(FileManagerPlugin);
export default {
  data () {
    return {
      // Initializing File Manager Azure cloud file system service.
      ajaxSettings:
      {
        // Replace the hosted port number in the place of "{port}"
        url: "http://localhost:{port}/api/AzureProvider/AzureFileOperations",
        getImageUrl: "http://localhost:{port}/api/AzureProvider/AzureGetImage",
        uploadUrl: "http://localhost:{port}/api/AzureProvider/AzureUpload",
        downloadUrl: "http://localhost:{port}/api/AzureProvider/AzureDownload"
      }
    };
  },
  provide: {
    filemanager: [DetailsView, NavigationPane, Toolbar]
  }
}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-icons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-inputs/styles/material.css";
@import "../node_modules/@syncfusion/ej2-popups/styles/material.css";
@import "../node_modules/@syncfusion/ej2-buttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-splitbuttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-navigations/styles/material.css";
@import "../node_modules/@syncfusion/ej2-layouts/styles/material.css";
@import "../node_modules/@syncfusion/ej2-grids/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-filemanager/styles/material.css";
</style>
`
```

**Note:** To learn more about file actions that can be performed with ASP.NET Core Azure cloud file system Provider, refer to this [link](#)

#### ASP.NET MVC Azure cloud file system Provider

In ASP.NET MVC, Azure file system provider allows the users to access and manage the blobs in the Azure blob storage. To get started, we need to clone the [ej2-azure-aspmvc-file-provider](#) using the following command

```
`ts
git clone https://github.com/SyncfusionExamples/ej2-azure-aspmvc-file-provider ej2-azure-aspmvc-file-provider
`
```

After cloning, open the project in Visual Studio and restore the NuGet packages. Now, we need to register the Azure storage by passing details like name, password, and blob name to the Register Azure method in the FileManager controller.

```
`ts
void RegisterAzure(string accountName, string accountKey, string blobName)
`
```

Then, set the blob container and the root blob directory by passing the corresponding URLs as parameters in the setBlobContainer method as follows.

```
`ts
void setBlobContainer( Blob-container-url, Root-blob-directory-url)
`
```

**Note:** Also, assign the same *blobPath URL* and *filePath URL* in [AzureFileOperations](#) and [AzureUpload](#) methods in the FileManager controller to determine the original path of the Azure blob.

After setting the blob container references, just build and run the project. Now, the project will be hosted in `http://localhost:{port}` and just mapping the **ajaxSettings** property of the FileManager component to the appropriate controller methods allows to manage the Azure blob storage.

```
<template>
<div id="app">
<ejs-filemanager id="file-manager" :ajaxSettings="ajaxSettings">
</ejs-filemanager>
</div>
</template>
<script>
import Vue from "vue";
import { FileManagerPlugin, DetailsView, NavigationPane, Toolbar } from "@syncfusion/ej2-vue-filemanager";
```

```
Vue.use(FileManagerPlugin);
export default {
  data () {
    return {
      // Initializing File Manager Azure cloud file system service.
      ajaxSettings:
      {
        // Replace the hosted port number in the place of "{port}"
        url: "http://localhost:{port}/AzureProvider/AzureFileOperations",
        getImageUrl: "http://localhost:{port}/AzureProvider/AzureGetImage",
        uploadUrl: "http://localhost:{port}/AzureProvider/AzureUpload",
        downloadUrl: "http://localhost:{port}/AzureProvider/AzureDownload"
      }
    };
  },
  provide: {
    filemanager: [DetailsView, NavigationPane, Toolbar]
  }
}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-icons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-inputs/styles/material.css";
@import "../node_modules/@syncfusion/ej2-popups/styles/material.css";
@import "../node_modules/@syncfusion/ej2-buttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-splitbuttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-navigations/styles/material.css";
@import "../node_modules/@syncfusion/ej2-layouts/styles/material.css";
@import "../node_modules/@syncfusion/ej2-grids/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-filemanager/styles/material.css";
</style>
`
```

**Note:** To learn more about file actions that can be performed with ASP.NET MVC Azure cloud file system Provider, refer to this [link](#)

#### ASP.NET Core Amazon S3 cloud file provider

In ASP.NET Core, Amazon **S3** (*Simple Storage Service*) cloud file provider allows the users to access and manage a server hosted file system as collection of objects stored in the Amazon S3 Bucket. To get started, clone the [amazon-s3-aspcore-file-provider](#) using the following command

```
`ts
git clone https://github.com/SyncfusionExamples/amazon-s3-aspcore-file-provider.git amazon-s3-
aspcore-file-provider.git
`
```

**Note:** To learn more about creating and configuring an Amazon S3 bucket, refer to this [link](#).

After cloning, open the project in Visual Studio and restore the NuGet packages. Now, register Amazon S3 client account details like *awsAccessKeyId*, *awsSecretKeyId* and *awsRegion* details in **RegisterAmazonS3** method in the FileManager controller to perform the file operations.

```
`ts
void RegisterAmazonS3(string bucketName, string awsAccessKeyId, string awsSecretAccessKey, string
bucketRegion)
`
```

After registering the Amazon client account details, build and run the project. Now, the project will be hosted in `http://localhost:{port}` and mapping the **ajaxSettings** property of the FileManager component to the appropriate controller methods allows to manage the Amazon **S3** (*Simple Storage Service*) bucket's objects storage.

```
<template>
<div id="app">
<ejs-filemanager id="file-manager" :ajaxSettings="ajaxSettings">
</ejs-filemanager>
</div>
</template>
<script>
import Vue from "vue";

import { FileManagerPlugin, DetailsView, NavigationPane, Toolbar } from "@syncfusion/ej2-vue-
filemanager";

Vue.use(FileManagerPlugin);

export default {
data () {
return {
```

```
// Initializing File Manager Amazon S3 file system service.
ajaxSettings:
{
  // Replace the hosted port number in the place of "{port}"
  url: "http://localhost:{port}/api/AmazonS3Provider/AmazonS3FileOperations",
  getImageUrl: "http://localhost:{port}/api/AmazonS3Provider/AmazonS3GetImage",
  uploadUrl: "http://localhost:{port}/api/AmazonS3Provider/AmazonS3Upload",
  downloadUrl: "http://localhost:{port}/api/AmazonS3Provider/AmazonS3Download"
}
};
},
provide: {
  filemanager: [DetailsView, NavigationPane, Toolbar]
}
}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-icons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-inputs/styles/material.css";
@import "../node_modules/@syncfusion/ej2-popups/styles/material.css";
@import "../node_modules/@syncfusion/ej2-buttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-splitbuttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-navigations/styles/material.css";
@import "../node_modules/@syncfusion/ej2-layouts/styles/material.css";
@import "../node_modules/@syncfusion/ej2-grids/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-filemanager/styles/material.css";
</style>
`
```

**Note:** To learn more about the file actions that can be performed with Amazon S3 Cloud File provider, refer to this [link](#)

### ASP.NET MVC Amazon S3 cloud file provider

In ASP.NET MVC, Amazon **S3** (*Simple Storage Service*) cloud file provider allows the users to access and manage a server hosted files as collection of objects stored in the Amazon S3 Bucket. To get started, clone the [ej2-amazon-s3-aspmvc-file-provider](https://github.com/SyncfusionExamples/ej2-amazon-s3-aspmvc-file-provider) using the following command

```
`ts
git clone https://github.com/SyncfusionExamples/ej2-amazon-s3-aspmvc-file-provider.git ej2-amazon-
s3-aspmvc-file-provider.git
`
```

**Note:** To learn more about creating and configuring an Amazon S3 bucket, refer to this [link](#).

After cloning, open the project in Visual Studio and restore the NuGet packages. Now, register Amazon S3 client account details like *awsAccessKeyId*, *awsSecretKeyId* and *awsRegion* details in **RegisterAmazonS3** method in the FileManager controller to perform the file operations.

```
`ts
void RegisterAmazonS3(string bucketName, string awsAccessKeyId, string awsSecretAccessKey, string
bucketRegion)
`
```

After registering the Amazon client account details, just build and run the project. Now, the project will be hosted in `http://localhost:{port}` and just mapping the **ajaxSettings** property of the FileManager component to the appropriate controller methods allows to manage the Amazon **S3** (*Simple Storage Service*) bucket's objects storage.

```
<template>
<div id="app">
<ejs-filemanager id="file-manager" :ajaxSettings="ajaxSettings">
</ejs-filemanager>
</div>
</template>
<script>
import Vue from "vue";

import { FileManagerPlugin, DetailsView, NavigationPane, Toolbar } from "@syncfusion/ej2-vue-
filemanager";

Vue.use(FileManagerPlugin);

export default {
  data () {
    return {
      // Initializing File Manager Amazon S3 file service.
```

```

ajaxSettings:
{
  // Replace the hosted port number in the place of "{port}"
  url: "http://localhost:{port}/FileManager/FileOperations",
  getImageUrl: "http://localhost:{port}/FileManager/GetImage",
  uploadUrl: "http://localhost:{port}/FileManager/Upload",
  downloadUrl: "http://localhost:{port}/FileManager/Download"
}
};
},
provide: {
  filemanager: [DetailsView, NavigationPane, Toolbar]
}
}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-icons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-inputs/styles/material.css";
@import "../node_modules/@syncfusion/ej2-popups/styles/material.css";
@import "../node_modules/@syncfusion/ej2-buttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-splitbuttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-navigations/styles/material.css";
@import "../node_modules/@syncfusion/ej2-layouts/styles/material.css";
@import "../node_modules/@syncfusion/ej2-grids/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-filemanager/styles/material.css";
</style>
`

```

**Note:** To learn more about the file actions that can be performed with ASP.NET MVC Amazon S3 Cloud File Provider, refer to this [link](#)

#### File Transfer Protocol file system provider

In ASP.NET Core, File Transfer Protocol file system provider allows the users to access to the hosted file system as collection of objects stored in the file storage using File Transfer Protocol. To get started, clone the [ftp-aspcore-file-provider](#) using the following command

```
`ts
```

```
git clone https://github.com/SyncfusionExamples/ftp-aspcore-file-provider.git ftp-aspcore-file-provider.git
```

```
,
```

After cloning, open the project in Visual Studio and restore the NuGet packages. Now, register File Transfer Protocol details like *hostName*, *userName* and *password* in **SetFTPConnection** method in the FileManager controller to perform the file operations.

```
`ts
```

```
void SetFTPConnection(string hostName, string userName, string password)
```

```
,
```

After registering the File Transfer Protocol details, just build and run the project. Now, the project will be hosted in `http://localhost:{port}` and mapping the **ajaxSettings** property of the FileManager component to the appropriate controller methods allows you to manage the FTP's objects storage.

```
,
```

```
<template>
```

```
<div id="app">
```

```
<ejs-filemanager id="file-manager" :ajaxSettings="ajaxSettings">
```

```
</ejs-filemanager>
```

```
</div>
```

```
</template>
```

```
<script>
```

```
import Vue from "vue";
```

```
import { FileManagerPlugin, DetailsView, NavigationPane, Toolbar } from "@syncfusion/ej2-vue-filemanager";
```

```
Vue.use(FileManagerPlugin);
```

```
export default {
```

```
  data () {
```

```
    return {
```

```
    // Initializing File Manager file transfer protocol file system service.
```

```
    ajaxSettings:
```

```
    {
```

```
    // Replace the hosted port number in the place of "{port}"
```

```
    url: "http://localhost:{port}/api/FTPProvider/FTPFileOperations",
```

```
    getImageUrl: "http://localhost:{port}/api/FTPProvider/FTPGetImage",
```

```
    uploadUrl: "http://localhost:{port}/api/FTPProvider/FTPUUpload",
```



```

downloadUrl: "http://localhost:{port}/api/FTPProvider/FTPDownload"
}
};
},
provide: {
  filemanager: [DetailsView, NavigationPane, Toolbar]
}
}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-icons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-inputs/styles/material.css";
@import "../node_modules/@syncfusion/ej2-popups/styles/material.css";
@import "../node_modules/@syncfusion/ej2-buttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-splitbuttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-navigations/styles/material.css";
@import "../node_modules/@syncfusion/ej2-layouts/styles/material.css";
@import "../node_modules/@syncfusion/ej2-grids/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-filemanager/styles/material.css";
</style>
`

```

**Note:** To learn more about the file actions that can be performed with File Transfer Protocol file system provider, refer to this [link](#)

### SQL database file system provider

In ASP.NET Core, SQL database file system provider allows the users to manage the file system being maintained in a SQL database table. Unlike the other file system providers, the SQL database file system provider works on ID basis. Here, each file and folder have a unique ID based on which all the file operations will be performed. To get started, we need to clone the [sql-server-database-aspcore-file-provider](#) using the following command.

```

`ts
<add name="FileExplorerConnection" connectionString="Data
Source=(LocalDB)\v11.0;AttachDbFilename=|DataDirectory|\FileManager.mdf;Integrated
Security=True;Trusted_Connection=true" />
`

```

After cloning, open the project in Visual Studio and restore the NuGet packages. To establish the SQL server connection with the database file (for eg: FileManager.mdf), we need to specify the connection string in the web config file as follows.

```
`ts
<add name="FileExplorerConnection" connectionString="Data
Source=(LocalDB)\v11.0;AttachDbFilename=|DataDirectory|\FileManager.mdf;Integrated
Security=True;Trusted_Connection=true" />
`
```

Then, make an entry for the connection string in `appsettings.json` file as follows.

```
`ts
"ConnectionStrings": {
  "FileManagerConnection": "Data
Source=(LocalDB)\MSSQLLocalDB;AttachDbFilename=|DataDirectory|\\App_Data\\FileManager.mdf;In
tegrated Security=True;Connect Timeout=30"
}
`
```

Now, to configure the database connection, we need to set the connection name, table name and root folder ID value by passing these values to the SetSQLConnection method.

```
`ts
void SetSQLConnection(string name, string tableName, string tableID)
`
```

Refer to this [FileManager.mdf](#), to learn about the pre-defined file system SQL database for the EJ2 File Manager.

After configuring the connection, build and run the project. Now, the project will be hosted in `http://localhost:{port}` and mapping the `ajaxSettings` property of the FileManager component to the appropriate controller methods allows to manage the files in the SQL database table.

```
`
<template>
<div id="app">
<ejs-filemanager id="file-manager" :ajaxSettings="ajaxSettings">
</ejs-filemanager>
</div>
</template>
<script>
import Vue from "vue";
```

```
import { FileManagerPlugin, DetailsView, NavigationPane, Toolbar } from "@syncfusion/ej2-vue-filemanager";

Vue.use(FileManagerPlugin);

export default {
  data () {
    return {
      // Initializing File Manager SQL database file system service.
      ajaxSettings:
      {
        // Replace the hosted port number in the place of "{port}"
        url: "http://localhost:{port}/api/SQLProvider/SQLFileOperations",
        getImageUrl: "http://localhost:{port}/api/SQLProvider/SQLGetImage",
        uploadUrl: "http://localhost:{port}/api/SQLProvider/SQLUpload",
        downloadUrl: "http://localhost:{port}/api/SQLProvider/SQLDownload"
      }
    };
  },
  provide: {
    filemanager: [DetailsView, NavigationPane, Toolbar]
  }
}

</script>

<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-icons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-inputs/styles/material.css";
@import "../node_modules/@syncfusion/ej2-popups/styles/material.css";
@import "../node_modules/@syncfusion/ej2-buttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-splitbuttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-navigations/styles/material.css";
@import "../node_modules/@syncfusion/ej2-layouts/styles/material.css";
@import "../node_modules/@syncfusion/ej2-grids/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-filemanager/styles/material.css";
```

</style>

,

**Note:** To learn more about file actions that can be performed with SQL database file system provider, refer to this [link](#)

### Google Drive file system provider

In ASP.NET Core, Google Drive file system provider allows the users to manage the files and folders in a Google Drive account. The Google Drive file system provider works on id basis where each file and folder have a unique ID. To get started, we need to clone the [google-drive-aspcore-file-provider](#) using the following command.

`ts

```
git clone https://github.com/SyncfusionExamples/google-drive-aspcore-file-provider google-drive-aspcore-file-provider
```

```
cd google-drive-aspcore-file-provider
```

,

Google Drive file system provider use the [Google Drive APIs](#) to read the file in the file system and uses the [OAuth 2.0](#) protocol for authentication and authorization. To authenticate from the client end, have to obtain OAuth 2.0 client credentials from the [Google API Console](#). To learn more about generating the client credentials from the from Google API Console, refer to this [link](#).

After generating the client secret data, copy the JSON data to the following specified JSON files in the cloned location.

- EJ2GoogleDriveFileProvider > credentials > client\_secret.json
- GoogleOAuth2.0Base > credentials > client\_secret.json

After updating the credentials, build and run the project. Now, the project will be hosted in <http://localhost:{port}>, and it will ask to log on to the Gmail account created the client secret credentials. Then, provide permission to access the Google Drive files by clicking the allow access button in the page. Now, mapping the ajaxSettings property of the FileManager component to the appropriate controller methods will allows to manage the files from the Google Drive.

,

<template>

<div id="app">

<ejs-filemanager id="file-manager" :ajaxSettings="ajaxSettings">

</ejs-filemanager>

</div>

</template>

<script>

import Vue from "vue";

```
import { FileManagerPlugin, DetailsView, NavigationPane, Toolbar } from "@syncfusion/ej2-vue-filemanager";

Vue.use(FileManagerPlugin);

export default {
  data () {
    return {
      // Initializing File Manager Google Drive file system service.
      ajaxSettings:
      {
        // Replace the hosted port number in the place of "{port}"
        url: "http://localhost:{port}/api/GoogleDriveProvider/GoogleDriveFileOperations",
        getImageUrl: "http://localhost:{port}/api/GoogleDriveProvider/GoogleDriveGetImage",
        uploadUrl: "http://localhost:{port}/api/GoogleDriveProvider/GoogleDriveUpload",
        downloadUrl: "http://localhost:{port}/api/GoogleDriveProvider/GoogleDriveDownload"
      }
    };
  },
  provide: {
    filemanager: [DetailsView, NavigationPane, Toolbar]
  }
}

</script>

<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-icons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-inputs/styles/material.css";
@import "../node_modules/@syncfusion/ej2-popups/styles/material.css";
@import "../node_modules/@syncfusion/ej2-buttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-splitbuttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-navigations/styles/material.css";
@import "../node_modules/@syncfusion/ej2-layouts/styles/material.css";
@import "../node_modules/@syncfusion/ej2-grids/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-filemanager/styles/material.css";
```

</style>

**Note:** To learn more about file actions that can be performed with Google drive file system provider, refer to this [link](#)

### NodeJS file system provider

In ASP.NET Core, NodeJS file system provider allows the users to manage the files and folders in a physical file system. It provides methods for performing all basic file operations like creating a folder, copy, move, delete, and download files and folders in the file system. We can use of the NodeJS file system provider either by installing the [ej2-filemanager-node-filesystem](#) package or by cloning the [file system provider](#) from the GitHub.

#### Using ej2-filemanager-node-filesystem package

- Install the ej2-filemanager-node-filesystem package by running the below command.

```
`ts
```

```
npm install @syncfusion/ej2-filemanager-node-filesystem
```

- After installing the package, navigate to the ej2-filemanager-node-filesystem package folder within the node-modules
- Run the command **npm install** command.

#### Cloning the ej2-filemanager-node-filesystem from GitHub

- Clone the ej2-filemanager-node-filesystem using the following command.

```
`ts
```

```
git clone https://github.com/SyncfusionExamples/ej2-filemanager-node-filesystem.git node-filesystem-provider
```

- After cloning, open the root folder and run the command **npm install** command.

After installing the packages, set the root folder directory of the physical file system in the package JSON under scripts sections as follows.

```
`ts
```

```
"start": "node filesystem-server.js -d D:/Projects"
```

**Note:** By default, the root directory will be configured to set **C:/Users** as the root directory.

To set the port in which the project to be hosted and the root directory of the file system. Run the following command.

```
`ts
set PORT=3000 && node filesystem-server.js -d D:/Projects
`
```

**Note:** By default, the service will run 8090 port.

Now, mapping the ajaxSettings property of the FileManager component to the appropriate file operation methods in the filesystem-server.js file will allow to manage the physical file system with NodeJS file system provider.

```
<template>
<div id="app">
  <ejs-filemanager id="file-manager" :ajaxSettings="ajaxSettings">
</ejs-filemanager>
</div>
</template>
<script>
import Vue from "vue";

import { FileManagerPlugin, DetailsView, NavigationPane, Toolbar } from "@syncfusion/ej2-vue-filemanager";

Vue.use(FileManagerPlugin);

export default {
  data () {
    return {
      // Initializing File Manager NodeJS file system service.
      ajaxSettings:
      {
        // Replace the hosted port number in the place of "{port}"
        url: "http://localhost:{port}/",
        getImageUrl: "http://localhost:{port}/GetImage",
        uploadUrl: "http://localhost:{port}/Upload",
        downloadUrl: "http://localhost:{port}/Download"
      }
    };
  },
  provide: {
```

```
filemanager: [DetailsView, NavigationPane, Toolbar]
}
}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-icons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-inputs/styles/material.css";
@import "../node_modules/@syncfusion/ej2-popups/styles/material.css";
@import "../node_modules/@syncfusion/ej2-buttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-splitbuttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-navigations/styles/material.css";
@import "../node_modules/@syncfusion/ej2-layouts/styles/material.css";
@import "../node_modules/@syncfusion/ej2-grids/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-filemanager/styles/material.css";
</style>
`
```

**Note:** To learn more about file actions that can be performed with NodeJS file system provider, refer to this [link](#)

#### Firebase Realtime Database file system provider

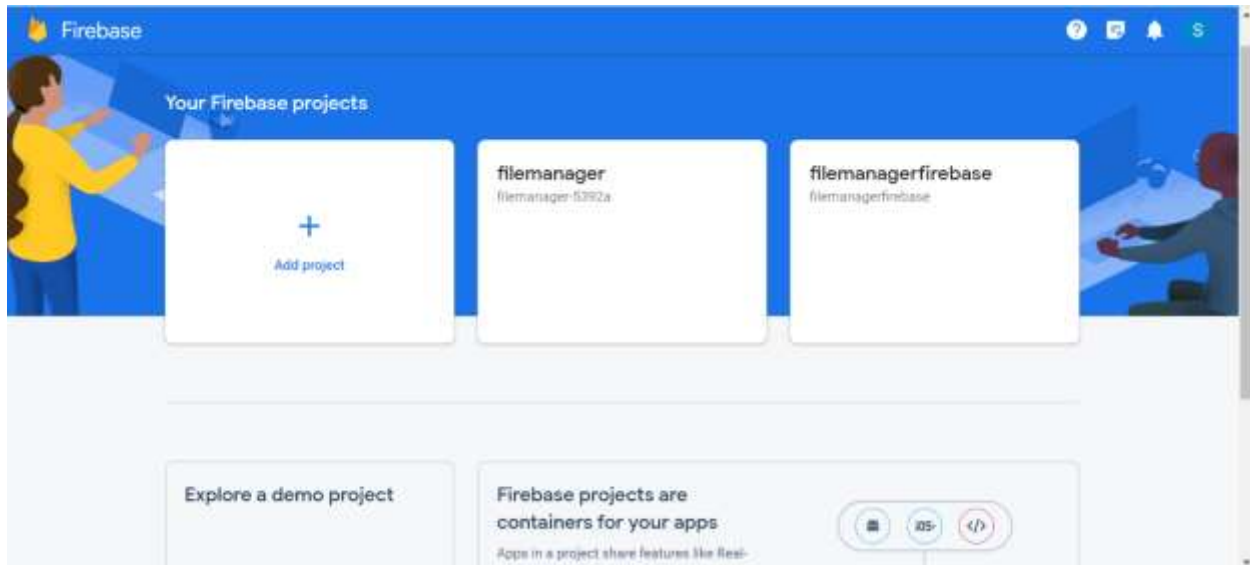
The [Firebase Realtime Database](#) file system provider in **ASP.NET Core** provides the efficient way to store the File Manager file system in a cloud database as JSON representation.

#### *Generate Secret access key from service account*

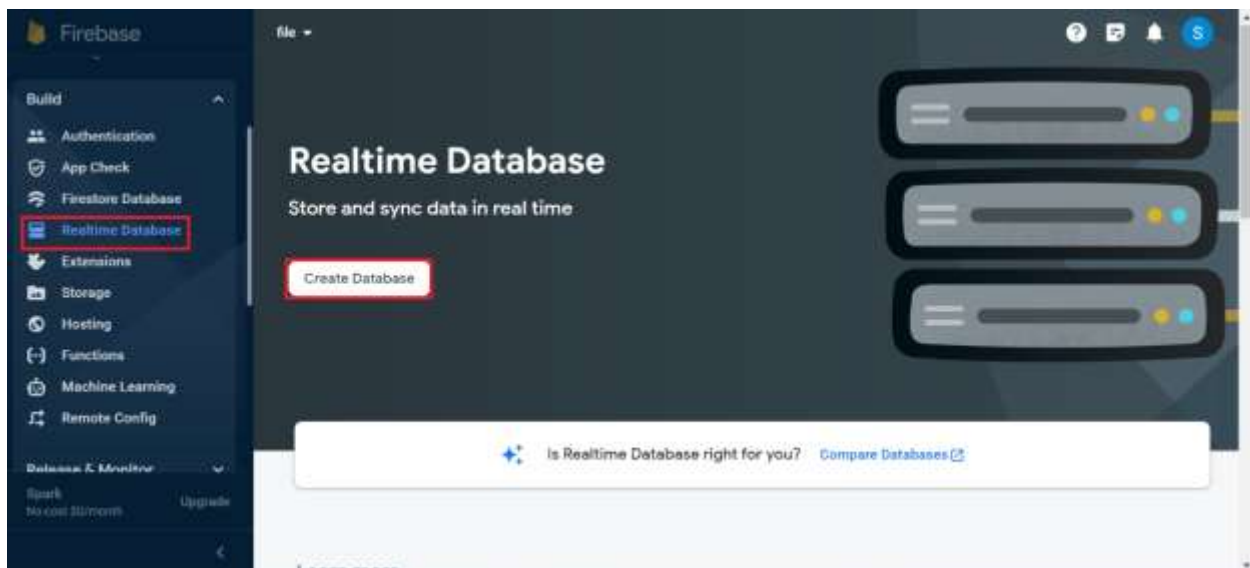
Follow the given steps to generate the secret access key:

- To access the Firebase console, please click on this [link](#). Once you have accessed the console, you can create a new project by filling in the necessary fields and clicking on the relevant buttons.





- Within the Firebase console, navigate to the **Build** tab. Under this tab, select the option for **Realtime Database**. From there, you can create a new database by clicking on the **Create Database** button.



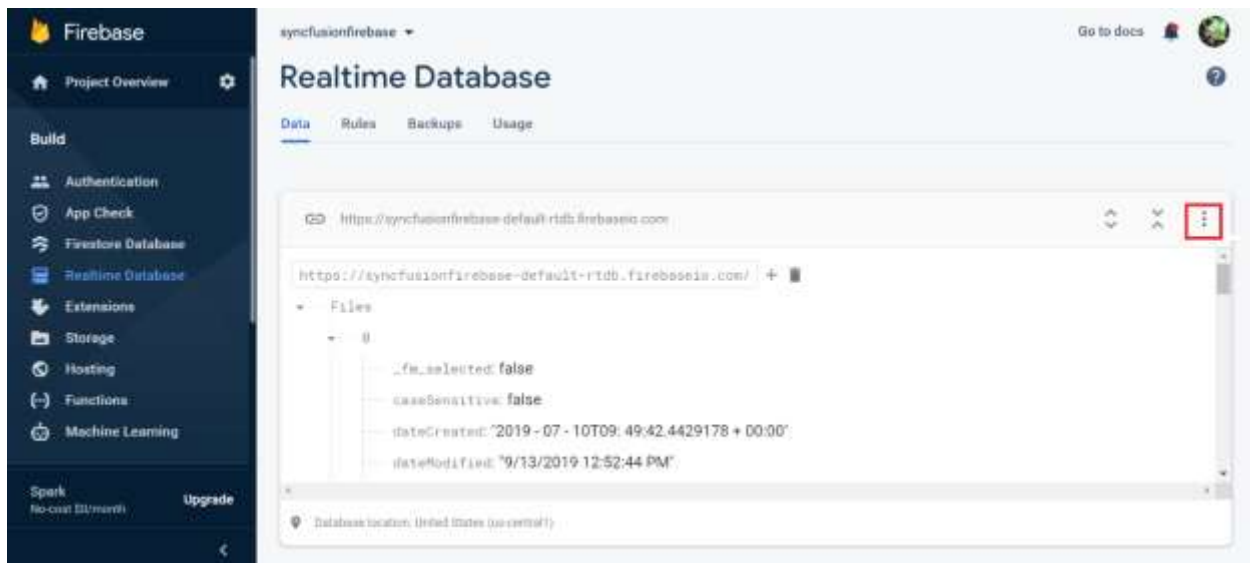
- To get started, create a root node and add any desired children to it. Please refer to the following code snippet for guidance on the structure of the JSON:

```
`ts
{
  "Files" : [ {
    "caseSensitive" : false,
    "dateCreated" : "8/22/2019 5:17:55 PM",
```

```
"dateModified" : "8/22/2019 5:17:55 PM",
"filterId" : "0/",
"filterPath" : "/",
"hasChild" : false,
"id" : "5",
"isFile" : false,
"isRoot" : true,
"name" : "Music",
"parentId" : "0",
"selected" : false,
"showHiddenItems" : false,
"size" : 0,
"type" : "folder"
},
{
  "caseSensitive" : false,
  "dateCreated" : "8/22/2019 5:18:03 PM",
  "dateModified" : "8/22/2019 5:18:03 PM",
  "filterId" : "0/",
  "filterPath" : "/",
  "hasChild" : false,
  "id" : "6",
  "isFile" : false,
  "isRoot" : true,
  "name" : "videos",
  "parentId" : "0",
  "selected" : false,
  "showHiddenItems" : false,
  "size" : 0,
  "type" : ""
}]
}
```

Here, the `Files` denotes the `rootNode` and the subsequent object refers to the children of the root node. `rootNode` will be taken as the root folder of the file system loaded which will be loaded in File Manager component.

- To import a JSON file into the Firebase Realtime Database, navigate to the **Data** tab and click on the action icon shown in the accompanying image. From there, select the **Import JSON** option and upload the JSON file that was created using the code provided above.

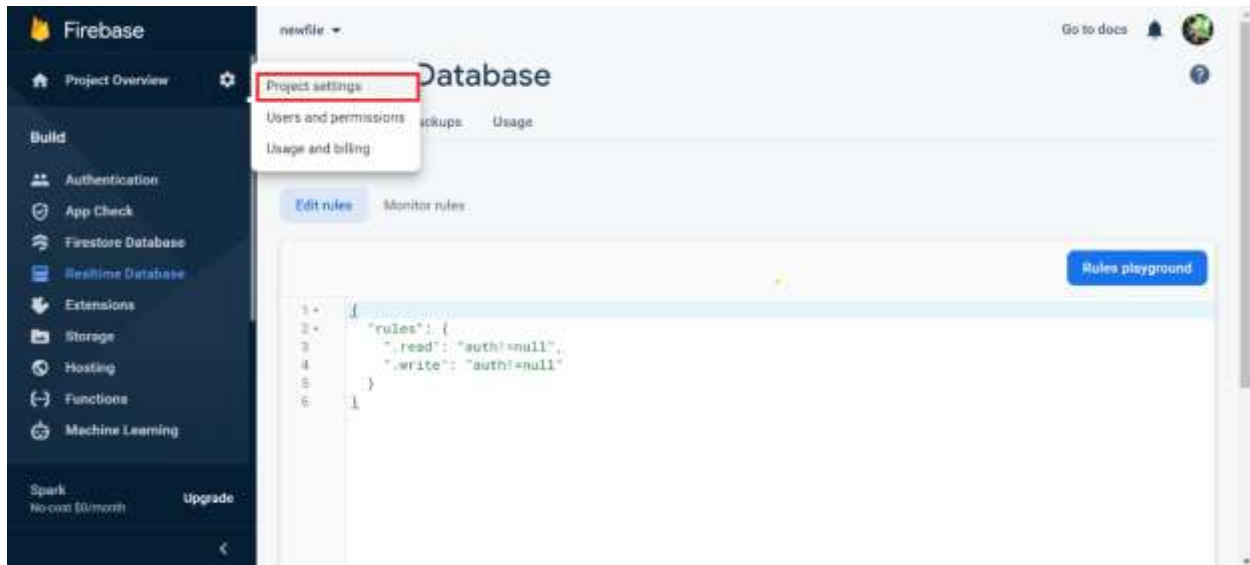


- To interact with the Firebase Realtime Database through your application, it is necessary to grant read and write permissions by defining appropriate rules in the Firebase project's **Rules tab**, as shown in the following code snippet. Once you have specified the rules, you can publish them by clicking the **Publish** button to enable the necessary authentication.

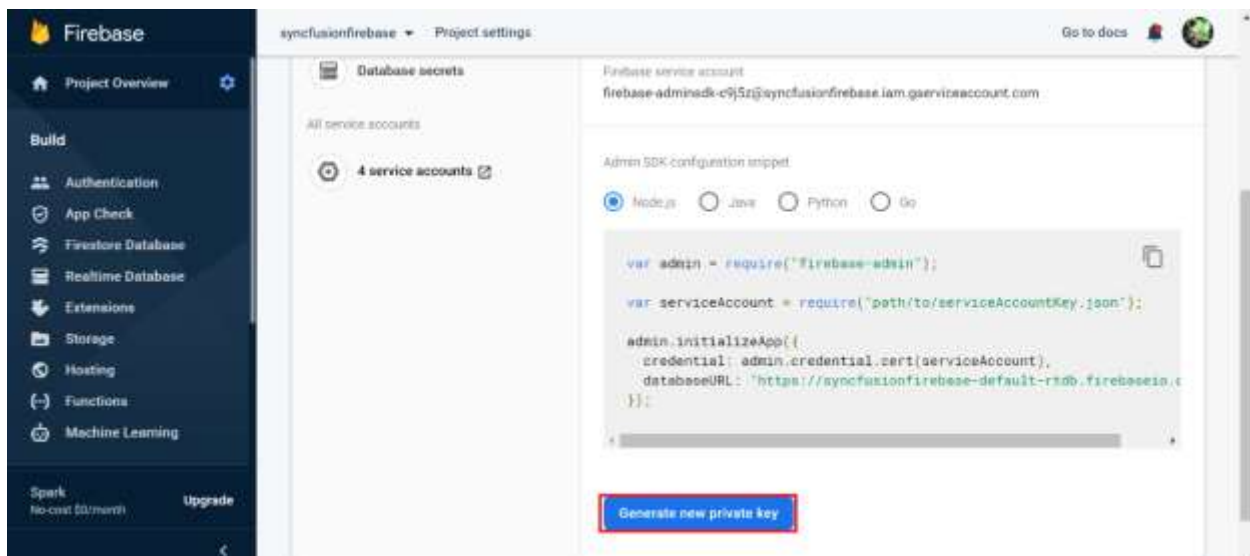
```
`ts
{
  / Visit https://firebase.google.com/docs/database/security to learn more about security rules. /
  "rules": {
    ".read": "auth!=null",
    ".write": "auth!=null"
  }
}
```

**Note:** By default, rules of a Firebase project will be **false**. To read and write the data, configure the **Rules** as given in the following code snippet in the *Rules* tab in the Firebase Realtime Database project.

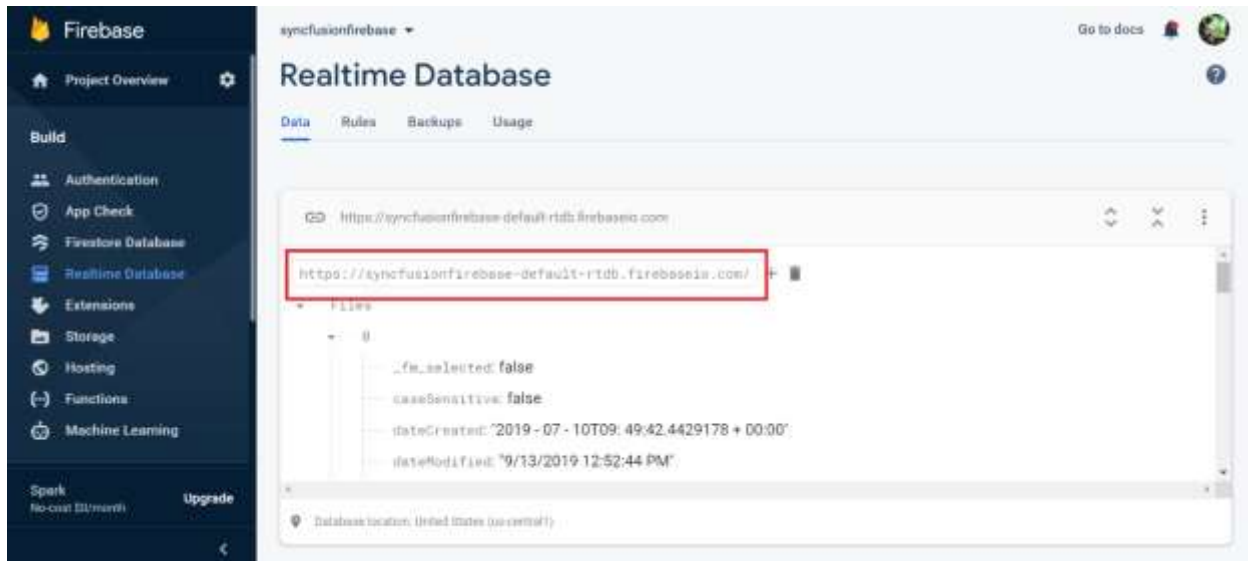
- Navigate to the project settings as instructed and then click on the **Service Account** tab.



- To obtain the access key JSON file, simply click on the **Generate new private key** button and then confirm by clicking the **Generate key** button in the pop-up window that appears.



- Next, you will need to clone the [firebase-realtime-database-apscore-file-provider](#) repository. Once cloned, simply open the project in Visual Studio and restore the NuGet package.
- Once you have generated the secret key, you will need to replace the JSON in the `access_key.json` file in the Firebase Realtime Database provider project with the newly generated key. This will enable authentication and allow you to perform read and write operations.
- In the **Data** tab, locate the project API URL and then paste it into the below mentioned section.



Register the Firebase Realtime Database by assigning *Firebase Realtime Database REST API link*, *rootNode*, and *serviceAccountKeyPath* parameters in the `RegisterFirebaseRealtimeDB` method of class `FirebaseRealtimeDBFileProvider` in the controller part of the ASP.NET Core application.

```
`ts
```

```
this.operation.RegisterFirebaseRealtimeDB(string apiUrl, string rootNode, string serviceAccountKeyPath)
`
```

#### Example:

```
`ts
```

```
this.operation.RegisterFirebaseRealtimeDB("{copy your API URL here}", "Files",
hostingEnvironment.ContentRootPath + "\\FirebaseRealtimeDBHelper\\access_key.json");
`
```

In the above code,

- `{copy your API URL here}` denotes Firebase Realtime Database REST API link.
- `Files` denotes newly created root node in Firebase Realtime Database.
- `hostingEnvironment.ContentRootPath + "\\FirebaseRealtimeDBHelper\\access_key.json` denotes service account key path which has authentication key for the Firebase Realtime Database data.

After configuring the Firebase Realtime Database service link, build and run the project. Now, the project will be hosted in `http://localhost:{port}` and just mapping the `ajaxSettings` property of the File Manager component to the appropriate controller methods allows to manage the files in the Firebase Realtime Database.

```
<template>
```

```
<div id="app">
```

```
<ejs-filemanager id="file-manager" :ajaxSettings="ajaxSettings">
</ejs-filemanager>
</div>
</template>
<script>
import Vue from "vue";

import { FileManagerPlugin, DetailsView, NavigationPane, Toolbar } from "@syncfusion/ej2-vue-filemanager";

Vue.use(FileManagerPlugin);

export default {
  data () {
    return {
      // Initializing File Manager NodeJS file system service.
      ajaxSettings:
      {
        // Replace the hosted port number in the place of "{port}"
        url: "http://localhost:{port}/api/FirebaseProvider/FirebaseRealtimeFileOperations",
        getImageUrl: "http://localhost:{port}/api/FirebaseProvider/FirebaseRealtimeGetImage",
        uploadUrl: "http://localhost:{port}/api/FirebaseProvider/FirebaseRealtimeUpload",
        downloadUrl: "http://localhost:{port}/api/FirebaseProvider/FirebaseRealtimeDownload"
      }
    };
  },
  provide: {
    filemanager: [DetailsView, NavigationPane, Toolbar]
  }
}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-icons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-inputs/styles/material.css";
@import "../node_modules/@syncfusion/ej2-popups/styles/material.css";
```

```
@import "../node_modules/@syncfusion/ej2-buttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-splitbuttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-navigations/styles/material.css";
@import "../node_modules/@syncfusion/ej2-layouts/styles/material.css";
@import "../node_modules/@syncfusion/ej2-grids/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-filemanager/styles/material.css";
</style>
`ts
```

> **Note:** To learn more about the file actions that can be performed with Firebase Realtime Database file system provider, refer to this [link](#)

### IBM Cloud Object Storage file provider

The IBM Cloud Object Storage file provider module allows you work with the IBM Cloud Object Storage. It also provides the methods for performing various file actions such as creating a new folder, renaming files, and deleting files. The IBM Cloud Object Storage file provider serves the file provider support for the File Manager component with the IBM Cloud Object Storage. We can make use of IBM Cloud Object Storage file provider by installing the [ej2-filemanager-ibm-cos-node-file-provider](#) npm package or by cloning the [file provider](#) from the GitHub.

#### Using ej2-filemanager-ibm-cos-node-file-provider npm package

- Install the ej2-filemanager-ibm-cos-node-file-provider npm package by running the below command.

```
`ts
```

```
npm install @syncfusion/ej2-filemanager-ibm-cos-node-file-provider
```

```
,
```

- After installing the package, navigate to the ej2-filemanager-ibm-cos-node-file-provider package folder within the node-modules.
- Run the **npm install** command to install the dependent packages for file provider.

#### Cloning the filemanager-ibm-cos-node-file-provider from GitHub

- Clone the filemanager-ibm-cos-node-file-provider using the following command.

```
`ts
```

```
git clone https://github.com/SyncfusionExamples/filemanager-ibm-cos-node-file-provider.git
```

```
,
```

- After cloning, open the root folder and run the command **npm install** command.

To set the port in which the project to be hosted. Run the following command.

```
`ts
set PORT=3000 && node index.js
`
```

**Note:** By default, the service will run 8090 port.

Now, just mapping the **ajaxSettings** property of the FileManager component to the appropriate file operation methods in the index.js file will allow to manage the IBM Cloud Object Storage.

```
<template>
<div id="app">
  <ejs-filemanager id="file-manager" :ajaxSettings="ajaxSettings">
</ejs-filemanager>
</div>
</template>
<script>
import Vue from "vue";

import { FileManagerPlugin, DetailsView, NavigationPane, Toolbar } from "@syncfusion/ej2-vue-filemanager";

Vue.use(FileManagerPlugin);

export default {
  data () {
    return {
      // Initializing File Manager NodeJS file system service.
      ajaxSettings:
      {
        // Replace the hosted port number in the place of "{port}"
        url: "http://localhost:{port}/",
        getImageUrl: "http://localhost:{port}/GetImage",
        uploadUrl: "http://localhost:{port}/Upload",
        downloadUrl: "http://localhost:{port}/Download"
      }
    };
  },
  provide: {
    filemanager: [DetailsView, NavigationPane, Toolbar]
  }
}
```



```

}
}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-icons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-inputs/styles/material.css";
@import "../node_modules/@syncfusion/ej2-popups/styles/material.css";
@import "../node_modules/@syncfusion/ej2-buttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-splitbuttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-navigations/styles/material.css";
@import "../node_modules/@syncfusion/ej2-layouts/styles/material.css";
@import "../node_modules/@syncfusion/ej2-grids/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-filemanager/styles/material.css";
</style>
`

```

**Note:** To learn more about the file actions that can be performed with IBM Cloud Object Storage file provider, refer to this [link](#)

### Localization in Vue File manager component

The file manager can be localized to any culture by defining the texts and messages of the file manager in the corresponding culture. The default locale of the file manager is **en** (English). The following table represents the default texts and messages of the file manager in **en** culture.

KEY	Text/Message
----	----
NewFolder	New folder
Upload	Upload
Delete	Delete
Rename	Rename
Download	Download
Cut	Cut
Copy	Copy
Paste	Paste
SortBy	Sort by
Refresh	Refresh

Item-Selection	item selected
Items-Selection	items selected
View	View
Details	Details
SelectAll	Select all
Open	Open
Tooltip-NewFolder	New folder
Tooltip-Upload	Upload
Tooltip-Delete	Delete
Tooltip-Rename	Rename
Tooltip-Download	Download
Tooltip-Cut	Cut
Tooltip-Copy	Copy
Tooltip-Paste	Paste
Tooltip-SortBy	Sort by
Tooltip-Refresh	Refresh
Tooltip-Selection	Clear selection
Tooltip-View	View
Tooltip-Details	Details
Tooltip-SelectAll	Select all
Name	Name
Size	Size
DateModified	Modified
DateCreated	Date created
Path	Path
Created	Created
Modified	Modified
Location	Location
Type	Type
Permission	Permission
Ascending	Ascending
Descending	Descending
None	None

View-LargeIcons	Large icons
View-Details	Details
Search	Search
Button-Ok	OK
Button-Cancel	Cancel
Button-Yes	Yes
Button-No	No
Button-Create	Create
Button-Save	Save
Header-NewFolder	Folder
Content-NewFolder	Enter your folder name
Header-Rename	Rename
Content-Rename	Enter your new name
Header-Rename-Confirmation	Rename Confirmation
Content-Rename-Confirmation	If you change a file name extension
Are you sure you want to change it?	
Header-Delete	Delete File
Content-Delete	Are you sure you want to delete this file?
Header-Multiple-Delete	Delete Multiple Files
Content-Multiple-Delete	Are you sure you want to delete these {0} files?
Header-Folder-Delete	Delete Folder
Content-Folder-Delete	Are you sure you want to delete this folder?
Header-Duplicate	File exists
Content-Duplicate	already exists. Are you sure you want to replace it?
Header-Upload	Upload Files
Error	Error
Validation-Empty	The file or folder name cannot be empty.
Validation-Invalid	The file or folder name {0} contains invalid characters. Please use a different name.
Valid file or folder names cannot end with a dot or space, and cannot contain any of the following	
characters: \\ \: \* ? \ " < > \	
Validation-NewFolder-Exists	A file or folder with the name {0} already exists.
Validation-Rename-Exists	Cannot rename {0} to {1}
Folder-Empty	This folder is empty
File-Upload	Drag files here to upload

|Search-Empty|No results found|

|Search-Key|Try with different keywords|

|Filter-Empty|No results found|

|Filter-Key|Try with different filter|

|Sub-Folder-Error|The destination folder is the subfolder of the source folder|

|Same-Folder-Error|The destination folder is the same as the source folder.|

|Access-Denied|Access Denied|

|Access-Details|You don't have permission to access this folder|

|Header-Retry|File Already Exists|

|Content-Retry|A file with this name already exists in this folder. What would you like to do?|

|Button-Keep-Both|Keep both|

|Button-Replace|Replace|

|Button-Skip|Skip|

|ApplyAll-Label|Do this for all current items|

|KB|KB|

|Access-Message|{0} is not accessible. You need permission to perform the {1} action.|

|Network-Error|Network Error: Failed to send on XMLHttpRequest: Failed to load|

|Server-Error|Server Error: Invalid response from|

The below example shows adding the German culture locale(**de-DE**)

#### **APP.VUE**

```
<template>
<div id="app">
  <div class="wrapper">
    <ejs-filemanager id="locale_filemanager" :ajaxSettings="ajaxSettings"
:locale="locale">
      </ejs-filemanager>
    </div>
  </div>
</template>
<script>
import Vue from "vue";
import { FileManagerPlugin, DetailsView, NavigationPane, Toolbar } from
"@syncfusion/ej2-vue-filemanager";
import { loadCldr,L10n } from "@syncfusion/ej2-base";
Vue.use(FileManagerPlugin);
//Defining texts and messages corresponding to German culture
L10n.load({
  "de": {
    "filemanager": {
      "NewFolder": "Neuer Ordner",
      "Upload": "Hochladen",
      "Delete": "Löschen",
      "Rename": "Umbenennen",
```

```

"Download": "Herunterladen",
"Cut": "Schnitt",
"Copy": "Kopieren",
"Paste": "Einfügen",
"SortBy": "Sortiere nach",
"Refresh": "Aktualisierung",
"Item-Selection": "Artikel ausgewählt",
"Items-Selection": "Elemente ausgewählt",
"View": "Aussicht",
"Details": "Einzelheiten",
"SelectAll": "Wählen Sie Alle",
"Open": "Öffnen",
"Tooltip-NewFolder": "Neuer Ordner",
"Tooltip-Upload": "Hochladen",
"Tooltip-Delete": "Löschen",
"Tooltip-Rename": "Umbenennen",
"Tooltip-Download": "Herunterladen",
"Tooltip-Cut": "Schnitt",
"Tooltip-Copy": "Kopieren",
"Tooltip-Paste": "Einfügen",
"Tooltip-SortBy": "Sortiere nach",
"Tooltip-Refresh": "Aktualisierung",
"Tooltip-Selection": "Auswahl aufheben",
"Tooltip-View": "Aussicht",
"Tooltip-Details": "Einzelheiten",
"Tooltip-SelectAll": "Wählen Sie Alle",
"Name": "Name",
"Size": "Größe",
"DateModified": "Geändert",
"DateCreated": "Datum erstellt",
"Path": "Pfad",
"Modified": "Geändert",
"Created": "Erstellt",
"Location": "Ort",
"Type": "Art",
"Permission": "Genehmigung",
"Ascending": "Aufsteigend",
"Descending": "Absteigend",
"None": "Keiner",
"View-LargeIcons": "Große Icons",
"View-Details": "Einzelheiten",
"Search": "Suche",
"Button-Ok": "OK",
"Button-Cancel": "Stornieren",
"Button-Yes": "Ja",
"Button-No": "Nein",
"Button-Create": "Erstellen",
"Button-Save": "Sparen",
"Header-NewFolder": "Mappe",
"Content-NewFolder": "Geben Sie Ihren Ordernamen ein",
"Header-Rename": "Umbenennen",
"Content-Rename": "Geben Sie Ihren neuen Namen ein",
"Header-Rename-Confirmation": "Bestätigung umbenennen",
"Content-Rename-Confirmation": "Wenn Sie eine
Dateinamenerweiterung ändern, wird die Datei möglicherweise instabil.
Möchten Sie sie wirklich ändern?",
"Header-Delete": "Datei löschen",

```

```

        "Content-Delete": "Möchten Sie diese Datei wirklich löschen?",
        "Header-Multiple-Delete": "Mehrere Dateien löschen",
        "Content-Multiple-Delete": "Möchten Sie diese {0} Dateien
wirklich löschen?",
        "Header-Folder-Delete": "Lösche Ordner",
        "Content-Folder-Delete": "Möchten Sie diesen Ordner wirklich
löschen?",
        "Header-Duplicate": "Datei / Ordner existiert",
        "Content-Duplicate": "{0} existiert bereits. Möchten Sie
umbenennen und einfügen?",
        "Header-Upload": "Daten hochladen",
        "Error": "Error",
        "Validation-Empty": "Der Datei - oder Ordnername darf nicht leer
sein.",
        "Validation-Invalid": "Der Datei- oder Ordnername {0} enthält
ungültige Zeichen. Bitte verwenden Sie einen anderen Namen. Gültige Datei-
oder Ordnernamen dürfen nicht mit einem Punkt oder Leerzeichen enden und
keines der folgenden Zeichen enthalten: \\ /: *? \" < > | ",
        "Validation-NewFolder-Exists": "Eine Datei oder ein Ordner mit
dem Namen {0} existiert bereits.",
        "Validation-Rename-Exists": "{0} kann nicht in {1} umbenannt
werden: Ziel existiert bereits.",
        "Folder-Empty": "Dieser Ordner ist leer",
        "File-Upload": "Dateien zum Hochladen hierher ziehen",
        "Search-Empty": "Keine Ergebnisse gefunden",
        "Search-Key": "Versuchen Sie es mit anderen Stichwörtern",
        "Filter-Empty": "keine Ergebnisse gefunden",
        "Filter-Key": "Versuchen Sie es mit einem anderen Filter",
        "Sub-Folder-Error": "Der Zielordner ist der Unterordner des
Quellordners.",
        "Same-Folder-Error": "Der Zielordner ist derselbe wie der
Quellordner.",
        "Access-Denied": "Zugriff verweigert",
        "Access-Details": "Sie haben keine Berechtigung, auf diesen
Ordner zuzugreifen.",
        "Header-Retry": "Die Datei existiert bereits",
        "Content-Retry": "In diesem Ordner ist bereits eine Datei mit
diesem Namen vorhanden. Was möchten Sie tun?",
        "Button-Keep-Both": "Behalte beides",
        "Button-Replace": "Ersetzen",
        "Button-Skip": "Überspringen",
        "ApplyAll-Label": "Mache das für alle aktuellen Artikel",
        "KB": "KB",
        "Access-Message": "{0} ist nicht zugänglich. Sie benötigen die
Berechtigung, um die Aktion {1} auszuführen.",
        "Network-Error": "NetworkError: Fehler beim Senden auf
XMLHttpRequest: Fehler beim Laden",
        "Server-Error": "ServerError: Ungültige Antwort von"
    }
  }
});
export default {
  data () {
    return {
      ajaxSettings:
    {

```

```

        url: "https://ej2-aspcore-
service.azurewebsites.net/api/FileManager/FileOperations",
        getImageUrl: "https://ej2-aspcore-
service.azurewebsites.net/api/FileManager/GetImage",
        uploadUrl: "https://ej2-aspcore-
service.azurewebsites.net/api/FileManager/Upload",
        downloadUrl: "https://ej2-aspcore-
service.azurewebsites.net/api/FileManager/Download"
    },
    //defining the locale for File Manager
    locale: "de"
  };
},
provide: {
  filemanager: [DetailsView, NavigationPane, Toolbar]
}
}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-icons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-inputs/styles/material.css";
@import "../node_modules/@syncfusion/ej2-popups/styles/material.css";
@import "../node_modules/@syncfusion/ej2-buttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-splitbuttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-navigations/styles/material.css";
@import "../node_modules/@syncfusion/ej2-layouts/styles/material.css";
@import "../node_modules/@syncfusion/ej2-grids/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-
filemanager/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/file-manager/locale-cs1" %}

### Accessibility in Vue File Manager component

The File Manager component followed the accessibility guidelines and standards, including [ADA](#), [Section 508](#), [WCAG 2.2](#) standards, and [WCAG roles](#) that are commonly used to evaluate accessibility.

The accessibility compliance for the File Manager component is outlined below.

| Accessibility Criteria | Compatibility |

| -- | -- |

| [WCAG 2.2](#) Support |  |

| [Section 508](#) Support |  |

| Screen Reader Support |  |

| Right-To-Left Support |  |

| Color Contrast |  |

| Mobile Device Support |  |

| Keyboard Navigation Support |  |

| [Accessibility Checker](#) Validation |  |

| [Axe-core](#) Accessibility Validation |  |

```
<style>
.post .post-content img {
display: inline-block;
margin: 0.5em 0;
}
</style>

<div> - All
features of the component meet the requirement.</div>

<div> - Some features of the component do not meet the requirement.</div>

<div> - The
component does not meet the requirement.</div>
```

### WAI-ARIA attributes

The File Manager component followed the [WAI-ARIA](#) patterns to meet the accessibility. The following ARIA attributes are used in the File Manager component:

Attributes	Purpose
role	Used to convey a significant and contextual message to the user.
aria-disabled	Indicates whether the File Manager component is in disabled state.
aria-haspopup	Indicates whether the toolbar item has a popup list or not.
aria-orientation	Indicates whether the File Manager element is oriented horizontally or vertically.
aria-expanded	Indicates whether the Treeview node has been expanded.
aria-owns	Contains the ID of the suggestion list to indicate popup as a child element.
aria-activedescendent	Holds the ID of the active list item to focus its descendant child element.
aria-level	Specifies the level of the element in Treeview Structure.
aria-selected	Indicates whether a particular node is in selected state.



- | **aria-placeholder** | Represents a hint (word or phrase) to the user about what to enter in the text field. |
- | **aria-label** | Provides an accessible name for the element. |
- | **aria-checked** | Indicates whether the checkbox is in checked state. |
- | **aria-labelledby** | Provides a label for the dialog. Typically, the "aria-labelledby" attribute will contain the id of the element used as the dialog's title. |
- | **aria-describedby** | This attribute points to the Dialog element describing the one it's set on. |
- | **aria-modal** | Indicates whether an element is a modal when display. |
- | **aria-colcount** | Specifies the number of columns in full table. |
- | **aria-colindexnt** | Defines the number of columns within a table in details view. |
- | **aria-rowspan** | Defines the number of rows a cell spanned within a table in details view. |
- | **aria-colspan** | Defines the number of columns a cell spanned within a table in details view. |
- | **aria-sort** | Indicates whether items in the table are sorted in ascending or descending order. |
- | **aria-grabbed** | When the folder/file item is chosen for dragging, the aria-grabbed attribute is set to "true." If it's set to "false," the element can be grabbed for drag-and-drop, but it won't be actively held. |
- | **aria-busy** | This attribute is set to false when table content is loaded. |
- | **aria-multiselectable** | Defines more than one item has been selected. |

### Keyboard interaction

The File Manager component followed the **keyboard interaction** guidelines, making it easy for people who use assistive technologies (AT) and those who completely rely on keyboard navigation. The following keyboard shortcuts are supported by the File Manager component.

- | **Press** | **To do this** |
- | --- | --- |
- | **Page Down** | Scrolls down to the next folder or file and selects the first item when files are loaded. |
- | **Page Up** | Scrolls up to previous folder and select the first item when files are loaded. |
- | **Enter** | Selects the focused item and navigate through the child elements. |
- | **Tab** | Focuses on the first element of toolbar and navigates through the next tab indexed element. |
- | **Esc(Escape)** | Closes the image when it is in open state. |
- | **Alt+N** | Creates a new folder dialog. |
- | **F5** | Refresh the file manager element. |
- | **Home** | Navigate through the first element of details view or large icons view. |

- | End | Navigate through the last element of details view or large icons view. |
- | Move Left | Scrolls left to the previous folder and select the first item when files are loaded |
- | Move Right | Scrolls right to the previous folder and select the first item when files are loaded |
- | Alt+Enter | Shows the get details info for selected folder. |
- | Shift+Right | Allows multiselection. Select the file or folder at the right of the previously selected folder. |
- | Shift+Left | Allows multiselection. Select the file or folder at the left of the previously selected folder. |
- | Shift+Down | Allows multiselection. Select the file or folder till the focused index. |
- | Shift+Delete | Permanently deletes the selected file or folder in the file manager element. |
- | Delete | Deletes the selected file or folder in the file manager element. |
- | Shift+Up | Allows multiselection. Select the file or folder till the focused index. |
- | Ctrl+C | Copies the selected file or folder in the file manager element. |
- | Ctrl+V | Pastes the copied/cut file or folder in the file manager element. |
- | Ctrl+X | Cuts the selected file or folder in the file manager element. |
- | Ctrl+A | Select all the files or folders in the details view or large icons view. |
- | F2 | Creates a rename dialog for a selected file or folder in the file manager element. |
- | Shift+F10 | Opens the context menu for the selected file or folder in the file manager element. |
- | Ctrl+D | Downloads the list of selected files or folders in the file manager element. |
- | Ctrl+Shift+1 | Changes the file manager layout to details view. |
- | Ctrl+Shift+2 | Changes the file manager layout to details view. |

### Ensuring accessibility

The File Manager component's accessibility levels are ensured through an [accessibility-checker](#) and [axe-core](#) software tools during automated testing.

The accessibility compliance of the File Manager component is shown in the following sample. Open the [sample](#) in a new window to evaluate the accessibility of the File Manager component with accessibility tools.

```
{% previewsample "https://ej2.syncfusion.com/accessibility/file-manager.html" %}
```

See also

- [Accessibility in Syncfusion Vue components](#)

## Access control in Vue File manager component

The [Vue FileManager](#) allows you to define access permissions for folders and files using a set of access rules to user(s).

- [Access Rules](#)
- [Permissions](#)

### Access Rules

The FileAccessController allows you to define security permissions for folders and files using a set of folder or file access rules.

To set up access rules for folders (including their files and sub-folders) and individual files, use the SetRules() method. The following table represents the AccessRule properties available for file and folder:

Properties	Applicable for file	Applicable for folder	Description
---	---	---	---
Copy	Yes	Yes	Allows access to copy a file or folder.
Read	Yes	Yes	Allows access to read a file or folder.
Write	Yes	Yes	Allows permission to write a file or folder.
WriteContents	No	Yes	Allows permission to write the content of folder.
Download	Yes	Yes	Allows permission to download a file or folder.
Upload	No	Yes	Allows permission to upload to the folder.
Path	Yes	Yes	Specifies the path to apply the rules, which are defined.
Role	Yes	Yes	Specifies the role to which the rule is applied.
IsFile	Yes	Yes	Specifies whether the rule is specified for folder or file.

The following syntax represents the access Rules for Administrator using file or folder.

```
`ts
//Administrator
//Access Rules for File
new AccessRule { Path = "/", Role = "Administrator", Read = Permission.Allow, Write =
Permission.Allow, Copy = Permission.Allow, Download = Permission.Allow, IsFile = true },
// Access Rules for folder
new AccessRule { Path = "**", Role = "Administrator", Read = Permission.Allow, Write = Permission.Allow,
Copy = Permission.Allow, WriteContents = Permission.Allow, Upload = Permission.Allow, Download =
Permission.Deny, IsFile = false },
`
```

The following syntax represent the access Rules for Default user using file or folder.

```
`ts
//Default User
```

```
//Access Rules for File
```

```
new AccessRule { Path = "/", Role = "Default User", Read = Permission.Deny, Write = Permission.Deny,
Copy = Permission.Deny, Download = Permission.Deny, IsFile = true },
```

```
// Access Rules for folder
```

```
new AccessRule { Path = "*", Role = "Default User", Read = Permission.Deny, Write = Permission.Deny,
Copy = Permission.Deny, WriteContents = Permission.Deny, Upload = Permission.Deny, Download =
Permission.Deny, IsFile = false },
```

```
,
```

### Permissions

It helps to explain how to apply security permission to file manager file or folder using access rules. The following table represent the value that determines the permission.

Value	Description
---	---
Allow	Allows you to do read, write, copy, and download operations.
Deny	Denies you to do read, write, copy, and download operations.

Use the **Role** property to apply created roles to the file manager. After that, the file manager displays folder or file and allow permission based on assigned roles.

The following syntax represent how to apply permission based on assigned roles

Permission denied for administrator to write a file or folder.

```
`ts
```

```
// For file
```

```
new AccessRule { Path = "/", Role = "Administrator", Read = Permission.Allow, Write = Permission.Deny,
IsFile = true},
```

```
// For folder
```

```
new AccessRule { Path = "*", Role = "Administrator", Read = Permission.Allow, Write = Permission.Deny,
IsFile = false},
```

```
,
```

The following syntax represent how to allow or deny permission based on file or folder access rule.

"Examples"

Permission denied for writing except for particular file or folder.

```
`ts
```

```
// Deny writing for particular folder
```

```
new AccessRule { Path = "/Documents", Role = "Document Manager", Read = Permission.Allow, Write =
Permission.Deny, Copy = Permission.Allow, WriteContents = Permission.Deny, Upload =
Permission.Deny, Download = Permission.Deny, IsFile = false },
```

```
// Deny writing for particular file
```

```
new AccessRule { Path = "/Pictures/Employees/Adam.png", Role = "Document Manager", Read =
Permission.Allow, Write = Permission.Deny, Copy = Permission.Deny, Download = Permission.Deny,
IsFile = true },
,
```

Permission denied for writing and uploading in root folder.

```
`ts
```

```
// Folder Rule
```

```
new AccessRule { Path = "/", Role = "Document Manager", Read = Permission.Allow, Write =
Permission.Deny, Copy = Permission.Deny, WriteContents = Permission.Deny, Upload =
Permission.Deny, Download = Permission.Deny, IsFile = false },
,
```

The following example demonstrate the file manager rendered with access control support.

### **APP.VUE**

```
<template>
  <div id="app">
    <ejs-filemanager id="file-manager" :ajaxSettings="ajaxSettings">
      </ejs-filemanager>
    </div>
  </template>
<script>
import Vue from "vue";
import { FileManagerPlugin, DetailsView, NavigationPane, Toolbar } from
"@syncfusion/ej2-vue-filemanager";
Vue.use(FileManagerPlugin);
export default {
  data () {
    return {
      ajaxSettings:
      {
        url: "https://ej2-aspcore-
service.azurewebsites.net/api/FileManagerAccess/FileOperations",
        getImageUrl: "https://ej2-aspcore-
service.azurewebsites.net/api/FileManagerAccess/GetImage",
        uploadUrl: "https://ej2-aspcore-
service.azurewebsites.net/api/FileManagerAccess/Upload",
        downloadUrl: "https://ej2-aspcore-
service.azurewebsites.net/api/FileManagerAccess/Download"
      },
    };
  },
  provide: {
    filemanager: [DetailsView, NavigationPane, Toolbar]
  }
}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-icons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-inputs/styles/material.css";
```

```
@import "../node_modules/@syncfusion/ej2-popups/styles/material.css";
@import "../node_modules/@syncfusion/ej2-buttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-splitbuttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-navigations/styles/material.css";
@import "../node_modules/@syncfusion/ej2-layouts/styles/material.css";
@import "../node_modules/@syncfusion/ej2-grids/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-
filemanager/styles/material.css";
</style>
```

```
{% previewsample "page.domainurl/code-snippet/file-manager/access-control-cs1" %}
```

## How To

Adding custom item to context menu in Vue File manager component

The context menu can be customized using the [contextMenuSettings](#), [menuOpen](#), and [menuClick](#) events.

The following example shows adding a custom item in the context menu.

The [contextMenuSettings](#) is used to add new menu item. The [menuOpen](#) event is used to add the icon to the new menu item. The [menuClick](#) event is used to add an event handler to the new menu item.

### APP.VUE

```
<template>
  <div id="app">
    <ejs-filemanager id="file-manager" ref="file_instance"
:contextMenuSettings="contextMenuSettings" :ajaxSettings="ajaxSettings"
:menuOpen="menuOpen" :menuClick="menuClick">
      </ejs-filemanager>
    </div>
  </template>
<script>
import Vue from "vue";
import { FileManagerPlugin, DetailsView, NavigationPane, Toolbar } from
"@syncfusion/ej2-vue-filemanager";
Vue.use(FileManagerPlugin);
export default {
  data () {
    return {
      ajaxSettings:
      {
        url: "https://ej2-aspcore-
service.azurewebsites.net/api/FileManager/FileOperations",
        getImageUrl: "https://ej2-aspcore-
service.azurewebsites.net/api/FileManager/GetImage",
        uploadUrl: "https://ej2-aspcore-
service.azurewebsites.net/api/FileManager/Upload",
        downloadUrl: "https://ej2-aspcore-
service.azurewebsites.net/api/FileManager/Download"
      },
      // Custom menu item added to context menu
      contextMenuSettings: {
        file: ["Custom", "Open", "|", "Delete", "Rename", "|",
"Details"],
```

```

        folder: ["Custom", "Open", "|", "Delete", "Rename", "|",
"Details", "Custom"],
        layout: ["Custom", "SortBy", "View", "Refresh", "|",
"NewFolder", "Upload", "|", "Details", "|", "SelectAll"],
        visible: true
    }
};
},
provide: {
    filemanager: [DetailsView, NavigationPane, Toolbar]
},
methods: {
    // Icon added to custom menu item in menuOpen event
    menuOpen: function(args){
        for(let i: number = 0; i<args.items.length; i++) {
            if(args.items[i].id === this.$refs.file_instance.$el.id
+"_cm_custom") {
                args.items[i].iconCss= "e-icons e-fe-tick";
            }
        }
    },
    // Displaying alert for custom menu in menuClick event
    menuClick: function(args) {
        if (args.item.text === "Custom") {
            alert("You have clicked custom menu item")
        }
    }
}
}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-icons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-inputs/styles/material.css";
@import "../node_modules/@syncfusion/ej2-popups/styles/material.css";
@import "../node_modules/@syncfusion/ej2-buttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-splitbuttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-navigations/styles/material.css";
@import "../node_modules/@syncfusion/ej2-layouts/styles/material.css";
@import "../node_modules/@syncfusion/ej2-grids/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-
filemanager/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/file-manager/contextmenu-cs2" %}

### Adding custom item to toolbar in Vue File manager component

You can modify the items displayed in the toolbar by utilizing the [toolbarItems](#) API. To display both default and customized items, it's essential to assign a unique **name** to each item. Additionally, you have the flexibility to alter the default items by adjusting properties such as **tooltipText**, **iconCss**, **Text**, **suffixIcon** and more. This level of customization allows you to tailor the toolbar to your specific requirements and design preferences. The names used in the code example below serve as unique identifiers for default toolbar items, while custom items can be assigned any unique name value to distinguish them from the defaults.

For instance, here's an example of how to add a custom checkbox to the toolbar using the **template** property. Here we have modified the default **New Folder** item and added a custom toolbar item for selection.

### APP.VUE

```
<template>
  <div id="app">
    <ejs-filemanager id="file-manager" ref="fileManagerInstance"
:ajaxSettings="ajaxSettings" >
      <e-toolbaritems>
        <e-toolbaritem name="NewFolder"></e-toolbaritem>
        <e-toolbaritem name="Upload"></e-toolbaritem>
        <e-toolbaritem name="SortBy"></e-toolbaritem>
        <e-toolbaritem name="Refresh"></e-toolbaritem>
        <e-toolbaritem name="Cut"></e-toolbaritem>
        <e-toolbaritem name="Copy"></e-toolbaritem>
        <e-toolbaritem name="Paste"></e-toolbaritem>
        <e-toolbaritem name="Delete"></e-toolbaritem>
        <e-toolbaritem name="Download"></e-toolbaritem>
        <e-toolbaritem name="Rename"></e-toolbaritem>
        <e-toolbaritem name="Select"
:template="'checkboxTemplate'">
          <template v-slot:checkboxTemplate>
            <div><ejs-checkbox ref="checkBoxInstance"
:label='Select All' :checked=false :change="onChange"></ejs-checkbox></div>
          </template>
        </e-toolbaritem>
        <e-toolbaritem name="Selection"></e-toolbaritem>
        <e-toolbaritem name="View"></e-toolbaritem>
        <e-toolbaritem name="Details"></e-toolbaritem>
      </e-toolbaritems>
    </ejs-filemanager>
  </div>
</template>
<script>
import Vue from "vue";
import { FileManagerPlugin, DetailsView, NavigationPane, Toolbar } from
"@syncfusion/ej2-vue-filemanager";
import { CheckBoxPlugin } from "@syncfusion/ej2-vue-buttons";
Vue.use(FileManagerPlugin);
Vue.use(CheckBoxPlugin);
export default {
  data () {
    return {
      ajaxSettings:
        {
          url: "https://ej2-aspcore-
service.azurewebsites.net/api/FileManager/FileOperations",
          getImageUrl: "https://ej2-aspcore-
service.azurewebsites.net/api/FileManager/GetImage",
          uploadUrl: "https://ej2-aspcore-
service.azurewebsites.net/api/FileManager/Upload",
          downloadUrl: "https://ej2-aspcore-
service.azurewebsites.net/api/FileManager/Download"
        }
    }
  };
};
```



```

    },
    provide: {
        filemanager: [DetailsView, NavigationPane, Toolbar]
    },
    methods: {
        onChange: function(args){
            if (args.checked) {
                this.$refs.fileManagerInstance.selectAll();
                this.$refs.checkBoxInstance.label = 'Unselect All';
            }
            else {
                this.$refs.fileManagerInstance.clearSelection();
                this.$refs.checkBoxInstance.label = 'Select All';
            }
        }
    }
}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-icons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-inputs/styles/material.css";
@import "../node_modules/@syncfusion/ej2-popups/styles/material.css";
@import "../node_modules/@syncfusion/ej2-buttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-splitbuttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-navigations/styles/material.css";
@import "../node_modules/@syncfusion/ej2-layouts/styles/material.css";
@import "../node_modules/@syncfusion/ej2-grids/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-filemanager/styles/material.css";
.e-fe-tick::before {
    content: '\e614';
}
</style>

```

{% previewsample "page.domainurl/code-snippet/file-manager/toolbar-cs2" %}

Enable/disable toolbar item in Vue File manager component

The toolbar items can be enabled/disabled by specifying the items in [enableToolbarItems](#) or [disableToolbarItems](#) methods respectively.

The following example shows enabling and disabling toolbar items on button click.

#### APP.VUE

```

<template>
  <div id="app">
    <ejs-button id="enable" cssClass="e-success">Enable New Folder
  toolbar item</ejs-button>
    <ejs-button id="disable" cssClass="e-danger">Disable New Folder
  toolbar item</ejs-button>
    <ejs-filemanager :created="onCreated" id="file-manager"
  ref="fileManagerInstance" :height="height" :ajaxSettings="ajaxSettings">
    </ejs-filemanager>
  </div>
</template>

```

```

<script>
import Vue from "vue";
import { FileManagerPlugin, DetailsView, NavigationPane, Toolbar } from
"@syncfusion/ej2-vue-filemanager";
import { ButtonPlugin } from "@syncfusion/ej2-vue-buttons";
Vue.use(FileManagerPlugin);
Vue.use(ButtonPlugin);
export default {
  data () {
    return {
      ajaxSettings:
      {
        url: "https://ej2-aspcore-
service.azurewebsites.net/api/FileManager/FileOperations",
        getImageUrl: "https://ej2-aspcore-
service.azurewebsites.net/api/FileManager/GetImage",
        uploadUrl: "https://ej2-aspcore-
service.azurewebsites.net/api/FileManager/Upload",
        downloadUrl: "https://ej2-aspcore-
service.azurewebsites.net/api/FileManager/Download"
      },
      view: "Details",
      height: "330px"
    };
  },
  provide: {
    filemanager: [DetailsView, NavigationPane, Toolbar]
  },
  methods: {
    onCreated: function(args){
      // Click event for enable button
      document.getElementById("enable").addEventListener('click',
(event) => {
        // Enable new folder toolbar item

this.$refs.fileManagerinstance.enableToolbarItems(["newfolder"]);
      });
      // Click event for disable button
      document.getElementById("disable").addEventListener('click',
(event) => {
        // Disable new folder toolbar item

this.$refs.fileManagerinstance.disableToolbarItems(["newfolder"]);
      });
    }
  }
}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-icons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-inputs/styles/material.css";
@import "../node_modules/@syncfusion/ej2-popups/styles/material.css";
@import "../node_modules/@syncfusion/ej2-buttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-splitbuttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-navigations/styles/material.css";
@import "../node_modules/@syncfusion/ej2-layouts/styles/material.css";

```

```
@import "../node_modules/@syncfusion/ej2-grids/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-
filemanager/styles/material.css";
#enable {
    position: relative;
    left: 20%;
    top: -10px;
}
#disable {
    position: relative;
    left: 30%;
    top: -10px;
}
</style>
```

{% previewsample "page.domainurl/code-snippet/file-manager/toolbar-items-cs1" %}

### Customize custom thumbnail in Vue File manager component

The default appearance of the file manager can customize with your own icon by using [showThumbnail](#) property.

The following example demonstrate how to add a custom icon in largeicons view.

#### APP.VUE

```
<template>
  <div id="app">
    <ejs-filemanager id="file-manager" :showThumbnail="showThumbnail"
    :ajaxSettings="ajaxSettings">
      </ejs-filemanager>
    </div>
  </template>
<script>
import Vue from "vue";
import { FileManagerPlugin, DetailsView, NavigationPane, Toolbar } from
"@syncfusion/ej2-vue-filemanager";
Vue.use(FileManagerPlugin);
export default {
  data () {
    return {
      ajaxSettings:
      {
        url: "https://ej2-aspcore-
service.azurewebsites.net/api/FileManager/FileOperations",
        getImageUrl: "https://ej2-aspcore-
service.azurewebsites.net/api/FileManager/GetImage",
        uploadUrl: "https://ej2-aspcore-
service.azurewebsites.net/api/FileManager/Upload",
        downloadUrl: "https://ej2-aspcore-
service.azurewebsites.net/api/FileManager/Download"
      },
      showThumbnail: false
    };
  },
  provide: {
    filemanager: [DetailsView, NavigationPane, Toolbar]
```

```

    }
  }
</script>
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-icons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-inputs/styles/material.css";
@import "../node_modules/@syncfusion/ej2-popups/styles/material.css";
@import "../node_modules/@syncfusion/ej2-buttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-splitbuttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-navigations/styles/material.css";
@import "../node_modules/@syncfusion/ej2-layouts/styles/material.css";
@import "../node_modules/@syncfusion/ej2-grids/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-
filemanager/styles/material.css";
.e-filemanager .e-large-icons .e-fe-image {
  background-image:
url("data:image/svg+xml;base64,PD94bWwgdmVyc2lvdj0iMS4wIiBlbmNvZGluZz0idXRmL
TgiPz48c3ZnIHZlcnNpb249IjEuMSIgaWQ9IkkxheWVYXzEiIHhtbG5zPSJodHRwOi8vd3d3LnczL
m9yZy8yMDAwL3N2ZyIgeG1sbnM6Gxpbms9Imh0dHA6Ly93d3d3LnczLm9yZy8yMDAwL3N2ZyI
Hg9IjBweCIgeT0iMHB4IiB2aWV3Qm94PSIwIDAgMzIgMzIiIHNoeWxlPSJlbmFibGUtYmFja2dyb
3VuZDpuZXcgMCAwIDMyIDMyOyIgeG1sOnNwYWNlPSJwcmVzZXJ2ZSI+PHN0eWxlIHR5cGU9InRle
HQvY3NzIj4uc3Qwe2ZpbGw6I0ZGOTI5Mj9LnN0MTtmaWxsOiNFODdFN0U7fS5zdDj7ZmlsbDojR
kZDM0Mz030uc3Qze2ZpbGw6IzIxRDRGRt9LnN0NHTmaWxsOiM2M0E3RDM7fS5zdDj7ZmlsbDojQ
zFFN0ZG030uc3Qze2ZpbGw6I0ZGRkZGRj9LnN0N3tmaWxsOiM4M0Q2Qjk7fS5zdDj7ZmlsbDojN
DZDNjhDO30uc3Q5e2ZpbGw6I0JCRTHeODt9LnN0MTB7ZmlsbDojRkVCMTdEO30uc3QxMXtmaWxsO
iNERDk2NjY7fS5zdDEye2ZpbGw6I0ZFRDRCNzt9LnN0MTN7ZmlsbDojRjJBMkEyO30uc3QxNHTma
WxsOiNGMUM1QzU7fS5zdDEle2ZpbGw6I0RCQjY2Mzt9LnN0MTZ7ZmlsbDojQ0VBMTUxO30uc3QxN
3tmaWxsOiNFQkQ3QTk7fS5zdDE4e2ZpbGw6I0NFQ0VDRt9LnN0MTl7ZmlsbDojQjdCN0I3O30uc
3QyMHTmaWxsOiNFNEU0RTQ7fS5zdDIxe2ZpbGw6IzY1QUFEMTt9LnN0MjJ7ZmlsbDojRTU3QTDBo
30uc3QyM3tmaWxsOiNFNkE2RTg7fS5zdDI0e2ZpbGw6I0Q2OEFENjt9LnN0MjV7ZmlsbDojRkZDQ
0ZFO30uc3QyNntmaWxsOiM5OENFNUY7fS5zdDI3e2ZpbGw6IzhDQUYyQzt9LnN0MjV7ZmlsbDojQ
zZFM0E3O30uc3QyOXTmaWxsOiNGRkI1Nzg7fS5zdDMwe2ZpbGw6I0VEOUY2NDt9LnN0MzF7Zmlsb
DojRkZENkI1O30uc3QzMntmaWxsOiNGNEExRUY7fS5zdDMze2ZpbGw6I0REODdERDt9LnN0MzR7Z
mlsbDojRjldQkY2O30uc3QzNXtmaWxsOiNBOEEYRjQ7fS5zdDM2e2ZpbGw6Izg4ODVFODt9LnN0M
zd7ZmlsbDojQ0ZDQ0Y4O30uc3QzOHTmaWxsOiNCQ0JDQkM7fS5zdDM5e2ZpbGw6I0E4QThBODt9L
nN0NDB7ZmlsbDojREFEQURBO30uc3Q0MXtmaWxsOiM3N0NDREI7fS5zdDQye2ZpbGw6IzREQkNDM
Tt9LnN0NDN7ZmlsbDojQjRFM0VCO30uc3Q0NHTmaWxsOiNGRkI3QTQ7fS5zdDQle2ZpbGw6I0Y2O
UE3Qjt9LnN0NDZ7ZmlsbDojRkZEN0NEO30uc3Q0N3tmaWxsOiM3MUM4RjQ7fS5zdDQ4e2ZpbGw6I
zhEQzk3Nzt9LnN0NDl7ZmlsbDojN0NBODUxO30uc3Q1MHTvcGFjaXR5OjAuNDU7ZmlsbDojRkZGR
kZG0308L3N0eWxlPjxnPjxnPjxwYXR0IGNsYXNzPSJzdDAiIGQ9Ik0yMS41LDAuNWgtMTdjLTEuM
SwwLTIsMCA45LTIsMnYyN2MwLDEuMSwwLjksMiwYLDJoMjNjMS4xLDAuMS4wLjksMiwYdi0yMUwyM
S41LDAuNXoiLz48GF0aCBjbGFzc20ic3QxIiBkPSJNMjcunSwzMmgtMjNDMy4xLDMYLDI5MzAuO
SwyLDI5LjV2LTI3QzIsMS4xLDMuMSwwLjQuNSwwaDE3LjJMMzAsOC4zdjIxLjJDMzAsMzAuOSwyO
C45LDMYLDI5LjUsMzJ6IE00LjUsMUMzLjcsMSwwLDEuNywzLDEuNXYYN0MzLDMwLjMsMy43LDMxL
DQuNSwwMwgyM2MwLjgsMCAwLjUtMCA43LDEuNS0xLjVWOC43TDIxLjMsMUG0LjV6Ii8+PC9nPjxnP
jxwYXR0IGNsYXNzPSJzdDIiIGQ9Ik0yMS41LDAuNXYY3YzAsMCA42LDAuNCwxLDEsMwgy3TDIxLjUsM
C41eiIvPjxwYXR0IGNsYXNzPSJzdDEiIGQ9Ik0yOS41LdLoLTDdMjEuNyw5LDEuXLDguMywyMSw3L
jV2LTDjMCAwLjIsMCA4xLTAuNCwwLjMtMCA41YzAuMi0wLjEsMCA40LDAuMCA41LDAuMwW4LDhDMzAsO
C4zLDMwLjDguNSwwMCAwLjddMjkuOSw4LjksMjkuNyw5LDEuXLDguXLDguXLDguXLDguXLDguXLDgu
y44LDIyLjIsOCwyMi41LDhONS44TDIyLDEuN3oiLz48L2c+PC9nPjxnPjxwYXR0IGNsYXNzPS
SJzdDYiIHBvaW50cz0iNSwyNCAuMCA42LDE3LjIgMTQuOCwyMS44LDIxLjQsMTMgMjcsMjQgIi8+P
C9nPjwvc3ZnPg==");
  }

.e-filemanager .e-large-icons .e-fe-music {

```

```
background-image:
url("data:image/svg+xml;base64,PD94bWwgdmVyc2lvbj0iMS4wIiBlbmNvZGluZz0idXRmL
TgiPz48c3ZnIHZlcnNpb249IjEuMSIgaWQ9IkhxheWVyXzEiIHhtbG5zPSJodHRwOi8vd3d3LnczL
m9yZy8yMDAwL3N2ZyIgeG1sbnM6eGxpbnM9Imh0dHA6Ly93d3cudzMub3JnLzE5OTkveGxpbnMsiI
Hg9IjBweCIgeT0iMHB4IiB2aWV3Qm94PSIwIDAgMzIgMzIiIHNoeWxlPSJlbmFibGUTYmFja2dyb
3VuZDpuZXCgMCAwIDMyIDMyOyIgeG1sOnNwYWNlPSJwcmVzZXJ2ZSI+PHN0eWxlIHR5cGU9InRle
HQvY3NzIj4uc3Qwe2ZpbGw6I0ZGOTI5Mjt9LnN0MXtmaWxsOiNFODdFN0U7fS5zdDJ7ZmlsbDojR
kZDM0MzO30uc3Qze2ZpbGw6IzIxRDRGRt9LnN0NHTmaWxsOiM2M0E3RDM7fS5zdDV7ZmlsbDojQ
zFFN0ZGO30uc3Q2e2ZpbGw6I0ZGRkZGRjt9LnN0N3tmaWxsOiM4M0Q2Qjk7fS5zdDh7ZmlsbDojN
DZDNjhDO30uc3Q5e2ZpbGw6I0JCRTHeODt9LnN0MTB7ZmlsbDojRkVCMTdEO30uc3QxMXtmaWxsO
iNERDk2NjY7fS5zdDEye2ZpbGw6I0ZFRDRCNzt9LnN0MTN7ZmlsbDojRjJBMkEyO30uc3QxNHTma
WxsOiNGMUM1QzU7fS5zdDEle2ZpbGw6I0RCQjY2Mzt9LnN0MTZ7ZmlsbDojQ0VBMTUxO30uc3QxN
3tmaWxsOiNFQkQ3QTk7fS5zdDE4e2ZpbGw6I0NFQ0VDRt9LnN0MTl7ZmlsbDojQjdCN0I3O30uc
3QyMHTmaWxsOiNFNEU0RTQ7fS5zdDIxe2ZpbGw6IzY1QUFEMTt9LnN0MjJ7ZmlsbDojRTU3QTDBo
30uc3QyM3tmaWxsOiNFNkE2RTg7fS5zdDI0e2ZpbGw6I0Q2OEFENjt9LnN0MjV7ZmlsbDojRkZDQ
0ZFO30uc3QyNntmaWxsOiM5OENFNUY7fS5zdDI3e2ZpbGw6IzhDQUYyQzt9LnN0Mjh7ZmlsbDojQ
zZFM0E3O30uc3QyOXtmaWxsOiNGRkI1Nzg7fS5zdDMwe2ZpbGw6I0VEOUY2NDt9LnN0MzF7Zmlsb
DojRkZENkI1O30uc3QzMntmaWxsOiNGNEExRUY7fS5zdDMze2ZpbGw6I0REODdERDt9LnN0MzR7Z
mlsbDojRjldQkY2O30uc3QzNXtmaWxsOiNBOEEYRjQ7fS5zdDM2e2ZpbGw6Izg4ODVFODt9LnN0M
zd7ZmlsbDojQ0ZDQ0Y4O30uc3QzOHTmaWxsOiNCQ0JDQkM7fS5zdDM5e2ZpbGw6I0E4QThBODt9L
nN0NDB7ZmlsbDojREFEQURBO30uc3Q0MXtmaWxsOiM3N0NDREI7fS5zdDQye2ZpbGw6IzREQkNDM
Tt9LnN0NDN7ZmlsbDojQjRFM0VCO30uc3Q0NHTmaWxsOiNGRkI3QTQ7fS5zdDQle2ZpbGw6I0Y2O
UE3Qjt9LnN0NDZ7ZmlsbDojRkZEN0NEO30uc3Q0N3tmaWxsOiM3MUM4RjQ7fS5zdDQ4e2ZpbGw6I
zhEQzk3Nzt9LnN0NDl7ZmlsbDojN0NBODUxO30uc3Q1MHTvcGFjaXR5OjAuNDU7ZmlsbDojRkZGR
kZGO308L3N0eWxlPjxnPjxwYXRoIGNsYXNzPSJzdDmiIGQ9Ikh0YMS41LDAuNWgtMTdjLTEuMSswL
TIsMCA45LTIsMnYyN2MwLDEuMSswLjksMiwYLDJoMjNjMS4xLDAsMi0wLjksMi0ydi0yMUwyMS41L
DAuNXoiLz48cGF0aCBjbGFzc0ic3QyMSIgdD0iTTI3LjUsMzJoLTIzQzMuMSwzMiwYLDJksM
iwyOS41di0yN0MyLDEuMSwzLjEsMCAwLjUsMGgxNy4yYTMwLdguM3YyMS4yQzMwLdMwLjksMjguO
SwzMiwYyNj41LDMyeiBNNC41LDFDMy43LDEsMywYLDJksMiwYyLjV2MjdmYwzMC4zLDMuNywzMSw0L
jUsMzFoMjNjMCA44LDAsMS41LTAuNywYLDJksMiwYyLjV2MjdmYwzMC4zLDFINC41eiIvPjwvZz48Zz48c
GF0aCBjbGFzc0ic3Q1IiBkPSJNMjEuNSswLjV2N2MwLdAuNiwwLjQsMSwzLDFoN0wyMS41LDAuN
XoiLz48cGF0aCBjbGFzc0ic3QyMSIgdD0iTTI5LjUsOWgtN0MyMS43LdksMjEsOC4zLdIXLdCuN
XYtN2MwLTAuMiwwLjEtMCA40LDAuMy0wLjVjMCA4yLTAuMSswLjQsMCAwLjUsMCA4xbDgsOEMzMCAwL
jMsMzAsOC41LDMwLdguN0MyOS45LdguOSwyOS43LdksMjkuNSw5eiBNMjIsMS43djUuOEMyMiwwL
jgsMjIuMiwwLjIyLjUsOGg1LjhMMjIsMS43eiIvPjwvZz48Zz48cGF0aCBjbGFzc0ic3Q2IiBkP
SJNMTIsMTQuOXY3LjZjLTAuNi0wLjUtMS40LTAuNy0yLjMtMCA40Yy0wLjgsMCA4yLTAuNCwwLjktM
S42LDEuN2MtMCA41LDEuOCwxLDMuNSwyLjgsMy4yYzEuMi0wLjIsMi4xLTAuNCwyLjEtMi42bDAtN
i40bDEwLTAuOXY0LjNjLTAuNi0wLjUtMS40LTAuNy0yLjMtMCA40Yy0wLjgsMCA4yLTAuNCwwLjktM
S42LDEuN2MtMCA41LDEuOCwxLDMuNSwyLjgsMy4yYzEuMi0wLjIsMi4xLTAuNCwyLjEtMi42bDAtO
S4zYzAtMCA42LTAuNi0xLjEtMS4yLTFsLTAuLDEuOUMxMi4zLDE0LDEyLDE0LjQsMTIsMTQuOXoiL
z48L2c+PC9zdmc+");
}

.e-filemanager .e-large-icons .e-fe-xlsx {
background-image:
url("data:image/svg+xml;base64,PD94bWwgdmVyc2lvbj0iMS4wIiBlbmNvZGluZz0idXRmL
TgiPz48c3ZnIHZlcnNpb249IjEuMSIgaWQ9IkhxheWVyXzEiIHhtbG5zPSJodHRwOi8vd3d3LnczL
m9yZy8yMDAwL3N2ZyIgeG1sbnM6eGxpbnM9Imh0dHA6Ly93d3cudzMub3JnLzE5OTkveGxpbnMsiI
Hg9IjBweCIgeT0iMHB4IiB2aWV3Qm94PSIwIDAgMzIgMzIiIHNoeWxlPSJlbmFibGUTYmFja2dyb
3VuZDpuZXCgMCAwIDMyIDMyOyIgeG1sOnNwYWNlPSJwcmVzZXJ2ZSI+PHN0eWxlIHR5cGU9InRle
HQvY3NzIj4uc3Qwe2ZpbGw6I0ZGOTI5Mjt9LnN0MXtmaWxsOiNFODdFN0U7fS5zdDJ7ZmlsbDojR
kZDM0MzO30uc3Qze2ZpbGw6IzIxRDRGRt9LnN0NHTmaWxsOiM2M0E3RDM7fS5zdDV7ZmlsbDojQ
zFFN0ZGO30uc3Q2e2ZpbGw6I0ZGRkZGRjt9LnN0N3tmaWxsOiM4M0Q2Qjk7fS5zdDh7ZmlsbDojN
DZDNjhDO30uc3Q5e2ZpbGw6I0JCRTHeODt9LnN0MTB7ZmlsbDojRkVCMTdEO30uc3QxMXtmaWxsO
iNERDk2NjY7fS5zdDEye2ZpbGw6I0ZFRDRCNzt9LnN0MTN7ZmlsbDojRjJBMkEyO30uc3QxNHTma
WxsOiNGMUM1QzU7fS5zdDEle2ZpbGw6I0RCQjY2Mzt9LnN0MTZ7ZmlsbDojQ0VBMTUxO30uc3QxN
3tmaWxsOiNFQkQ3QTk7fS5zdDE4e2ZpbGw6I0NFQ0VDRt9LnN0MTl7ZmlsbDojQjdCN0I3O30uc
```

3QyMHTmaWxsOiNFNEU0RTQ7fS5zdDIxe2ZpbGw6IzY1QUFEMTt9LnN0MjJ7ZmlsbDoJRtU3QTdBo  
30uc3QyM3tmaWxsOiNFNkeE2Rtg7fS5zdDI0e2ZpbGw6IOQ2OEFENjt9LnN0MjV7ZmlsbDoJRkZDQ  
0ZF030uc3QyNntmaWxsOiM5OENFNUY7fS5zdDI3e2ZpbGw6IzhDUYyQzt9LnN0Mjh7ZmlsbDoJQ  
zzFM0E3O30uc3QyOXtmaWxsOiNGRki1Nzg7fS5zdDMwe2ZpbGw6IOVEOUY2NDt9LnN0Mzf7Zmlsb  
DoJRkZENkI1030uc3QzMntmaWxsOiNGNEExRUy7fS5zdDMze2ZpbGw6IOREODdERdt9LnN0MZR7Z  
mlsbDoJRjlDQky2O30uc3QzNXtmaWxsOiNB0EEYrJq7fS5zdDM2e2ZpbGw6Izg4ODVFODt9LnN0M  
zd7ZmlsbDoJQ0ZDQ0Y4O30uc3QzOHtmaWxsOiNCQ0JDQkm7fS5zdDM5e2ZpbGw6IOE4QThBODt9L  
nN0NDB7ZmlsbDojREFEQURBO30uc3Q0MXtmaWxsOiM3N0NDREI7fS5zdDQye2ZpbGw6IzREQkNDM  
Tt9LnN0NDN7ZmlsbDojQjRFM0VCO30uc3Q0NHtmaWxsOiNGRki3QTQ7fS5zdDQle2ZpbGw6IOY2O  
UE3Qjt9LnN0NDZ7ZmlsbDojRkZEN0EO30uc3Q0N3tmaWxsOiM3MUM4Rjq7fS5zdDQ4e2ZpbGw6I  
zhEqzk3Nzt9LnN0NDl7ZmlsbDojN0NBODuxO30uc3QlMhtvcGFjaXR5OjAuNDU7ZmlsbDojRkZGR  
kZGO308L3NOEWxlpjxnPjxnPjxwYXRoIGNsYXNzPSJzdDciIGQ9Ik0yMS41LDAuNWgtMTdjLTEuMR  
SwWLTIsMC45LTIsmNyYn2MwLDEuMSswLjksMiwyLDJoMjNmjs4xLDASmi0wLjksMi0ydi0yMUwyM  
S41LDAuNXoiLz48cGF0aCBjbGFzc2c0ic3Q4IiBkPSJNMjcunSwzMmgtMjNDMy4xLDMYLDIismZAUo  
SwyLDI5LjV2LTI3QzIsMS4xLDMuMSswLDQuNSwwaDE3LjJMMzAsOC4zdjIxLjJDMzAsMzAuOSwyO  
C45LDMYLDI3LjUsMzJ6IE00LjUsMUMzLjksMSwzLDEuNywzLDIuNXYYn0MzLDMwLjMsMy43LDMxL  
DQuNSwzMWgyM2MwLjgsMCMwLjUtMC43LDEuNS0xLjVWOC43TDIxLjMsMuG0LjV6Ii8+PC9nPjxnP  
jxwYXRoIGNsYXNzPSJzdDkiIGQ9Ik0yMS41LDAuNXYYn0MzAsMC42LDAuNCwxLDESWMwg3TDIxLjUsM  
C4leiIvPjxwYXRoIGNsYXNzPSJzdDgiIGQ9Ik0yOS41LDloLTDdmJEuNyw5LDIxLDguMywyMSw3I  
jV2LTdjMCOwLjIsMC4xLTAuNCwwLjMtMC41YZAuMi0wLjEsMC40LDAuSMC41LDAuMWW4LdhDMzAsO  
C4zLDMwLDguNSwzMCMw4LjdDMjkuOSw4LjksMjkunyw5LDI5LjUsOXogTTIyLDEuN3Y1LjhDMjIsN  
y44LDIyLjIsOCwyMi41LDhoNS44TDIyLDEuN3oiLz48L2c+PGc+PHJlY3QgeD0iNyIgeT0iMTGiI  
GNsYXNzPSJzdDYiIHdpZHROPSixNyIgaGVpZ2h0PSiyIi8+PC9nPjxnPjxyZWNOIHg9IjkiIHk9I  
je4IiB0cmFuc2Zvc09ImldhdHJpeCgtMS44MzY5NzBlLTE2IDEgLTEgLTEuODM2OTcwZS0xNiAzN  
C45OTk5IDMuMDAwMSkiIGNsYXNzPSJzdDYiIHdpZHROPSixNCIgaGVpZ2h0PSiyIi8+PC9nPjxnP  
jxwYXRoIGNsYXNzPSJzdDYiIGQ9Ik0yNSwyNkg3VjEyade4VjI2eiBNOSwyNGgxNfYxNEG5Vji0e  
iiVpjvwZz48L2c+PC9zdmc+") ;

}

```
.e-filemanager .e-large-icons .e-fe-video {
  background-image:
url("data:image/svg+xml;base64,PD94bWwgdMvYvc2lvdj0iMS4wIiB1bmVzZGluZz0idXRmL
TgiPz48c3ZnIHZ1cnNpb249IjEuMSIgaWQ9IktxeHVWYXZlIiHtbG5zPSJodHRWoi8vd3d3LnczL
m9yZy8yMDAwL3N2ZyIgeG1sbnM6eGxpbnM5Imh0dHA6Ly93d3duc2Mub3JnLzE5OTkveGxpbnM5I
Hg9IjBweCIgeT0iMHBA4IiB2aWV3Qm94PSIwIDAgMzIgMzIiIHNoeWxlPSJlbmFibGUTYmFja2dyb
3VuZDpuZXCgMCAwIDMyIDMyOyIgeG1sOnNwYWNlPSJwcmVzZXJ2ZSI+PHNoeWxlIHR5cGU9InRle
HQvY3NzIj4uc3Qwe2ZpbGw6I0ZGOTI5Mjt9LnN0MXtmaWxsOiNFODdFN0U7fS5zdDJ7ZmlsbDoJR
kZDM0Mz030uc3Qze2ZpbGw6IzIxRDRGRt9LnN0NhtmaWxsOiM2M0E3RDM7fS5zdDV7ZmlsbDoJQ
zFFN0ZGO30uc3Q2e2ZpbGw6I0ZGRkZGRj9LnN0N3tmaWxsOiM4M0Q2Qjk7fS5zdDh7ZmlsbDoJN
DZDNjhDO30uc3Q5e2ZpbGw6I0JCRThEODt9LnN0MTB7ZmlsbDoJRkVCMTE0O30uc3QxMXtmaWxsO
iNERdk2NjY7fS5zdDEye2ZpbGw6I0ZFRDRCNz9LnN0MTN7ZmlsbDoJRjJBMkEyO30uc3QxNhtma
WxsOiNGMUM1QzU7fS5zdDEle2ZpbGw6I0RCQjY2Mzt9LnN0MTZ7ZmlsbDoJQ0VBMtUxO30uc3QxN
3tmaWxsOiNFQkQ3Q3Tk7fS5zdDE4e2ZpbGw6I0NFQ0VDRt9LnN0MT17ZmlsbDoJQjdCN0I3O30uc
3QyMhtmaWxsOiNFNEU0RTQ7fS5zdDIxe2ZpbGw6IzY1QUFEMT9LnN0Mj7ZmlsbDoJRtU3QTdBO
30uc3QyM3tmaWxsOiNFNkE2RTg7fS5zdDI0e2ZpbGw6I0Q2OEFENjt9LnN0MjV7ZmlsbDoJRkZDQ
0ZFO30uc3QyNntmaWxsOiM5OENFNUY7fS5zdDI3e2ZpbGw6IzhDUYyQzt9LnN0Mjh7ZmlsbDoJQ
zZFM0E3O30uc3QyOXtmaWxsOiNGRkI1Nzg7fS5zdDMwe2ZpbGw6I0VEOUY2NDt9LnN0MzF7Zmlsb
DoJRkZENkI1O30uc3QzMntmaWxsOiNGNEEeRUY7fS5zdDMze2ZpbGw6I0REODdERDt9LnN0MzR7Z
mlsbDoJRjldQkY2O30uc3QzNXtmaWxsOiNB0EEYrjQ7fS5zdDM2e2ZpbGw6Izg4ODVFOdt9LnN0M
zd7ZmlsbDoJQ0ZDQ0Y4O30uc3QzOHtmaWxsOiNCQ0JDQkM7fS5zdDM5e2ZpbGw6I0E4QThBODt9L
nN0NDB7ZmlsbDoJREFEQURBO30uc3Q0MXtmaWxsOiM3N0NDREI7fS5zdDQye2ZpbGw6IzREQkNDM
Tt9LnN0NDN7ZmlsbDoJjRFM0VCO30uc3Q0NhtmaWxsOiNGRkI3QTQ7fS5zdDQle2ZpbGw6I0Y2O
UE3Qjt9LnN0NDN7ZmlsbDoJRkZEN0EO30uc3Q0N3tmaWxsOiM3MUM4RjQ7fS5zdDQ4e2ZpbGw6I
zhEQzk3Nzt9LnN0NDl7ZmlsbDoJjRkZj0N0BODUxO30uc3Q1MhtvcGFjaXR5OjAuNDU7ZmlsbDoJRkZGR
kZGO308L3N0eWxlPjxnPjxnPjxwYXRoIGNsYXNzPSJ3dDAiIGQ9Ikt0YMS41LDAuNwgtMtdjLTEuM
SwwLTI5MC45LTI5MnYyN2MwLDEuMSwwLjksMiwYLDJoMjNjMS4xLDAsMi0wLjksMi0ydi0yMUwyM
S41LDAuNXoiLz48cGF0aCBjbGFzc20ic3QyMiIgZD0iTTI3LjUsMzJ0LTlZzQzMwMuSwzMiwiYLDMwL
```



```

ksMiwyOS41di0yN0MyLDEuMSwzLjEsMCw0LjUsMGgxNy4yTDMwLDguM3YyMS4yQzMwLDMwLjksM
jguOSwzMiwyNy41LDMyeiBNNC41LDFDMY43LDEsMywxLjcsMywyLjV2MjdDMYwzMC4zLDMuNywzM
Sw0LjUsMzFoMjNjMC44LDASMS41LTAuNywxLjUtMs41VjguN0wyMS4zLDFINC41eiIvPjwvZz48Z
z48cGF0aCBjbGFzc0ic3QyIiBkPSJNMjEuNSwwLjV2N2MwLDAuNiwwLjQsMSwzLDFoN0wyMS41L
DAuNXoiLz48cGF0aCBjbGFzc0ic3QyMiIgZD0iTTI5LjUsOWgtN0MyMS43LDksMjEsOC4zLDIxL
DcuNXYtN2MwLTAuMiwWlJEtMC40LDAuMy0wLjVjMC4yLTAuMSwwLjQsMCwwLjUsMC4xbDgsOEMzM
Cw4LjMsMzAsOC41LDMwLDguN0MyOS45LDguOSwyOS43LDksMjkuNSw5eiBNMjIsMS43djUuOEMyM
iw3LjgsMjIuMiw4LDIyLjUsOGglLjhmMjIsMS43eiIvPjwvZz48Zz48cG9seWdviBjbGFzc0ic
3Q2IiBwb2ludHM9IjElLDE2LjUgMTksMjAgMTUsMjMuNSAiLz48L2c+PGc+PHBhdGggY2xhc3M9I
nN0NiIgZD0iTTIyLDI3SDEwYy0xLjcsMC0zLTEuMy0zLTN2LTJhMjM0xLjcsMS4zLTMsMy0zaDEy
zEuNywwLDMsMS4zLDMsM3Y4QzI1LDI1LjcsMjMuNywyNywyMiwYn3ogTTEwLDE1Yy0wLjYsMC0xL
DAuNC0xLDF2OGMwLDAuNiwwLjQsMSwzLDFoMTJjMC42LDASMS0wLjQsMS0xdi04YzAtMC42LTAuN
C0xLTETMuGxMHoilz48L2c+PC9nPjwvc3ZnPg==" );
}

.e-filemanager .e-large-icons .e-fe-pptx {
    background-image:
url ("data:image/svg+xml;base64,PD94bWwgdmlvY2lvdj0iMS4wIiBlbmVzZGluZz0idXRmL
TgiPz48c3ZnIHZ1cnNpb249IjEuMSIgaWQ9IkkheWVYXzEiIHhtbG5zPSJodHRwOi8vd3d3LnczL
m9yZy8yMDAwLjN2ZyIgeG1sbnM6Gxpbms9Imh0dHA6Ly93d3d3LnczLm9yZy8yMDAwLjN2ZyI
Hg9IjBweCIgeT0iMHb4IiB2aWV3Qm94PSIwIDAgMzIgMzIiIHNoeWxlPSJlbmFibGUTYmFja2dyb
3VuZDpuZXCgMCAwIDMyIDMyOyIgeG1sOnNwYWNlPSJwcmVzZXJ2ZSI+PHN0eWxlIHR5cGU9InRle
HQvY3NzIj4uc3Qwe2ZpbGw6I0ZGOTI5Mj9LnN0MXtmaWxsOiNFODdFN0U7fS5zdDJ7ZmlsbDoJR
kZDM0Mz030uc3Qze2ZpbGw6IzkkXRDRGRt9LnN0NhtmaWxsOiM2M0E3RDM7fS5zdDV7ZmlsbDoJQ
zFFN0ZG030uc3Q2e2ZpbGw6I0ZGRkZGRj9LnN0N3tmaWxsOiM4M0Q2Qjk7fS5zdDh7ZmlsbDoJN
DZDNjhDO30uc3Q5e2ZpbGw6I0JCRTI0EOD9LnN0MTB7ZmlsbDoJRkVCMtDEO30uc3QxMXtmaWxsO
iNERDk2NjY7fS5zdDEye2ZpbGw6I0ZFRDRCNzt9LnN0MTN7ZmlsbDoJRjJBMcEY030uc3QxNhtma
WxsOiNGNMUmqZU7fS5zdDE1e2ZpbGw6I0RCQjY2Mzt9LnN0MTZ7ZmlsbDoJQ0VBMTUxO30uc3QxN
3tmaWxsOiNFQkQ3Q3Tk7fS5zdDE4e2ZpbGw6I0NFQ0VDRt9LnN0MT17ZmlsbDoJQjdCN0I3O30uc
3QyMhtmaWxsOiNFNEU0RTQ7fS5zdDIxe2ZpbGw6IzY1QUFEMT9LnN0MjJ7ZmlsbDoJRtU3QTdBO
30uc3QyM3tmaWxsOiNFNke2RTg7fS5zdDI0e2ZpbGw6I0Q2OEFENj9LnN0MjV7ZmlsbDoJRkZDQ
0ZFO30uc3QyNntmaWxsOiM5OENFNUY7fS5zdDI3e2ZpbGw6IzhDQUYyQzt9LnN0Mjh7ZmlsbDoJQ
zZFM0E3O30uc3QyOXtmaWxsOiNGRkI1Nzg7fS5zdDMwe2ZpbGw6I0VEOUY2NDt9LnN0MzF7Zmlsb
DoJRkZENkI1O30uc3QzMntmaWxsOiNGNEEEXRUY7fS5zdDMze2ZpbGw6I0REODdERdt9LnN0MzR7Z
mlsbDoJRjldQkY2O30uc3QzNXtmaWxsOiNBOEEYRjQ7fS5zdDM2e2ZpbGw6Izg4ODVFOdt9LnN0M
zd7ZmlsbDoJQ0ZDQ0Y4O30uc3QzOhtmaWxsOiNCQ0JDQkm7fS5zdDM5e2ZpbGw6I0E4QThBODt9L
nN0NDB7ZmlsbDoJREFEQURBO30uc3Q0MXtmaWxsOiM3N0NDREI7fS5zdDQye2ZpbGw6IzREQkNDM
Tt9LnN0NDN7ZmlsbDoJjRFM0VCO30uc3Q0NhtmaWxsOiNGRkI3QTQ7fS5zdDQ1e2ZpbGw6I0Y2O
UE3Qjt9LnN0NDZ7ZmlsbDoJRkZEN0NEO30uc3Q0N3tmaWxsOiM3MUM4RjQ7fS5zdDQ4e2ZpbGw6I
zhEQzk3Nzt9LnN0ND17ZmlsbDojN0NBODUxO30uc3Q1MhtvcGFjaXR5OjAuNDU7ZmlsbDoJRkZGR
kZGO308L3N0eWxlPjxnPjxnPjxwYXRoIGNsYXNzPSJzdDEwIiBkPSJNMjEuNSwwLjV2LTJhMjM0x
LjEsMC0yLDAuNS0yLDJ2MjdjMCwwLjEsMC45LDI5IiwyaDIzYzEuMSwwLjDI5tMC45LDI5tMnYtMjFMM
jEuNSwwLjV6Ii8+PHBhdGggY2xhc3M9InN0MTEiIGQ9Ikk0YyNy41LDMyaC0yM0MzLjEsMzIsMiwz
M45LDI5MjkuNXYtMjdmIiwxLjEsMy4xLDASNC41LDBoMTcuMkwzMCw4LjN2MjEuMkMzMCwzMC45L
DI4LjksMzIsMjcuNSwzMnogTTQuNSwzQzMunYwxLDMsMS43LDMsMi41djI3QzMzMzAuMywzLjcsM
zEsNC41LDMxaDIzYzAuOCwwLDEuNS0wLjcsMS41LTEuNVY4LjdMMjEuMywzSDQuNXoiLz48L2c+P
Gc+PHBhdGggY2xhc3M9InN0MTIiIGQ9Ikk0yMS41LDAuNXY3YzAsMC42LDAuNCwwLDEsMWG3TDIxL
jUsMC41eiIvPjxwYXRoIGNsYXNzPSJzdDEwIiBkPSJNMjkuNSw5aC03QzIxLjcsOSwyMSw4LjMsM
jEsNy41di03YzAtMC4yLDAuMS0wLjQsMC4zLTAuNWwwLjItMC4xLDAuNCwwLDAuNSwwLjFsOCw4Q
zMwLDguMywzMCw4LjUsMzAsOC43QzI5LjksOC45LDI5LjcsOSwyOS41LDl6IE0yMiwxLjd2NS44Q
zIyLDcuOCwyMi4yLDgsMjIuNSw4aDUuOEwyMiwxLjd6Ii8+PC9nPjxnPjxyZWNOIHg9IjgiIHk9I
jI1IiBjbGFzc0ic3Q2IiB3aWR0aD0iMTciIGhlaWdodD0iMiIvPjwvZz48Zz48Zz48cGF0aCBjb
GFzc0ic3Q2IiBkPSJNMjIsMTYyUwgtNS4xVjExSDE3QzE5LjgsMTESMjIsMTMuMiwyMiwNi4xT
DIyLDE2LjF6Ii8+PC9nPjxnPjxwYXRoIGNsYXNzPSJzdDYiIGQ9Ikk0xNi45LDE4LjFfoLTJLTJ2L
TMuOGMtMi41LDAuNS00LjQsMy0zLjgsNS45YzAuNCwwLjgsMS45LDMuMywzLjcsMy43YzIuOCwwL
jYsNS4zLTEuMiwlLjktMy44SDE2Ljl6Ii8+PC9nPjwvZz48L2c+PC9zdmc=") ;
}

```

```
e-filemanager .e-large-icons .e-fe-rar {
background-image:
url("data:image/svg+xml;base64,PD94bWwgdmVyc2lvbj0iMS4wIiBlbmNvZGlucyZzOidXRmL
TgiPz48c3ZnIHZlcnNpb249IjEuMSIgaWQ9IGxheWVyXzEiIHhtbG5zPSJodHRwOi8vd3d3LnczL
m9yZy8yMDAwL3N2ZyIgeG1sbmM6eGxpbnMsImh0dHA6Ly93d3duc2Mub3JnLnZlE5OTkveGxpbnMiI
Hg9IjBweCIgeT0iMHBB4IiB2aWV3Qm94PSiwIDAgaGZlgMzIiIHNoeWxlPSJlbmfibGUtYmFja2dyb
3VuZDpuZXcgMCAwIDMyIDMyOyIgeG1sOnNwYWVN1PSJwcmlzc2ZSJ2ZSI+PHN0eWxlIHR5cGU9InRle
HQvY3NzIj4uc3Qwe2ZpbGw6IOZGOTI5Mjt9LnN0MXtmaWxsOiNFODdFN0U7fS5zdDJ7ZmlsbDoJR
kZDM0MzO30uc3Qze2ZpbGw6IzkxRDGRGTt9LnN0NHtmaWxsOiM2M0E3RDM7fS5zdDV7ZmlsbDoJQ
zFFNOZGO30uc3Q2e2ZpbGw6IOZGRkZGRjt9LnN0N3tmaWxsOiM4M0Q2Qjk7fS5zdDh7ZmlsbDoJN
DZNjhDO30uc3Q5e2ZpbGw6IOJCRTHeODt9LnN0MTB7ZmlsbDoJRkVCMTdeO30uc3QxMXtmaWxsOI
NERDK2NjY7fS5zdDEye2ZpbGw6IOZFDRDCNzt9LnN0MTN7ZmlsbDoJRjJBmkEyO30uc3QxNHTma
WxsOiNGMUM1QzU7fS5zdDE1e2ZpbGw6IORCQjY2Mzt9LnN0MTZ7ZmlsbDoJQ0VBMTUxO30uc3QxN
3tmaWxsOiNFQkQ3Q3Tk7fS5zdDE4e2ZpbGw6IOVFODdFRDt9LnN0MT17ZmlsbDoJQjdCN0I3O30uc
3QyMHTmaWxsOiNFNEU0RTQ7fS5zdDIxe2ZpbGw6IzY1QUFEMTt9LnN0MjJ7ZmlsbDoJRtU3QTdBO
30uc3QyM3tmaWxsOiNFNKER2RTg7fS5zdDI0e2ZpbGw6IOQ2OEFENjt9LnN0MjV7ZmlsbDoJRkZDQ
0ZFO30uc3QyNntmaWxsOiM5OENFNUY7fS5zdDI3e2ZpbGw6IzhDUYyQzt9LnN0Mjh7ZmlsbDoJQ
zZFM0E3O30uc3QyOXtmaWxsOiNGRkI1Nzg7fS5zdDMwe2ZpbGw6IOVEOUY2NDt9LnN0MzF7Zmlsb
DoJRkZENKI1O30uc3QzMntmaWxsOiNGNEEXRUy7fS5zdDMze2ZpbGw6IOREODderdt9LnN0MzR7Z
mlsbDoJRjldQKY2O30uc3QzNXtmaWxsOiNB0EEYrjQ7fS5zdDM2e2ZpbGw6Izg4ODVFODt9LnN0M
zd7ZmlsbDoJQ0ZDQ0Y4O30uc3QzOHtmaWxsOiNCQ0JDQkm7fS5zdDM5e2ZpbGw6IOE4QThBODt9L
nN0NDB7ZmlsbDoJREFEQURBO30uc3Q0MXtmaWxsOiM3N0NDREI7fS5zdDQye2ZpbGw6IzREqKNDM
Tt9LnN0NDN7ZmlsbDoJJRFM0VCO30uc3Q0NHtmaWxsOiNGRkI3QTQ7fS5zdDQ1e2ZpbGw6IOY2O
UE3Qt9LnN0NDZ7ZmlsbDoJRkZEN0NEO30uc3Q0N3tmaWxsOiM3MUM4RjQ7fS5zdDQ4e2ZpbGw6I
zheQzk3Nzt9LnN0ND17ZmlsbDoJN0NBODUxO30uc3Q1MhtvcGFjaXR5OjAuNDU7ZmlsbDoJRkZGR
kZO308L3N0eWxlPjxnPjxwYXR0IGNsYXNzPSJzdDI0IiBkPSJNMjkuNSwwLjVoLTE3YyOxLjEsM
C0YLDAuOS0YLjD2MjdYJMcwLjEsMC45LDIsMiwyadIZyZEmSwswLDITMC45LDITMnYjFMJMjEsN
SwwLjV6Ii8+PHBhdGggY2xhc3M9InN0MjQiIGQ9Ik0yNy41LDMyaC0yM0MzLjEsMzIsMjwzMCM45L
DISmjkuNXYtMjdmIwxLjEsMy4xLDAsNC41LDBoMTcuMkwzMCMw4LjN2MjEuUmKMzMCMwzMCM45LDI4L
jksMzIsMjcunSwzMnogTTQuNSwXQzMuNywxLDMsMS43LDMsMi41djI3QzMsMZAuMywzLjcsMzEsN
C41LDMxaDiZyZAUOCwWLDEuNS0wLjcsMS41LTEuNVY4LjldMMjEuMywXSDQuNXoiLz48L2c+PGc+P
HBhdGggY2xhc3M9InN0MjUiIGQ9Ik0yMS41LDAuNXY3YzAsMC42LDAuNCwXLDESMMWg3TDIXLjUsM
C4leiIvPjxwYXR0IGNsYXNzPSJzdDI0IiBkPSJNMjkuNSw5aC03QzIxLjcsOSwYMSw4LjMsMjEsN
y41di03YzAtMC4yLDAuMS0wLjQsMC4zLTAuNXMWLjQsMCcwLjUsMC4xbDgsOEMzMCMw4LjMsMzAsO
C41LDMwLDguNOMyOS45LDguOSwYOS43LDksMjkuNSw5eiBNMJIsMS43djUuOEMyMiW3LjgSMjIuM
iw4LDIyLjUsOGglLjhMMjIsMS43eiIvPjwvZz48Zz48cGF0aCBjbGFzc20ic3Q2IiBkPSJNOS44L
DE0LjhjMCM43LDAsMS4zLDAuMiwxLjcsMC41YzAuNCwWLjQsMC42LDAuOSwwLjYsMS43YzAsMC42L
TAuMSwXLTAuMywXLjRjLTAuMiwwLjQtMC42LDAuNi0xLjEsMC43djBjMCM40LDAuMSwwLjcsMC4yL
DAuOSwwLjRjMCM4yLDAuMiwwLjMsMC42LDAuNCwYzAsMC4yLDAuSMC4zLDAuSMC41czAsMC40LDAS
M42YzAsMC41LDASMC44LDAuMSwYzAsMC4yLDAuMiwwLjQsMC4zLDAuNXxyWjFoLTJEUOWMtMC4xL
TAuMS0wLjEtMC4zLTAuMi0wLjRjMCM0wLjIsMC0wLjLjMsMC0LjVsmCM0xLjZjMCM0wLjEtMC4xL
TAuNi0wLjMtMC44Yy0wLjItMC4yLTAuNC0wLjMtMC44LTAuM0g4LjN2My42SDYuNnYtOC42SDkuOHogT
TksMTGuNmMwLjQsMCcwLjctMC4xLDEtMC4zYzAuMi0wLjIsMC4zLTAuNSwwLjMtMWMwLTAuOC0wL
jQtMS4yLLEuMi0xLjJI0C4zdjIuNUg5eiIvPjxwYXR0IGNsYXNzPSJzdDYiIGQ9Ik0xNy4xLDE0L
jhsMi4zLDguNmgmtMS44bC0wLjQtMS44aC0yLjNsLTAuNCwXLjhoLTEuOGwyLjMtOC42SDE3LjF6I
EOxNi44LDIwLjFMFTYsMTYu2gwbC0wLjgSMY45SDE2Ljh6Ii8+PHBhdGggY2xhc3M9InN0NiIgZ
D0iTTEiZLjEsMTQuOGMwLjcsMCcwLjMsMC4yLDEuNywwLjVjMCM40LDAuNCwWLjYsMC45LDAuNiwxL
jdjMCMwLjYtMC4xLDEtMC4zLDEuNGMtMC4yLDAuNC0wLjYsMC42LTEuMSwwLjld2MGmwLjQsMC4xL
DAuNywwLjIsMC45LDAuNGMwLjIsMC4yLDAuMywwLjYsMC40LDFjMCMwLjIsMCcwLjMsMCcwLjVzM
CwwLjQsMCcwLjZjMCMwLjUsMCcwLjgSMC4xLDFjMCMwLjIsMC4yLDAuNCwWLjMsMC41djAuMWgtM
S45Yy0wLjEtMC4xLTAuMS0wLjMtMC4yLTAuNGMwLTAuMiwwLTAuMywwLTAuNWwwLTAuNmMwLTAuM
y0wLjEtMC42LTAuMy0wLjhhLTAuMi0wLjItMC40LTAuMy0wLjgtMC4zaC0wLjl2My42aC0xLjd2L
TguNkgYMy4xeiBNMJiuNCwXOC42YzAuNCwWLDAuNy0wLjEsMS0wLjNjMCM4yLTAuMiwwLjMtMC41L
DAuMy0xYzAtMC44LTAuNC0xLjItMS4yLLEuMmgmtMC44djIuNUgyMi40eiIvPjwvZz48L3N2Zz4="
```



```

    .e-filemanager .e-large-icons .e-fe-zip {
    background-image:
url("data:image/svg+xml;base64,PD94bWwgdmVyc2lvbj0iMS4wIiBlbmNvZGluZz0idXRmLTgiPz48c3ZnIHZlcnNpb249IjEuMSIgaWQ9IkxheWVyXzEiIHhtbG5zPSJodHRwOi8vd3d3LnczLm9yZy8yMDAwL3N2ZyIgeG1sbnM6eGxpbnM9Imh0dHA6Ly93d3d3cudzMub3JnLzE5OTkveGxpbnMsiIHg9IjBweCIgeT0iMHB4IiB2aWV3Qm94PSIwIDAgaWQ9IjEiIH0eWxlPSJlbmFibGUtYmFja2dyb3VuZDpuZXcgMCAwIDMyIDMyOyIgeG1sOnNwYWNlPSJwcmVzZXJ2ZSI+PHN0eWxlIHR5cGU9InRleHQvY3NzIj4uc3Qwe2ZpbGw6I0ZGOTI5Mjt9LnN0MXtmaWxsOiNFODdFN0U7fS5zdDJ7ZmlsbDojRkZDM0MzO30uc3Qze2ZpbGw6IzIxRDRGRt9LnN0NhtmaWxsOiM2M0E3RDM7fS5zdDV7ZmlsbDojQzFFN0ZGO30uc3Q2e2ZpbGw6I0ZGRkZGRjt9LnN0N3tmaWxsOiM4M0Q2Qjk7fS5zdDh7ZmlsbDojNDZDNjhDO30uc3Q5e2ZpbGw6I0JCRThEODt9LnN0MTB7ZmlsbDojRkVCMTdEO30uc3QxMXtmaWxsOiNERDk2NjY7fS5zdDEye2ZpbGw6I0ZFRDRCNzt9LnN0MTN7ZmlsbDojRjJBMkEyO30uc3QxNhtmaWxsOiNGMUM1QzU7fS5zdDEle2ZpbGw6I0RCQjY2Mzt9LnN0MTZ7ZmlsbDojQ0VBMTUxO30uc3QxN3tmaWxsOiNFQkQ3QTk7fS5zdDE4e2ZpbGw6I0NFQ0VDRt9LnN0MTl7ZmlsbDojQjdcN0I3O30uc3QyMHTmaWxsOiNFNEU0RTQ7fS5zdDIxe2ZpbGw6IzY1QUFEMTt9LnN0MjJ7ZmlsbDojRTU3QTdBO30uc3QyM3tmaWxsOiNFNke2RTg7fS5zdDI0e2ZpbGw6I0Q2OEFENjt9LnN0MjV7ZmlsbDojRkZDQ0ZFO30uc3QyNntmaWxsOiM5OENFNUY7fS5zdDI3e2ZpbGw6IzhdQUYyQzt9LnN0Mjh7ZmlsbDojQzZFM0E3O30uc3QyOXtmaWxsOiNGRkI1Nzg7fS5zdDMwe2ZpbGw6I0VEOUY2NDt9LnN0MzF7ZmlsbDojRkZENkI1O30uc3QzMntmaWxsOiNGNEExRUY7fS5zdDMze2ZpbGw6I0REODdERDt9LnN0MzR7ZmlsbDojRjldQkY2O30uc3QzNXtmaWxsOiNB0EEYRjQ7fS5zdDM2e2ZpbGw6IzgzODVFOdt9LnN0Mzd7ZmlsbDojQ0ZDQ0Y4O30uc3QzOHTmaWxsOiNCQ0JDQkM7fS5zdDM5e2ZpbGw6I0E4QTbODt9LnN0ND7ZmlsbDojREFEFURBO30uc3Q0MXtmaWxsOiM3N0NDREI7fS5zdDQye2ZpbGw6IzREQkNDMTt9LnN0NDN7ZmlsbDojQjRFM0VCO30uc3Q0NhtmaWxsOiNGRkI3QTQ7fS5zdDQle2ZpbGw6I0Y2OUE3Qjt9LnN0NDZ7ZmlsbDojRkZEN0NEO30uc3Q0N3tmaWxsOiM3MUM4RjQ7fS5zdDQ4e2ZpbGw6IzhEQzk3Nzt9LnN0NDl7ZmlsbDojN0NBODUxO30uc3Q1MHTvcGFjaXR5OjAuNDU7ZmlsbDojRkZGRkZGO308L3N0eWxlPjxnPjxnPjxwYXRoIGNsYXNzPSJzdDE1IiBkPSJNMjEuNSwLjVoLTE3Yy0xLjEsMC0yLDAuOS0yLDJ2MjdjMCwLjEsMC45LDIsMiwyaDIzYzEuMSwLdItMC45LDItMnYtMjFMMjEuNSwLjV6Ii8+PHBhdGggY2xhc3M9InN0MTYiIGQ9Ik0yNy41LDMyaC0yM0MzLjEsMzIsMiwzMCA45LDIsMjkuNXYtMjdmIwLjEsMy4xLDA5NC41LDBoMTcuMkwzMCA4LjN2MjEuMkMzMCA45LDI4LjksMzIsMjkuNSwzMnogTTQuNSwzQzMuNywxLDMsMS43LDMsMi41djI3QzMsMzAuMywzLjcsMzEsNC41LDMxaDIzYzAuOCwLDEuNS0wLjcsMS41LDEuNVY4LjdMMjEuMywzSDQuNXoiLz48L2c+PGc+PHBhdGggY2xhc3M9InN0MTciIGQ9Ik0yMS41LDAuNXY3YzAsMC42LDAuNCwLDEsMWg3TDIxLjUsMC41eiIvPjxwYXRoIGNsYXNzPSJzdDE2IiBkPSJNMjkuNSw5aC03QzIxLjcsOSwzMSw4LjMsMjEuNSw541di03YzAtMC4yLDAuMS0wLjQsMC4zLTAuNXMwLjQsMCwLjUsMC4xbDgsOEMzMCA4LjMsMzAsOC41LDMwLDguN0MyOS45LDguOSwzOS43LDMsMjkuNSw5eiBNMjIsMS43djUuOEMyMiw3LjgsMjEuMiw4LDIyLjUsOGg1LjhmMjIsMS43eiIvPjwvZz48Zz48cmVjdCB4PSI3IiB5PSI4IiBjbGFzc20ic3Q2IiB3aWR0aD0iMTMiIGhlaWdodD0iMiIvPjwvZz48Zz48cG9seWdvbiBjbGFzc20ic3Q2IiBwb2ludHM9IjEzLjUsMjkgNywyMyAyMCwyMyAiLz48L2c+PGc+PHJlY3QgeD0iNyIgeT0iMTMiIGNsYXNzPSJzdDYiIHdpZHRoPSIyIgaWQ9Vz2h0PSIyIi8+PC9nPjxnPjxyZWNOIHg9IjciIHk9IjE4IiBjbGFzc20ic3Q2IiB3aWR0aD0iMTMiIGhlaWdodD0iMiIvPjwvZz48L2c+PC9zdmc+");
    }

    .e-filemanager .e-large-icons .e-fe-txt {
    background-image:
url("data:image/svg+xml;base64,PD94bWwgdmVyc2lvbj0iMS4wIiBlbmNvZGluZz0idXRmLTgiPz48c3ZnIHZlcnNpb249IjEuMSIgaWQ9IkxheWVyXzEiIHhtbG5zPSJodHRwOi8vd3d3LnczLm9yZy8yMDAwL3N2ZyIgeG1sbnM6eGxpbnM9Imh0dHA6Ly93d3d3cudzMub3JnLzE5OTkveGxpbnMsiIHg9IjBweCIgeT0iMHB4IiB2aWV3Qm94PSIwIDAgaWQ9IjEiIH0eWxlPSJlbmFibGUtYmFja2dyb3VuZDpuZXcgMCAwIDMyIDMyOyIgeG1sOnNwYWNlPSJwcmVzZXJ2ZSI+PHN0eWxlIHR5cGU9InRleHQvY3NzIj4uc3Qwe2ZpbGw6I0ZGOTI5Mjt9LnN0MXtmaWxsOiNFODdFN0U7fS5zdDJ7ZmlsbDojRkZDM0MzO30uc3Qze2ZpbGw6IzIxRDRGRt9LnN0NhtmaWxsOiM2M0E3RDM7fS5zdDV7ZmlsbDojQzFFN0ZGO30uc3Q2e2ZpbGw6I0ZGRkZGRjt9LnN0N3tmaWxsOiM4M0Q2Qjk7fS5zdDh7ZmlsbDojNDZDNjhDO30uc3Q5e2ZpbGw6I0JCRThEODt9LnN0MTB7ZmlsbDojRkVCMTdEO30uc3QxMXtmaWxsOiNERDk2NjY7fS5zdDEye2ZpbGw6I0ZFRDRCNzt9LnN0MTN7ZmlsbDojRjJBMkEyO30uc3QxNhtmaWxsOiNGMUM1QzU7fS5zdDEle2ZpbGw6I0RCQjY2Mzt9LnN0MTZ7ZmlsbDojQ0VBMTUxO30uc3QxN3tmaWxsOiNFQkQ3QTk7fS5zdDE4e2ZpbGw6I0NFQ0VDRt9LnN0MTl7ZmlsbDojQjdcN0I3O30uc3QyMHTmaWxsOiNFNEU0RTQ7fS5zdDIxe2ZpbGw6IzY1QUFEMTt9LnN0MjJ7ZmlsbDojRTU3QTdBO30uc3QyM3tmaWxsOiNFNke2RTg7fS5zdDI0e2ZpbGw6I0Q2OEFENjt9LnN0MjV7ZmlsbDojRkZDQ0ZFO30uc3QyNntmaWxsOiM5OENFNUY7fS5zdDI3e2ZpbGw6IzhdQUYyQzt9LnN0Mjh7ZmlsbDojQzZFM0E3O30uc3QyOXtmaWxsOiNGRkI1Nzg7fS5zdDMwe2ZpbGw6I0VEOUY2NDt9LnN0MzF7ZmlsbDojRkZENkI1O30uc3QzMntmaWxsOiNGNEExRUY7fS5zdDMze2ZpbGw6I0REODdERDt9LnN0MzR7ZmlsbDojRjldQkY2O30uc3QzNXtmaWxsOiNB0EEYRjQ7fS5zdDM2e2ZpbGw6IzgzODVFOdt9LnN0Mzd7ZmlsbDojQ0ZDQ0Y4O30uc3QzOHTmaWxsOiNCQ0JDQkM7fS5zdDM5e2ZpbGw6I0E4QTbODt9LnN0ND7ZmlsbDojREFEFURBO30uc3Q0MXtmaWxsOiM3N0NDREI7fS5zdDQye2ZpbGw6IzREQkNDMTt9LnN0NDN7ZmlsbDojQjRFM0VCO30uc3Q0NhtmaWxsOiNGRkI3QTQ7fS5zdDQle2ZpbGw6I0Y2OUE3Qjt9LnN0NDZ7ZmlsbDojRkZEN0NEO30uc3Q0N3tmaWxsOiM3MUM4RjQ7fS5zdDQ4e2ZpbGw6IzhEQzk3Nzt9LnN0NDl7ZmlsbDojN0NBODUxO30uc3Q1MHTvcGFjaXR5OjAuNDU7ZmlsbDojRkZGRkZGO308L3N0eWxlPjxnPjxnPjxwYXRoIGNsYXNzPSJzdDE1IiBkPSJNMjEuNSwLjVoLTE3Yy0xLjEsMC0yLDAuOS0yLDJ2MjdjMCwLjEsMC45LDIsMiwyaDIzYzEuMSwLdItMC45LDItMnYtMjFMMjEuNSwLjV6Ii8+PHBhdGggY2xhc3M9InN0MTYiIGQ9Ik0yNy41LDMyaC0yM0MzLjEsMzIsMiwzMCA45LDIsMjkuNXYtMjdmIwLjEsMy4xLDA5NC41LDBoMTcuMkwzMCA4LjN2MjEuMkMzMCA45LDI4LjksMzIsMjkuNSwzMnogTTQuNSwzQzMuNywxLDMsMS43LDMsMi41djI3QzMsMzAuMywzLjcsMzEsNC41LDMxaDIzYzAuOCwLDEuNS0wLjcsMS41LDEuNVY4LjdMMjEuMywzSDQuNXoiLz48L2c+PGc+PHBhdGggY2xhc3M9InN0MTciIGQ9Ik0yMS41LDAuNXY3YzAsMC42LDAuNCwLDEsMWg3TDIxLjUsMC41eiIvPjxwYXRoIGNsYXNzPSJzdDE2IiBkPSJNMjkuNSw5aC03QzIxLjcsOSwzMSw4LjMsMjEuNSw541di03YzAtMC4yLDAuMS0wLjQsMC4zLTAuNXMwLjQsMCwLjUsMC4xbDgsOEMzMCA4LjMsMzAsOC41LDMwLDguN0MyOS45LDguOSwzOS43LDMsMjkuNSw5eiBNMjIsMS43djUuOEMyMiw3LjgsMjEuMiw4LDIyLjUsOGg1LjhmMjIsMS43eiIvPjwvZz48Zz48cmVjdCB4PSI3IiB5PSI4IiBjbGFzc20ic3Q2IiB3aWR0aD0iMTMiIGhlaWdodD0iMiIvPjwvZz48Zz48cG9seWdvbiBjbGFzc20ic3Q2IiBwb2ludHM9IjEzLjUsMjkgNywyMyAyMCwyMyAiLz48L2c+PGc+PHJlY3QgeD0iNyIgeT0iMTMiIGNsYXNzPSJzdDYiIHdpZHRoPSIyIgaWQ9Vz2h0PSIyIi8+PC9nPjxnPjxyZWNOIHg9IjciIHk9IjE4IiBjbGFzc20ic3Q2IiB3aWR0aD0iMTMiIGhlaWdodD0iMiIvPjwvZz48L2c+PC9zdmc+");
    }

```

```
3QyMHTmaWxsOiNFNEU0RTQ7fS5zdDIxe2ZpbGw6IzY1QUFEMTt9LnN0MjJ7ZmlsbDojRTU3QTdBO
30uc3QyM3tmaWxsOiNFNkE2RTg7fS5zdDI0e2ZpbGw6IOQ2OEFENjt9LnN0MjV7ZmlsbDojRkZDQ
0ZF030uc3QyNntmaWxsOiM5OENFNUY7fS5zdDI3e2ZpbGw6IzhDQUYyQzt9LnN0Mjh7ZmlsbDojQ
zZFM0E3O30uc3QyOXtmaWxsOiNGRkI1Nzg7fS5zdDMwe2ZpbGw6IOVEOUY2NDt9LnN0MzF7Zmlsb
DojRkZENkI1O30uc3QzMntmaWxsOiNGNEExRUY7fS5zdDMze2ZpbGw6IOREODdERDt9LnN0MzR7Z
mlsbDojRjldQkY2O30uc3QzNXtmaWxsOiNB0EEYRjQ7fS5zdDM2e2ZpbGw6Izg4ODVFODt9LnN0M
zd7ZmlsbDojQ0ZDQ0Y4O30uc3QzOHTmaWxsOiNCQ0JDQkM7fS5zdDM5e2ZpbGw6IOE4QTbODt9L
nN0NDB7ZmlsbDojREFEQURBO30uc3Q0MXtmaWxsOiM3N0NDREI7fS5zdDQye2ZpbGw6IzREqkNDM
Tt9LnN0NDN7ZmlsbDojQjRfM0VCO30uc3Q0NHTmaWxsOiNGRkI3QTQ7fS5zdDQ1e2ZpbGw6IOY2O
UE3Qjt9LnN0NDZ7ZmlsbDojRkZEN0NEO30uc3Q0N3tmaWxsOiM3MUM4RjQ7fS5zdDQ4e2ZpbGw6I
zhEQzk3Nzt9LnN0ND17ZmlsbDojN0NBODUxO30uc3Q1MHTvcGFjaXR5OjAuNDU7ZmlsbDojRkZGR
kZGO308L3N0eWxlPjxnPjxwYXR0IGNsYXNzPSJzdDE4IiBkPSJNMjEuNSwwLjVoLTE3Yy0xLjEsM
C0yLDAuOS0yLDJ2MjdjMCwxLjEsMC45LDIsMiwyADIzYzEuMSwwLDItMC45LDItMnYtMjFMMjEuN
SwwLjV6Ii8+PHBhdGggY2xhc3M9InN0MTkiIGQ9Ik0yNy41LDMyaC0yM0MzLjEsMzIsMiwwMC45L
DIsmjkuNXytMjdDMiwxLjEsMy4xLDA5NC41LDBoMTcuMkwzMCw4LjN2MjEuMkMzMzMCwzMC45LDI4L
jksMzIsMjcuNSwwMnogTTQuNSwwQzMuNywxLDMsMS43LDMsMi41djI3QzMsMzAuMywzLjcsMzEsN
C41LDMxaDIzYzAuOCwwLDEuNS0wLjcsMS41LTEuNVY4LjdmMjEuMywxSDQuNXoiLz48L2c+PGc+P
HBhdGggY2xhc3M9InN0MjAiIGQ9Ik0yMS41LDAuNXI3YzAsMC42LDAuNCwwLDEsMWg3TDIxLjUsM
C41eiIvPjxwYXR0IGNsYXNzPSJzdDE5IiBkPSJNMjkuNSw5aC03QzIxLjcsOSwyMSw4LjMsMjEsN
y41di03YzAtMC4yLDAuMS0wLjQsMC4zLTAuNWwzLjItMC4xLDAuNCwwLDAuNSwwLjFsOCw4QzMwL
DguMywzMCw4LjUsMzAsOC43QzI5LjksOC45LDI5LjcsOSwyOS41LDl6IE0yMiwxLjd2NS44QzIyL
DcuOCwyMi4yLDgsMjIuNSw4aDUuOEwyMiwxLjd6Ii8+PC9nPjxnPjxyZWN0IHg9IjciIHk9IjIzI
iBjbGFzc20ic3Q2IiB3aWR0aD0iMTgiIGhlaWdodD0iMiIvPjwvZz48Zz48cmVjdCB4PSI3IiB5P
SIxNyIgy2xhc3M9InN0NiIgd2lkdg9IjE4IiBoZWlnaHQ9IjIiLz48L2c+PGc+PHJlY3QgeD0iN
yIgeT0iMTEiIGNsYXNzPSJzdDYiIHdpZHRoPSIxMSIgaGVpZ2h0PSIyIi8+PC9nPjwvc3ZnPg=="
);
}
```

```
.e-filemanager .e-large-icons .e-fe-js {
background-image:
```

```
url("data:image/svg+xml;base64,PD94bWwgdmVyc2lvbj0iMS4wIiBlbmNvZGluZz0idXRmL
TgiPz48c3ZnIHZlcnNpb249IjEuMSIgaWQ9IkxheWVyXzEiIHhtbG5zPSJodHRwOi8vd3d3LnczL
m9yZy8yMDAwL3N2ZyIgeG1sbnM6eGxpbnM9Imh0dHA6Ly93d3d3Lm93d3d3Lm93d3d3Lm93d3d3L
Hg9IjBweCIgeT0iMHB4IiB2aWV3Qm94PSIwIDAgaWQ9IjEiIHhtbG5zPSJodHRwOi8vd3d3LnczL
m9yZy8yMDAwL3N2ZyIgeG1sbnM6eGxpbnM9Imh0dHA6Ly93d3d3Lm93d3d3Lm93d3d3Lm93d3d3L
Hq9Y3NzIj4uc3Qwe2ZpbGw6IOZGOTI5Mjt9LnN0MXtmaWxsOiNFODdFN0U7fS5zdDQ7ZmlsbDojR
kZDM0MzO30uc3Qze2ZpbGw6IzIxRDRGRt9LnN0NHTmaWxsOiM2M0E3RDM7fS5zdDV7ZmlsbDojQ
zFFN0ZGO30uc3Q2e2ZpbGw6IOZGRkZGRjt9LnN0N3tmaWxsOiM4M0Q2Qjk7fS5zdDh7ZmlsbDojN
DZDNjhDO30uc3Q5e2ZpbGw6IOJCRThEODt9LnN0MTB7ZmlsbDojRkVCMTdEO30uc3QxMXtmaWxsO
iNERDk2NjY7fS5zdDEye2ZpbGw6IOZFRDRCNzt9LnN0MTN7ZmlsbDojRjJBMkEyO30uc3QxNHTma
WxsOiNGMUM1QzU7fS5zdDE1e2ZpbGw6IOZRCQjY2Mzt9LnN0MTZ7ZmlsbDojQ0VBMTUxO30uc3QxN
3tmaWxsOiNFQkQ3QTk7fS5zdDE4e2ZpbGw6IOZFRDQ0VDRt9LnN0MT17ZmlsbDojQjdCN0I3O30uc
3QyMHTmaWxsOiNFNEU0RTQ7fS5zdDIxe2ZpbGw6IzY1QUFEMTt9LnN0MjJ7ZmlsbDojRTU3QTdBO
30uc3QyM3tmaWxsOiNFNkE2RTg7fS5zdDI0e2ZpbGw6IOQ2OEFENjt9LnN0MjV7ZmlsbDojRkZDQ
0ZF030uc3QyNntmaWxsOiM5OENFNUY7fS5zdDI3e2ZpbGw6IzhDQUYyQzt9LnN0Mjh7ZmlsbDojQ
zZFM0E3O30uc3QyOXtmaWxsOiNGRkI1Nzg7fS5zdDMwe2ZpbGw6IOVEOUY2NDt9LnN0MzF7Zmlsb
DojRkZENkI1O30uc3QzMntmaWxsOiNGNEExRUY7fS5zdDMze2ZpbGw6IOREODdERDt9LnN0MzR7Z
mlsbDojRjldQkY2O30uc3QzNXtmaWxsOiNB0EEYRjQ7fS5zdDM2e2ZpbGw6Izg4ODVFODt9LnN0M
zd7ZmlsbDojQ0ZDQ0Y4O30uc3QzOHTmaWxsOiNCQ0JDQkM7fS5zdDM5e2ZpbGw6IOE4QTbODt9L
nN0NDB7ZmlsbDojREFEQURBO30uc3Q0MXtmaWxsOiM3N0NDREI7fS5zdDQye2ZpbGw6IzREqkNDM
Tt9LnN0NDN7ZmlsbDojQjRfM0VCO30uc3Q0NHTmaWxsOiNGRkI3QTQ7fS5zdDQ1e2ZpbGw6IOY2O
UE3Qjt9LnN0NDZ7ZmlsbDojRkZEN0NEO30uc3Q0N3tmaWxsOiM3MUM4RjQ7fS5zdDQ4e2ZpbGw6I
zhEQzk3Nzt9LnN0ND17ZmlsbDojN0NBODUxO30uc3Q1MHTvcGFjaXR5OjAuNDU7ZmlsbDojRkZGR
kZGO308L3N0eWxlPjxnPjxwYXR0IGNsYXNzPSJzdDI2IiBkPSJNMjEuNSwwLjVoLTE3Yy0xLjEsM
C0yLDAuOS0yLDJ2MjdjMCwxLjEsMC45LDIsMiwyADIzYzEuMSwwLDItMC45LDItMnYtMjFMMjEuN
SwwLjV6Ii8+PHBhdGggY2xhc3M9InN0MjciIGQ9Ik0yNy41LDMyaC0yM0MzLjEsMzIsMiwwMC45L
DIsmjkuNXytMjdDMiwxLjEsMy4xLDA5NC41LDBoMTcuMkwzMCw4LjN2MjEuMkMzMzMCwzMC45LDI4L
DIsmjkuNXytMjdDMiwxLjEsMy4xLDA5NC41LDBoMTcuMkwzMCw4LjN2MjEuMkMzMzMCwzMC45LDI4L
```

```
jksMzIsMjcuNSwzMnogTTQuNSwxQzMuNywxLDMsMS43LDMsMi41djI3QzMsMzAuMywzLjcsMzEsN
C41LDMxaDiZyZuAuOCwwLDEuNS0wLjcsMS41LTEuNVY4LjdMMjEuMywxSDQuNXoiLz48L2c+PGc+P
HBhdGggY2xhc3M9InN0MjgiIGQ9Ik0yMS41LDAuNXy3YzAsMC42LDAuNCwwLDEsMWg3TDIxLjUsM
C41eiIvPjxwYXRoIGNsYXNzPSJzdDI3IiBkPSJNMjkuNSw5aC03QzIxLjcsOSwyMSw4LjMsMjEsN
y41di03YzAtMC4yLDAuMS0wLjQsMC4zLTAuNWMwLjItMC4xLDAuNCwwLDAuNSwwLjFsOCw4QzMwL
DguMywzMw4LjUsMzAsOC43QzI5LjksOC45LDI5LjcsOSwyOS41LDl6IE0yMiwxLjd2NS44QzIyL
DcuOCwyMi4yLDgsMjIuNSw4aDUuOEwyMiwxLjd6Ii8+PC9nPjxnPjxnPjxwYXRoIGNsYXNzPSJzd
DYiIGQ9Ik0xNS4xLDIxYzAsMC45LTAuMiwxLjYtMC42LDEuOWMtMC40LDAuNC0xLjEsMC42LTEuO
SwwLjZjLTAuNSwwLTAuOC0wLjEtMS4xLTAuMmMtMC4zLTAuMS0wLjUtMC4zLTAuNy0wLjVzLTAuM
y0wLjUtMC40LTAuOGMtMC4xLTAuMy0wLjEtMC42LTAuMS0xdI0wLjRoMS42djAuM2MwLDAuNSwwL
DAuOCwwLjIsMWMwLjEsMC4yLDAuMywwLjMsMC42LDAuM2MwLjMsMCwwLjUtMC4xLDAuNi0wLjNjM
C4xLTAuMiwwLjEtMC42LDAuMS0xdI02LjFoMS43VjIxeiIvPjxwYXRoIGNsYXNzPSJzdDYiIGQ9I
k0xNy43LDIxYzAsMC4yLDAuMS40LDAuMS41czAuMSwwLjMsMC4yLDAuNGMwLjEsMC4xLDAuMiwwL
jIsMC40LDAuM2MwLjEsMC4xLDAuMywwLjEsMC42LDAuMWMwLjMsMCwwLjUtMC4xLDAuNy0wLjNzM
C4zLTAuNCwwLjMtMC44YzAtMC4yLDAuTMC40LTAuMS0wLjVzLTAuMS0wLjMtMC4zLTAuNGMtMC4xL
TAuMS0wLjMtMC4yLTAuNS0wLjNjLTAuMi0wLjEtMC40LTAuMi0wLjctMC4zYy0wLjQtMC4xLTAuN
y0wLjMtMS0wLjRzLTAuNS0wLjMtMC43LTAuNWMtMC4yLTAuMi0wLjMtMC40LTAuNC0wLjdtMTYsM
TcuNSwxNiwxNy4yYzAtMC44LDAuMi0xLjUsMC43LTEuOXMxLjEtMC42LDEuOS0wLjZjMC40LDAuM
C43LDAuMS4xLDAuMWMwLjMsMC4xLDAuNiwwLjIsMC44LDAuNGMwLjIsMC4yLDAuNCwwLjQsMC42L
DAuN2MwLjEsMC4zLDAuMiwwLjYsMC4yLDF2MC4yaC0xLjdjMC0wLjQtMC4xLTAuNy0wLjItMC45Y
y0wLjEtMC4yLTAuNC0wLjMtMC43LTAuM2MtMC4yLDAuTMC40LDAuTMC41LDAuMVMxOCwwNi4xLDE4L
DE2LjJjLTAuMSwwLjEtMC4xLDAuMi0wLjIsMC4zYzAsMC4xLDAuMS4yLDAuMS40YzAsMC4zLDAuM
SwwLjUsMC4yLDAuN2MwLjEsMC4yLDAuNCwwLjMsMC43LDAuNWwwLjMsMC42YzAuMywwLjEsMC42L
DAuMywwLjgsMC40czAuNCwwLjMsMC41LDAuNXMwLjIsMC40LDAuMywwLjZjMCwwLjIsMC4xLDAuN
SwwLjEsMC43YzAsMC45LTAuMywxLjYtMC44LDJjLTAuNSwwLjQtMS4zLDAuNi0yLjIsMC42Yy0xL
DAtMS43LTAuMi0yLjEtMC42Yy0wLjQtMC40LTAuNi0xLTAuNi0xLjh2LTAuM2gxLjdWMjF6Ii8+P
C9nPjwvZz48L3N2Zz4=");
```

```
}
```

```
.e-filemanager .e-large-icons .e-fe-css {
background-image:
```

```
url("data:image/svg+xml;base64,PD94bWwgdmVyc2lvbj0iMS4wIiBlbmNvZGluZz0idXRmL
TgiPz48c3ZnIHZlcnNpb249IjEuMSIgaWQ9IkxheWVyXzEiIHhtbG5zPSJodHRwOi8vd3d3LnczL
m9yZy8yMDAwL3N2ZyIgeG1sbnM6eGxpbnM6Imh0dHA6Ly93d3d3LnczLm9yZy8yMDAwL3N2ZyI
Hg9IjBweCIgeT0iMHB4IiB2aWV3Qm94PSIwIDAgaWQ9IjEiIH0eWx1PSJlbmFibGUtYmFja2dyb
3VuZDpuZXMwMCAwIDMyIDMyOyIgeG1sOnNwYWNlPSJwcmVzZXJ2ZSI+PHN0eWx1IHR5cGU9InRle
HQvY3NzIj4uc3Qwe2ZpbGw6I0ZGOTI5Mj9LnN0MXTmaWxsOiNFODdFN0U7fS5zdDZjZmlsbDojR
kZDM0Mz030uc3Qze2ZpbGw6IzIxRDRGRt9LnN0NHTmaWxsOiM2M0E3RDM7fS5zdDZjZmlsbDojQ
zFFN0ZG030uc3Qze2ZpbGw6I0ZGRkZGRj9LnN0N3tmaWxsOiM4M0Q2Qjk7fS5zdDh7ZmlsbDojN
DZDNjhDO30uc3Qze2ZpbGw6I0JCRThEODt9LnN0MTB7ZmlsbDojRkVCMTdEO30uc3QxMXtmaWxsO
iNERDk2NjY7fS5zdDEye2ZpbGw6I0ZFRDRCNzt9LnN0MTN7ZmlsbDojRjJBMkEyO30uc3QxNHTma
WxsOiNGMUM1QzU7fS5zdDEle2ZpbGw6I0RCQjY2Mzt9LnN0MTZ7ZmlsbDojQ0VBMTUxO30uc3QxN
3tmaWxsOiNFQkQ3QTk7fS5zdDE4e2ZpbGw6I0NFQ0VDRTt9LnN0MTl7ZmlsbDojQjdCN0I3O30uc
3QyMHTmaWxsOiNFNEU0RTQ7fS5zdDIxe2ZpbGw6IzY1QUFEMTt9LnN0MjJ7ZmlsbDojRTU3QTdBO
30uc3QyM3tmaWxsOiNFNkE2RTg7fS5zdDI0e2ZpbGw6I0Q2OEFENjt9LnN0MjV7ZmlsbDojRkZDQ
0ZFO30uc3QyNntmaWxsOiM5OENFNUY7fS5zdDI3e2ZpbGw6IzhDQUYyQzt9LnN0Mjh7ZmlsbDojQ
zZFM0E3O30uc3QyOxtmaWxsOiNGRkI1Nzg7fS5zdDMwe2ZpbGw6I0VEOUY2NDt9LnN0MzF7Zmlsb
DojRkZENkI1O30uc3QzMntmaWxsOiNGNEExRUy7fS5zdDMze2ZpbGw6I0REODdERD9LnN0MzR7Z
mlsbDojRjldQkY2O30uc3QzNXtmaWxsOiNB0EEYRjQ7fS5zdDM2e2ZpbGw6Izg4ODVFOdt9LnN0M
zdzZmlsbDojQ0ZDQ0Y4O30uc3QzOhtmaWxsOiNCQ0JDQkM7fS5zdDM5e2ZpbGw6I0E4QThBODt9L
nN0ND7ZmlsbDojREFEQUROBO30uc3Q0MXtmaWxsOiM3N0NDREI7fS5zdDQye2ZpbGw6IzREQkNDM
Tt9LnN0NDN7ZmlsbDojQjRFM0VCO30uc3Q0NHTmaWxsOiNGRkI3QTQ7fS5zdDQle2ZpbGw6I0Y2O
UE3Qjt9LnN0NDZ7ZmlsbDojRkZEN0NEO30uc3Q0N3tmaWxsOiM3MUM4RjQ7fS5zdDQ4e2ZpbGw6I
zhEQzk3Nzt9LnN0NDl7ZmlsbDojN0NBODUxO30uc3Q1MHTvcGFjaXR5OjAuNDU7ZmlsbDojRkZGR
kZG0308L3N0eWx1PjxnPjxwYXRoIGNsYXNzPSJzdDI5IiBkPSJNMjkuNSwwLjVoLTE3Yy0xLjEsM
C0yLDAuOS0yLDJ2MjdjMCwwLjEsMC45LDI5MiwyYDIzYzEuMSwwLDI5TMC45LDI5TmYtMjFMMjEuN
SwwLjV6Ii8+PHBhdGggY2xhc3M9InN0MzAiIGQ9Ik0yNy41LDMyaC0yM0MzLjEsMzIsMiwxMC45L
```

```
DIsmJkuNXyTmjdMiwXlJEsMy4xLDAsNC41LDBoMTcuMkwzMCw4LjN2MjEuMkMzMCwzMC45LDI4L
jksMzIsMjcuNSwzMnogTTQuNSwXQzMuNywxLDMsMS43LDMsMi41djI3QzMsMzAuMywzLjcsMzEsN
C41LDMxaDIzYzAuOCwwLDEuNS0wLjcsMS41LTEuNVY4LjdMMjEuMywxSDQuNXoiLz48L2c+PGc+P
HBhdGggY2xhc3M9InN0MzEiIGQ9Ik0yMS41LDAuNXy3YzAsMC42LDAuNCwwLDEsMWG3TDIxLjUsM
C41eiIvPjxwYXRoIGNsYXNzPSJzdDMwIiBkPSJNMjkuNSw5aC03QzIxLjcsOSwyMSw4LjMsMjEsN
y41di03YzAtMC4yLDAuMS0wLjQsMC4zLTAuNWMwLjItMC4xLDAuNCwwLDAuNSwwLjFsOCw4QzMwL
DguMywzMCw4LjUsMzAsOC43QzI5LjksOC45LDI5LjcsOSwyOS41LDl6IE0yMiwxLjd2NS44QzIyL
DcuOCwyMi4yLDgsMjIuNSw4aDUuOEwyMiwxLjd6Ii8+PC9nPjxnPjxwYXRoIGNsYXNzPSJzdDYiI
GQ9Ik0xMC43LDE2LjNjLTAuMS0wLjMtMC40LTAuNC0wLjgtMC40Yy0wLjIsMC0wLjQsMC4xLTAuN
SwwLjJjLTAuMSwwLjEtMC4yLDAuMy0wLjMsMC42cy0wLjEsMC42LTAuMiwxYzAsMC40LDAsMC45L
DAsMS41YzAsMC42LDAsMS4yLDAuMSwwLjZzMC4xLDAuNywwLjIsMC45czAuMiwwLjQsMC4zLDAuN
GMwLjEsMC4xLDAuMywwLjEsMC41LDAuMWMwLjIsMCwwLjMsMCwwLjQtMC4xYzAuMS0wLjEsMC4yL
TAuMiwwLjMtMC4zYzAuMS0wLjIsMC4yLTAuNCwwLjItMC43YzAuMS0wLjMsMC4xLTAuNywwLjEtM
S4xaDEuN2MwLDAuNSwwLDAuOS0wLjEsMS4zcy0wLjIsMC44LTAuNCwwLjFjLTAuMiwwLjMtMC41L
DAuNi0wLjgsMC43Yy0wLjQsMC4yLTAuOCwwLjMtMS40LDAuM2MtMC42LDAuMS4yLTAuMS0xLjUtM
C4zYy0wLjQtMC4yLTAuNy0wLjUtMC45LTAuOXMtMC4zLTAuOS0wLjQtMS40Yy0wLjEtMC41LTAuM
S0xLjEtMC4xLTEuOGMwLTAuNiwwLTAuMiwwLjEtMS44YzAuMS0wLjUsMC4yLTAuMS40LTAuNGMwL
jItMC40LDAuNS0wLjcsMC45LTAuOWMwLjQtMC4yLDAuOS0wLjMsMS41LTAuM2MwLjYsMCwwLjEsM
C4xLDEuNSwwLjNjMC40LDAuMiwwLjYsMC41LDAuOCwwLjhzMC4zLDAuNiwwLjMsMWMwLDAuNCwwL
jEsMC43LDAuMSwxaC0xLjdMTAuOSwXNy4xLDEwLjgsMTYyNiwxMC43LDE2LjN6Ii8+PHBhdGggY
2xhc3M9InN0NiIGZD0iITTE1LjEsMjFjMCwwLjIsMCwwLjQsMCwwLjVzMC4xLDAuMywwLjIsMC40Y
zAuMSwwLjEsMC4yLDAuMiwwLjQsMC4zYzAuMSwwLjEsMC4zLDAuMSwwLjYsMC4xYzAuMywwLDAuN
S0wLjEsMC43LTAuM3MwLjMtMC40LDAuMy0wLjhjMC0wLjIsMC0wLjQtMC4xLTAuNXMtMC4xLTAuM
y0wLjMtMC40Yy0wLjEtMC4xLTAuMy0wLjItMC41LTAuM2MtMC4yLTAuMS0wLjQtMC4yLTAuNy0wL
jNjLTAuNC0wLjEtMC43LTAuMy0xLTAuNFMxNC4yLDE5LDE0LDE4LjhjLTAuMi0wLjItMC4zLTAuN
C0wLjQtMC43cy0wLjEtMC42LTAuMS0wLjljMC0wLjgsMC4yLTAuNSwwLjctMS45czEuMS0wLjYsM
S45LTAuNmMwLjQsMCwwLjcsMCwwLjEsMC4xYzAuMywwLjEsMC42LDAuMiwwLjgsMC40YzAuMiwwL
jIsMC40LDAuNCwwLjYsMC43YzAuMSwwLjMsMC4yLDAuNiwwLjIsMXyYwLjJoLTEuN2MwLTAuNC0wL
jEtMC43LTAuMi0wLjljLTAuMS0wLjItMC40LTAuMy0wLjctMC4zYy0wLjIsMC0wLjQsMC0wLjUsM
C4xcy0wLjIsMC4xLTAuMywwLjJjLTAuMSwwLjEtMC4xLDAuMi0wLjIsMC4zYzAsMC4xLDAuMS4yL
DAsMC40YzAsMC4zLDAuMSwwLjUsMC4yLDAuN2MwLjEsMC4yLDAuNCwwLjMsMC43LDAuNWwwLjMsM
C42YzAuMywwLjEsMC42LDAuMywwLjgsMC40czAuNCwwLjJsMC41LDAuNVMxOSwyMCwwOSwyMC4yY
zAsMC4yLDAuMSwwLjUsMC4xLDAuN2MwLDAuOS0wLjMsMS42LTAuOCwyYy0wLjUsMC40LTAuMywwL
jYtMi4yLDAuNmMtMSwwLTAuNy0wLjItMi4xLTAuNmMtMC40LTAuNC0wLjYtMS0wLjYtMS44di0wL
jNoMS43VjIxeiIvPjxwYXRoIGNsYXNzPSJzdDYiIGQ9Ik0yMS4zLDEixYzAsMC4yLDAsMC40LDAsM
C41czAuMSwwLjMsMC4yLDAuNGMwLjEsMC4xLDAuMiwwLjIsMC40LDAuM2MwLjEsMC4xLDAuMywwL
jEsMC42LDAuMWMwLjMsMCwwLjUtMC4xLDAuNy0wLjNzMC4zLTAuNCwwLjMtMC44YzAtMC4yLDAuT
C40LTAuMS0wLjVzLTAuMS0wLjMtMC4zLTAuNGMtMC4xLTAuMS0wLjMtMC4yLTAuNS0wLjNjLTAuM
i0wLjEtMC40LTAuMi0wLjctMC4zYy0wLjQtMC4xLTAuNy0wLjMtMS0wLjRzLTAuNS0wLjMtMC43L
TAuNWMtMC4yLTAuMi0wLjMtMC40LTAuNC0wLjdzLTAuMS0wLjYtMC4xLTAuOWMwLTAuOCwwLjItM
S41LDAuNy0xLjlzMS4xLTAuNiwwLjktMC42YzAuNCwwLDAuNywwLDEuMSwwLjFjMC4zLDAuMSwwL
jYsMC4yLDAuOCwwLjRjMC4yLDAuMiwwLjQsMC40LDAuNiwwLjdjMC4xLDAuMywwLjIsMC42LDAuM
iwxLjAuMmgtMS43YzAtMC40LTAuMS0wLjctMC4yLTAuOWMtMC4xLTAuMi0wLjQtMC4zLTAuNy0wL
jNjLTAuMiwwLTAuNCwwLTAuNSwwLjFzLTAuMiwwLjEtMC4zLDAuMmMtMC4xLDAuMS0wLjEsMC4yL
TAuMiwwLjNjMCwwLjEsMCwwLjIsMCwwLjRjMCwwLjMsMC4xLDAuNSwwLjIsMC43YzAuMSwwLjIsM
C40LDAuMywwLjcsMC41bDEuMywwLjZjMC4zLDAuMSwwLjYsMC4zLDAuOCwwLjRzMC40LDAuMywwL
jUsMC41czAuMiwwLjQsMC4zLDAuNmMwLDAuMiwwLjEsMC41LDAuMSwwLjdjMCwwLjktMC4zLDEuN
i0wLjgsMmMtMC41LDAuNC0xLjMsMC42LTIuMiwwLjZjLTAuMS0wLjctMC4yLTIuMS0wLjZjLTAuN
C0wLjQtMC42LTAuTETMC42LTAuOHYtMC4zaDEuN1YyMXoiLz48L2c+PC9zdmc+");
}

.e-filemanager .e-large-icons .e-fe-html {
background-image:
url("data:image/svg+xml;base64,PD94bWwgdmVyc2lvbj0iMS4wIiBlbmNvZGluZz0idXRmL
TgiPz48c3ZnIHZlcnNpb249IjEuMSIgaWQ9IkxheWVyXzEiIHhtbG5zPSJodHRwOi8vd3d3LnczL
m9yZy8yMDAwL3N2ZyIgeG1sbnM6Gxpbms9Imh0dHA6Ly93d3cudzMub3JnLzE5OTkveGxpbmsiI
Hg9IjBweCIgeT0iMHB4IiB2aWV3Qm94PSIwIDAgMzIiIHNoeWx1PSJlbmFibGutYmFja2dyb
```

```
3VuZDpuZXcgMCAwIDMyIDMyOyIgeG1sOnNwYWNlPSJwcmVzZXJ2ZSI+PHN0eWxlIHR5cGU9InRle
HQvY3NzIj4uc3Qwe2ZpbGw6IOZGOTI5Mjt9LnN0MXtmaWxsOiNFODdFN0U7fS5zdDJ7ZmlsbDojR
kZDM0MzO30uc3Qze2ZpbGw6IzkrRDRGRtT9LnN0NHTmaWxsOiM2M0E3RDM7fS5zdDV7ZmlsbDojQ
zFFN0ZGO30uc3Q2e2ZpbGw6IOZGRkZGRjt9LnN0N3tmaWxsOiM4M0Q2Qjk7fS5zdDh7ZmlsbDojN
DZDNjhDO30uc3Q5e2ZpbGw6IOJCRThEODt9LnN0MTB7ZmlsbDojRkVCMTdEO30uc3QxMXtmaWxsO
iNERDk2NjY7fS5zdDEye2ZpbGw6IOZFRDRCNzt9LnN0MTN7ZmlsbDojRjJBMkEyO30uc3QxNHTma
WxsOiNGMUM1QzU7fS5zdDEle2ZpbGw6IORCQjY2Mzt9LnN0MTZ7ZmlsbDojQ0VBMTUxO30uc3QxN
3tmaWxsOiNFQkQ3QTk7fS5zdDE4e2ZpbGw6IOVFQ0VDRTt9LnN0MTl7ZmlsbDojQjdCN0I3O30uc
3QyMHTmaWxsOiNFNEU0RTQ7fS5zdDIxe2ZpbGw6IzY1QUFEMTt9LnN0MjJ7ZmlsbDojRTU3QTdBO
30uc3QyM3tmaWxsOiNFNkE2RTg7fS5zdDI0e2ZpbGw6IOQ2OEFENjt9LnN0MjV7ZmlsbDojRkZDQ
0ZFO30uc3QyNntmaWxsOiM5OENFNUY7fS5zdDI3e2ZpbGw6IzhDQUYyQzt9LnN0Mjh7ZmlsbDojQ
zZFM0E3O30uc3QyOXtmaWxsOiNGRkI1Nzg7fS5zdDMwe2ZpbGw6IOVEOUY2NDt9LnN0MzF7Zmlsb
DojRkZENkI1O30uc3QzMntmaWxsOiNGNEExRUY7fS5zdDMze2ZpbGw6IOREODdERTt9LnN0MzR7Z
mlsbDojRjldQkY2O30uc3QzNXtmaWxsOiNBOEEYRjQ7fS5zdDM2e2ZpbGw6IzgzODVFOdt9LnN0M
zd7ZmlsbDojQ0ZDQ0Y4O30uc3QzOHTmaWxsOiNCQ0JDQkM7fS5zdDM5e2ZpbGw6IOE4QThBODt9L
nN0NDB7ZmlsbDojREFEQURBO30uc3Q0MXtmaWxsOiM3N0NDREI7fS5zdDQye2ZpbGw6IzREQkNDM
Tt9LnN0NDN7ZmlsbDojQjRfM0VCO30uc3Q0NHTmaWxsOiNGRkI3QTQ7fS5zdDQle2ZpbGw6IOY2O
UE3Qjt9LnN0NDZ7ZmlsbDojRkZEN0NEO30uc3Q0N3tmaWxsOiM3MUM4RjQ7fS5zdDQ4e2ZpbGw6I
zhEQzk3Nzt9LnN0NDl7ZmlsbDojN0NBODUxO30uc3Q1MHTvcGFjaXR5OjAuNDU7ZmlsbDojRkZGR
kZGO308L3N0eWxlPjxnPjxwYXR0IGNsYXNzPSJzdDMYIiBkPSJNMjEuNSwwLjVoLTE3Yy0xLjEsM
C0yLDAuOS0yLDJ2MjdjMCwLjEsMC45LDIsMiwyADIzYzEuMSwwLDItMC45LDItMnYtMjFMMjEuN
SwwLjV6Ii8+PHBhdGggY2xhc3M9InN0MzMiIGQ9Ik0yNy41LDMyaC0yM0MzLjEsMzIsMiwwLjEsM
C45LDIsMjkuNXItMjdmIiwLjEsMy4xLDAuSNC4LDBoMTcuMkwzMCw4LjN2MjEuMkMzMCwzMC45LDI4L
jksMzIsMjcuNSwzMnogTTQuNSwzQzMuNywxLDMsMS43LDMsMi41djI3QzMsMzAuMywzLjcsMzEsN
C41LDMxaDIzYzAuOCwwLDEuNS0wLjcsMS41LTEuNVY4LjdmMjEuMywxSDQuNXoiLz48L2c+PGc+P
HBhdGggY2xhc3M9InN0MzQiIGQ9Ik0yMS41LDAuNXI3YzAsMC42LDAuNCwwLDEsMWG3TDIxLjUsM
C41eiIvPjxwYXR0IGNsYXNzPSJzdDMzIiBkPSJNMjkuNSw5aC03QzIxLjcsOSwyMSw4LjMsMjEsN
y41di03YzAtMC4yLDAuMS0wLjQsMC4zLTAuNWwLjItMC4xLDAuNCwwLDAuNSwwLjFsOCw4QzMwL
DguMywzMCw4LjUsMzAsOC43QzI5LjksOC45LDI5LjcsOSwyOS41LDl6IE0yMiwxLjd2NS44QzIyL
DcuOCwyMi4yLdGsmjIuNSw4aDUuOEwyMiwxLjd6Ii8+PC9nPjxnPjxwYXR0IGNsYXNzPSJzdDYiI
GQ9Ik0yMC4xLDI0LjNsLTEuMy0xLjVsNS4xLTQuMmwtNS4xLTQuMmwxLjMtMS41bDYsNWwLjIsM
C4yLDAuNCwwLjUsMC40LDAuOHMTMC4xLDAuNi0wLjQsMC44TDIwLjEsMjQuM3oiLz48L2c+PGc+P
HBhdGggY2xhc3M9InN0NiIgZD0iTTEwLjgsMjQuM2wtNi01Yy0wLjItMC4yLTAuNC0wLjUtMC40L
TAuOHMwLjEtMC42LDAuNC0wLjhsNi01bDEuMywxLjVMNywxOC41bDUuMSw0LjJMMTAuOCwyNC4ze
iIvPjwvZz48Zz48cmVjdCB4PSI3LjMiIHk9IjE3LjUiIHRyYW5zM9ybT0ibWF0cm14KDAuMjQyM
iAtMC45NzAyIDAuOTcwMiAwLjIOMjIgLTUyMjAzMiAyOS4wNTgpIiBjbGFzc29ic3Q2IiB3aWR0a
D0iMTYyNSIgaGVpZ2h0PSIyIi8+PC9nPjwvc3ZnPg==");
}

.e-filemanager .e-large-icons .e-fe-unknown {
background-image:
url("data:image/svg+xml;base64,PD94bWwgdmlkZS50eWxlIHR5cGU9InRle
HQvY3NzIj4uc3Qwe2ZpbGw6IOZGOTI5Mjt9LnN0MXtmaWxsOiNFODdFN0U7fS5zdDJ7ZmlsbDojR
kZDM0MzO30uc3Qze2ZpbGw6IzkrRDRGRtT9LnN0NHTmaWxsOiM2M0E3RDM7fS5zdDV7ZmlsbDojQ
zFFN0ZGO30uc3Q2e2ZpbGw6IOZGRkZGRjt9LnN0N3tmaWxsOiM4M0Q2Qjk7fS5zdDh7ZmlsbDojN
DZDNjhDO30uc3Q5e2ZpbGw6IOJCRThEODt9LnN0MTB7ZmlsbDojRkVCMTdEO30uc3QxMXtmaWxsO
iNERDk2NjY7fS5zdDEye2ZpbGw6IOZFRDRCNzt9LnN0MTN7ZmlsbDojRjJBMkEyO30uc3QxNHTma
WxsOiNGMUM1QzU7fS5zdDEle2ZpbGw6IORCQjY2Mzt9LnN0MTZ7ZmlsbDojQ0VBMTUxO30uc3QxN
3tmaWxsOiNFQkQ3QTk7fS5zdDE4e2ZpbGw6IOVFQ0VDRTt9LnN0MTl7ZmlsbDojQjdCN0I3O30uc
3QyMHTmaWxsOiNFNEU0RTQ7fS5zdDIxe2ZpbGw6IzY1QUFEMTt9LnN0MjJ7ZmlsbDojRTU3QTdBO
30uc3QyM3tmaWxsOiNFNkE2RTg7fS5zdDI0e2ZpbGw6IOQ2OEFENjt9LnN0MjV7ZmlsbDojRkZDQ
0ZFO30uc3QyNntmaWxsOiM5OENFNUY7fS5zdDI3e2ZpbGw6IzhDQUYyQzt9LnN0Mjh7ZmlsbDojQ
zZFM0E3O30uc3QyOXtmaWxsOiNGRkI1Nzg7fS5zdDMwe2ZpbGw6IOVEOUY2NDt9LnN0MzF7Zmlsb
DojRkZENkI1O30uc3QzMntmaWxsOiNGNEExRUY7fS5zdDMze2ZpbGw6IOREODdERTt9LnN0MzR7Z
mlsbDojRjldQkY2O30uc3QzNXtmaWxsOiNBOEEYRjQ7fS5zdDM2e2ZpbGw6IzgzODVFOdt9LnN0M
zd7ZmlsbDojQ0ZDQ0Y4O30uc3QzOHTmaWxsOiNCQ0JDQkM7fS5zdDM5e2ZpbGw6IOE4QThBODt9L
nN0NDB7ZmlsbDojREFEQURBO30uc3Q0MXtmaWxsOiM3N0NDREI7fS5zdDQye2ZpbGw6IzREQkNDM
Tt9LnN0NDN7ZmlsbDojQjRfM0VCO30uc3Q0NHTmaWxsOiNGRkI3QTQ7fS5zdDQle2ZpbGw6IOY2O
UE3Qjt9LnN0NDZ7ZmlsbDojRkZEN0NEO30uc3Q0N3tmaWxsOiM3MUM4RjQ7fS5zdDQ4e2ZpbGw6I
zhEQzk3Nzt9LnN0NDl7ZmlsbDojN0NBODUxO30uc3Q1MHTvcGFjaXR5OjAuNDU7ZmlsbDojRkZGR
kZGO308L3N0eWxlPjxnPjxwYXR0IGNsYXNzPSJzdDMYIiBkPSJNMjEuNSwwLjVoLTE3Yy0xLjEsM
C0yLDAuOS0yLDJ2MjdjMCwLjEsMC45LDIsMiwyADIzYzEuMSwwLDItMC45LDItMnYtMjFMMjEuN
SwwLjV6Ii8+PHBhdGggY2xhc3M9InN0MzMiIGQ9Ik0yNy41LDMyaC0yM0MzLjEsMzIsMiwwLjEsM
C45LDIsMjkuNXItMjdmIiwLjEsMy4xLDAuSNC4LDBoMTcuMkwzMCw4LjN2MjEuMkMzMCwzMC45LDI4L
jksMzIsMjcuNSwzMnogTTQuNSwzQzMuNywxLDMsMS43LDMsMi41djI3QzMsMzAuMywzLjcsMzEsN
C41LDMxaDIzYzAuOCwwLDEuNS0wLjcsMS41LTEuNVY4LjdmMjEuMywxSDQuNXoiLz48L2c+PGc+P
HBhdGggY2xhc3M9InN0MzQiIGQ9Ik0yMS41LDAuNXI3YzAsMC42LDAuNCwwLDEsMWG3TDIxLjUsM
C41eiIvPjxwYXR0IGNsYXNzPSJzdDMzIiBkPSJNMjkuNSw5aC03QzIxLjcsOSwyMSw4LjMsMjEsN
y41di03YzAtMC4yLDAuMS0wLjQsMC4zLTAuNWwLjItMC4xLDAuNCwwLDAuNSwwLjFsOCw4QzMwL
DguMywzMCw4LjUsMzAsOC43QzI5LjksOC45LDI5LjcsOSwyOS41LDl6IE0yMiwxLjd2NS44QzIyL
DcuOCwyMi4yLdGsmjIuNSw4aDUuOEwyMiwxLjd6Ii8+PC9nPjxnPjxwYXR0IGNsYXNzPSJzdDYiI
GQ9Ik0yMC4xLDI0LjNsLTEuMy0xLjVsNS4xLTQuMmwtNS4xLTQuMmwxLjMtMS41bDYsNWwLjIsM
C4yLDAuNCwwLjUsMC40LDAuOHMTMC4xLDAuNi0wLjQsMC44TDIwLjEsMjQuM3oiLz48L2c+PGc+P
HBhdGggY2xhc3M9InN0NiIgZD0iTTEwLjgsMjQuM2wtNi01Yy0wLjItMC4yLTAuNC0wLjUtMC40L
TAuOHMwLjEtMC42LDAuNC0wLjhsNi01bDEuMywxLjVMNywxOC41bDUuMSw0LjJMMTAuOCwyNC4ze
iIvPjwvZz48Zz48cmVjdCB4PSI3LjMiIHk9IjE3LjUiIHRyYW5zM9ybT0ibWF0cm14KDAuMjQyM
iAtMC45NzAyIDAuOTcwMiAwLjIOMjIgLTUyMjAzMiAyOS4wNTgpIiBjbGFzc29ic3Q2IiB3aWR0a
D0iMTYyNSIgaGVpZ2h0PSIyIi8+PC9nPjwvc3ZnPg==");
}
```



```
DojRkZENkI1O30uc3QzMntmaWxsOiNGNEExRUY7fS5zdDMze2ZpbGw6I0REODdERDt9LnN0MzR7Z
mlsbDojRjLDQkY2O30uc3QzNXtmaWxsOiNB0EEYRjQ7fS5zdDM2e2ZpbGw6Izg4ODVFODt9LnN0M
zd7ZmlsbDojQ0ZDQ0Y4O30uc3QzOHTmaWxsOiNCQ0JDQkM7fS5zdDM5e2ZpbGw6I0E4QThBODt9L
nN0NDB7ZmlsbDojREFEFURBO30uc3Q0MXtmaWxsOiM3N0NDREI7fS5zdDQye2ZpbGw6IzREqkNDM
Tt9LnN0NDN7ZmlsbDojQjRFM0VCO30uc3Q0NHTmaWxsOiNGRkI3QTQ7fS5zdDQle2ZpbGw6I0Y2O
UE3Qjt9LnN0NDZ7ZmlsbDojRkZEN0NEO30uc3Q0N3tmaWxsOiM3MUM4RjQ7fS5zdDQ4e2ZpbGw6I
zhEQzk3Nzt9LnN0NDl7ZmlsbDojN0NBODUxO30uc3Q1MHTvcGFjaXR5OjAuNDU7ZmlsbDojRkZGR
kZGO308L3N0eWxlPjxnPjxwYXRoIGNsYXNzPSJzdDQ0IiBkPSJNMjEuNSwwLjVoLTE3Yy0xLjEsM
C0yLDAuOS0yLDJ2MjdjMCwLjEsMC45LDIsmiwyADIzYzEuMSwwLDItMC45LDItMnYtMjFMMjEuN
SwwLjV6Ii8+PHBhdGggY2xhc3M9InN0NDUiIGQ9Ik0yNy41LDMyaC0yM0MzLjEsMzIsMiwzMC45L
DIsmjkuNXytMjdDMiwxLjEsMy4xLDAsNC41LDBoMTcuMkwzMCw4LjN2MjEuMkMzMzMCwzMC45LDI4L
jksMzIsMjcuNSwzMnogTTQuNSwzQzMuNywxLDMsMS43LDMsMi41djI3QzMzMzAuMywzLjcsMzEsN
C41LDMxaDIzYzAuOCwwLDEuNS0wLjcsMS41LTEuNVY4LjdMMjEuMywxSDQuNXoiLz48L2c+PGc+P
HBhdGggY2xhc3M9InN0NDYiIGQ9Ik0yMS41LDAuNXI3YzAsMC42LDAuNCwwLDEsMWg3TDIxLjUsM
C4leiIvPjxwYXRoIGNsYXNzPSJzdDQ1IiBkPSJNMjkuNSw5aC03QzIxLjcsOSwyMSw4LjMsMjEsN
y41di03YzAtMC4yLDAuMS0wLjQsMC4zLTAuNWwLjItMC4xLDAuNCwwLDAuNSwwLjFsOCw4QzMwL
DguMywzMCw4LjUsMzAsOC43QzI5LjksOC45LDI5LjcsOSwyOS41LDl6IE0yMiwxLjd2NS44QzIyL
DcuOCwyMi4yLDgsMjIuNSw4aDUuOEwyMiwxLjd6Ii8+PC9nPjxnPjxnPjxw2x5Z29uIGNsYXNzP
SJzdDYiIHbvaW50cz0iMTcuNSwyMiAxNC41LDIyIDE0LDEwIDE4LDEwIClPjwvZz48Zz48Y2lyY
2xlIGNsYXNzPSJzdDYiIGN4PSIxNiIgY3k9IjI2IiByPSIyIi8+PC9nPjwvZz48L3N2Zz4=");
}
```

```
.e-filemanager.e-large-icons.e-fe-exe {
    background-image:
```

```
url("data:image/svg+xml;base64,PD94bWwgdmlkZ2lvdj0iMS4wIiBlbmNvZGluZz0idXRmL
TgiPz48c3ZnIHZlcnNpb249IjEuMSIgaWQ9IkxheWVyXzEiIHhtbG5zPSJodHRwOi8vd3d3LnczL
m9yZy8yMDAwL3N2ZyIgeG1sbnM6eGxpbnM9Imh0dHA6Ly93d3d3LnczLm9yZy8yMDAwL3N2ZyI
Hg9IjBweCIgeT0iMHB4IiB2aWV3Qm94PSIwIDAgaWQ9IjEiIH0eWxlPSJlbmFibGUtYmFja2dyb
3VuZDpuZXRxcGMAAwIDMyIDMyOyIgeG1sOnNwYWNlPSJwcmVzZXJ2ZSI+PHN0eWxlIHR5cGU9InRl
eHQvY3NzIj4uc3Qwe2ZpbGw6I0ZGOTI5Mj9LnN0MXtmaWxsOiNFODdFN0U7fS5zdDQ7ZmlsbDojR
kZDM0MzO30uc3Qze2ZpbGw6IzksRDRGRt9LnN0NHTmaWxsOiM2M0E3RDM7fS5zdDQ7ZmlsbDojQ
zFFN0ZGO30uc3Qze2ZpbGw6I0ZGRkZGRj9LnN0N3tmaWxsOiM4M0Q2Qjk7fS5zdDh7ZmlsbDojN
DZDNjhDO30uc3Q5e2ZpbGw6I0JCRThEODt9LnN0MTB7ZmlsbDojRkVCMTdEO30uc3QxMXtmaWxsO
iNERDk2NjY7fS5zdDEye2ZpbGw6I0ZFRDRCNzt9LnN0MTN7ZmlsbDojRjJBMkEyO30uc3QxNHTma
WxsOiNGMUM1QzU7fS5zdDEle2ZpbGw6I0RCQjY2Mzt9LnN0MTZ7ZmlsbDojQ0VBMTUxO30uc3QxN
3tmaWxsOiNFQkQ3QTk7fS5zdDE4e2ZpbGw6I0NFQ0VDRTt9LnN0MTl7ZmlsbDojQjdCN0I3O30uc
3QyMHTmaWxsOiNFNEU0RTQ7fS5zdDIxe2ZpbGw6IzY1QUFEMTt9LnN0MjJ7ZmlsbDojRTU3QTdBO
30uc3QyM3tmaWxsOiNFNkE2RTg7fS5zdDI0e2ZpbGw6I0Q2OEFENjt9LnN0MjV7ZmlsbDojRkZDQ
0ZFO30uc3QyNntmaWxsOiM5OENFNUY7fS5zdDI3e2ZpbGw6IzhDQUYyQzt9LnN0Mjh7ZmlsbDojQ
zZFM0E3O30uc3QyOXtmaWxsOiNGRkI1Nzg7fS5zdDMwe2ZpbGw6I0VEOUY2NDt9LnN0MzF7Zmlsb
DojRkZENkI1O30uc3QzMntmaWxsOiNGNEExRUY7fS5zdDMze2ZpbGw6I0REODdERDt9LnN0MzR7Z
mlsbDojRjLDQkY2O30uc3QzNXtmaWxsOiNB0EEYRjQ7fS5zdDM2e2ZpbGw6Izg4ODVFODt9LnN0M
zd7ZmlsbDojQ0ZDQ0Y4O30uc3QzOHTmaWxsOiNCQ0JDQkM7fS5zdDM5e2ZpbGw6I0E4QThBODt9L
nN0NDB7ZmlsbDojREFEFURBO30uc3Q0MXtmaWxsOiM3N0NDREI7fS5zdDQye2ZpbGw6IzREqkNDM
Tt9LnN0NDN7ZmlsbDojQjRFM0VCO30uc3Q0NHTmaWxsOiNGRkI3QTQ7fS5zdDQle2ZpbGw6I0Y2O
UE3Qjt9LnN0NDZ7ZmlsbDojRkZEN0NEO30uc3Q0N3tmaWxsOiM3MUM4RjQ7fS5zdDQ4e2ZpbGw6I
zhEQzk3Nzt9LnN0NDl7ZmlsbDojN0NBODUxO30uc3Q1MHTvcGFjaXR5OjAuNDU7ZmlsbDojRkZGR
kZGO308L3N0eWxlPjxnPjxwYXRoIGNsYXNzPSJzdDQ0IiBkPSJNMjEuNSwwLjVoLTE3Yy0xLjEsM
C0yLDAuOS0yLDJ2MjdjMCwLjEsMC45LDIsmiwyADIzYzEuMSwwLDItMC45LDItMnYtMjFMMjEuN
SwwLjV6Ii8+PHBhdGggY2xhc3M9InN0NDUiIGQ9Ik0yNy41LDMyaC0yM0MzLjEsMzIsMiwzMC45L
DIsmjkuNXytMjdDMiwxLjEsMy4xLDAsNC41LDBoMTcuMkwzMCw4LjN2MjEuMkMzMzMCwzMC45LDI4L
jksMzIsMjcuNSwzMnogTTQuNSwzQzMuNywxLDMsMS43LDMsMi41djI3QzMzMzAuMywzLjcsMzEsN
C41LDMxaDIzYzAuOCwwLDEuNS0wLjcsMS41LTEuNVY4LjdMMjEuMywxSDQuNXoiLz48L2c+PGc+P
HBhdGggY2xhc3M9InN0NDYiIGQ9Ik0yMS41LDAuNXI3YzAsMC42LDAuNCwwLDEsMWg3TDIxLjUsM
C4leiIvPjxwYXRoIGNsYXNzPSJzdDM5IiBkPSJNMjkuNSw5aC03QzIxLjcsOSwyMSw4LjMsMjEsN
y41di03YzAtMC4yLDAuMS0wLjQsMC4zLTAuNWwLjItMC4xLDAuNCwwLDAuNSwwLjFsOCw4QzMwL
DguMywzMCw4LjUsMzAsOC43QzI5LjksOC45LDI5LjcsOSwyOS41LDl6IE0yMiwxLjd2NS44QzIyL
DcuOCwyMi4yLDgsMjIuNSw4aDUuOEwyMiwxLjd6Ii8+PC9nPjxnPjxnPjxw2x5Z29uIGNsYXNzP
SJzdDYiIHbvaW50cz0iMTcuNSwyMiAxNC41LDIyIDE0LDEwIDE4LDEwIClPjwvZz48Zz48Y2lyY
2xlIGNsYXNzPSJzdDYiIGN4PSIxNiIgY3k9IjI2IiByPSIyIi8+PC9nPjwvZz48L3N2Zz4=");
```

```
DcuOCwyMi4yLDgsMjIuNSw4aDUuOEwyMiwxLjd6Ii8+PC9nPjxnPjxwYXRoIGNsYXNzPSJzdDYiI
GQ9Ik0xMS44LDE2LjJlOC44djJoMi45djEuNEg4LjhWMjJoMy4ydjEuNEg3di04LjZoNC44VjE2L
jJ6Ii8+PHBhdGggY2xhc3M9InN0NiIgZD0iTTE0LjQsMTQuOGwxLjEsMi44bDEuMS0yLjhoMS45b
C0yLDQuMmwyLjEsNC4zaC0xLjlsLTEuMi0yLjlsLTEuMiwyLjloLTEuOWwyLjEtNC4zbC0yLTQuM
kgxNC40eiIvPjxwYXRoIGNsYXNzPSJzdDYiIGQ9Ik0yNCwxNi4ySDIxdjJoMi45djEuNEgyMVYyM
mgzLjJ2MS40aC00Lj12LTguNkgYNFYxNi4yeiIvPjwvZz48L3N2Zz4=") ;
}

.e-filemanager .e-large-icons .e-fe-msi {
  background-image:
url("data:image/svg+xml;base64,PD94bWwgdmVyc2lvbj0iMS4wIiBlbmNvZGluc2ludXRM
TgiPz48c3ZnIHZlcnNpb249IjEuMSIgaWQ9IkkxheWVYXZeiIHhtbG5zPSJodHRwOi8vd3d3LnczL
m9yZy8yMDAwL3N2ZyIgeG1sbnM6Gxpbsms9Imh0dHA6Ly93d3cudzMub3JnLzE5OTkveGxpbsiI
Hg9IjBweCIgeT0iMHB4IiB2aWV3Qm94PSIwIDAgMzIgMzIiIHNoeWxlPSJlbmFibGUTYmFja2dyb
3VuZDpuZXMCAwIDMyIDMyOyIgeG1sOnNwYWNlPSJwcmVzZXJ2ZSI+PHN0eWxlIHR5cGU9InRle
HQvY3NzIj4uc3Qwe2ZpbGw6I0ZGOTI5Mjt9LnN0MXtmaWxsOiNFODdFN0U7fS5zdDj7ZmlsbDojR
kZDM0Mz030uc3Qze2ZpbGw6IzkkxRDRGRt9LnN0NHTmaWxsOiM2M0E3RDM7fS5zdDV7ZmlsbDojQ
zFFN0ZGO30uc3Q2e2ZpbGw6I0ZGRkZGRjt9LnN0N3tmaWxsOiM4M0Q2Qjk7fS5zdDh7ZmlsbDojN
DZDNjhDO30uc3Q5e2ZpbGw6I0JCRThEODt9LnN0MTB7ZmlsbDojRkVCMTdEO30uc3QxMXtmaWxsO
iNERDk2NjY7fS5zdDEye2ZpbGw6I0ZFRDRCNzt9LnN0MTN7ZmlsbDojRjJBMkEyO30uc3QxNHTma
WxsOiNGMUM1QzU7fS5zdDEle2ZpbGw6I0RCQjY2Mzt9LnN0MTZ7ZmlsbDojQ0VBMTUxO30uc3QxN
3tmaWxsOiNFQkQ3QTk7fS5zdDE4e2ZpbGw6I0NFQ0VDRt9LnN0MT17ZmlsbDojQjdCN0I3O30uc
3QyMHTmaWxsOiNFNEU0RTQ7fS5zdDIxe2ZpbGw6IzY1QUFEMTt9LnN0MjJ7ZmlsbDojRTU3QTdBO
30uc3QyM3tmaWxsOiNFNkE2RTg7fS5zdDI0e2ZpbGw6I0Q2OEFENjt9LnN0MjV7ZmlsbDojRkZDQ
0ZFO30uc3QyNntmaWxsOiM5OENFNUY7fS5zdDI3e2ZpbGw6IzhDUYyQzt9LnN0Mjh7ZmlsbDojQ
zZFM0E3O30uc3QyOXtmaWxsOiNGRkI1Nzg7fS5zdDMwe2ZpbGw6I0VEOUY2NDt9LnN0MzF7Zmlsb
DojRkZENkI1O30uc3QzMntmaWxsOiNGNEExRUY7fS5zdDMze2ZpbGw6I0REODdERDt9LnN0MzR7Z
mlsbDojRj1DQkY2O30uc3QzNXtmaWxsOiNBOEEYRjQ7fS5zdDM2e2ZpbGw6Izg4ODVFODt9LnN0M
zd7ZmlsbDojQ0ZDQ0Y4O30uc3QzOHTmaWxsOiNCQ0JDQkM7fS5zdDM5e2ZpbGw6I0E4QThBODt9L
nN0NDB7ZmlsbDojREFEQURBO30uc3Q0MXtmaWxsOiM3N0NDREI7fS5zdDQye2ZpbGw6IzREQkNDM
Tt9LnN0NDN7ZmlsbDojQjRFM0VCO30uc3Q0NHTmaWxsOiNGRkI3QTQ7fS5zdDQle2ZpbGw6I0Y2O
UE3Qjt9LnN0NDZ7ZmlsbDojRkZEN0NEO30uc3Q0N3tmaWxsOiM3MUM4RjQ7fS5zdDQ4e2ZpbGw6I
zhEQzk3Nzt9LnN0NDl7ZmlsbDojN0NBODUxO30uc3Q1MHTvcGFjaXR5OjAuNDU7ZmlsbDojRkZGR
kZGO308L3N0eWxlPjxnPjxwYXRoIGNsYXNzPSJzdDQxIiBkPSJNMjEuNSwwLjVoLTE3Yy0xLjEsM
C0yLDAuOS0yLDJ2MjdjMCwxLjEsMC45LDI5MiwyADIzYzEuMSwwLDItMC45LDI5MnYtMjFMMjEuN
SwwLjV6Ii8+PHBhdGggY2xhc3M9InN0NDIiIGQ9Ik0yNy41LDMyaC0yM0MzLjEsMzIsMiwwLjE5L
DI5MjkuNXItMjdmIiwLjEsMy4xLDA5NC41LDBoMTcuMkwzMCAwLjN2MjEuMkMzMCAwLjE5LjE4L
jksMzIsMjkuNSwzMnogTTQuNSwzQzMuNywxLDMsMS43LDMsMi41djI3QzMzMzAuMywzLjcsMzEsN
C41LDMxaDIzYzAuOCwwLDEuNS0wLjcsMS41LDEuNVY4LjdmMjEuMywxSDQuNXoiLz48L2c+PGc+P
HBhdGggY2xhc3M9InN0NDMiIGQ9Ik0yMS41LDAuNXI3YzAsMC42LDAuNCwwLDEsMWg3TDIxLjUsM
C41eiIvPjxwYXRoIGNsYXNzPSJzdDQyIiBkPSJNMjkuNSw5aC03QzIxLjcsOSwyMSw4LjMsMjEsN
y41di03YzAtMC4yLDAuMS0wLjQsMC4zLTAuNWMwLjItMC4xLDAuNCwwLDAuNSwwLjFsOCw4QzMwL
DguMywzMCAwLjUsMzAsOC43QzI5LjksOC45LDI5LjcsOSwyOS41LDl6IE0yMiwxLjd2NS44QzIyL
DcuOCwyMi4yLDgsMjIuNSw4aDUuOEwyMiwxLjd6Ii8+PC9nPjxnPjxwYXRoIGNsYXNzPSJzdDYiI
GQ9Ik05LjYsMTQuOGwxLjIsNmgbwDEuMi02aDIuNnY4LjZoLTEuNnYtNi44aDBsLTEuNSw2LjhoL
TEuM2wtMS41LTYuOGgdjYuOEg3LjF2LTguNkg5LjZ6Ii8+PHBhdGggY2xhc3M9InN0NiIgZD0iT
TE3LjMsMjFjMCwwLjIsMCwwLjQsMCwwLjVzMCA4LDAuMywwLjIsMC40YzAuMSwwLjEsMC4yLDAuM
iwwLjQsMC4zYzAuMSwwLjEsMC4zLDAuMSwwLjYsMC4xYzAuMywwLDAuNS0wLjEsMC43LTAuM3MwL
jMtMC40LDAuMy0wLjhjMC0wLjIsMC0wLjQtMC4xLTAuNXMtMC4xLTAuMy0wLjMtMC40Yy0wLjEtM
C4xLTAuMy0wLjItMC41LTAuM2MtMC4yLTAuMS0wLjQtMC4yLTAuNy0wLjNjLTAuNC0wLjEtMC43L
TAuMy0xLTAuNHMtMC41LTAuMy0wLjctMC41Yy0wLjItMC4yLTAuMy0wLjQtMC40LTAuN3MtMC4xL
TAuNi0wLjEtMC45YzAtMC44LDAuMi0xLjUsMC43LTAuOXMxLjEtMC42LDEuOS0wLjZjMC40LDA5M
C43LDA5MS4xLDAuMWMwLjMsMC4xLDAuNiwwLjIsMC44LDAuNGMwLjIsMC4yLDAuNCwwLjQsMC42L
DAuN2MwLjEsMC4zLDAuMiwwLjYsMC4yLDF2MCA4yaC0xLjdjMC0wLjQtMC4xLTAuNy0wLjItMC45Y
y0wLjEtMC4yLTAuNC0wLjMtMC43LTAuM2MtMC4yLDA5TMC40LDA5TMC41LDAuMXMtMC4yLDAuMS0wL
jMsMC4yYy0wLjEsMC4xLTAuMSwwLjItMC4yLDAuM2MwLDAuMSwwLDAuMiwwLDAuNGMwLDAuMywwL
jEsMC41LDAuMiwwLjdjMC4xLDAuMiwwLjQsMC4zLDAuNywwLjVsMS4zLDAuNmMwLjMsMC4xLDAuN
```

```
iwwLjMsMC44LDAuNHmWlJQsMC4zLDAuNSwwLjVzMC4yLDAuNCwwLjMsMC42YzAsMC4yLDAuMSwwL
jUsMC4xLDAuN2MwLDAuOS0wLjMsMS42LTAuOCwyYy0wLjUsMC40LTEuMywwLjYtMi4yLDAuNmMtM
SwwLTEuNy0wLjItMi4xLTAuNmMtMC40LTAuNC0wLjYtMS0wLjYtMS44di0wLjNoMS43VjIxeiIvP
jxwYXRoIGNsYXNzPSJzdDYiIGQ9Ik0yMi4yLDE0LjhIMjR2OC42aC0xLjdWMTQuOHoiLz48L2c+P
C9zdmc+");
}

.e-filemanager .e-large-icons .e-fe-php {
    background-image:
url("data:image/svg+xml;base64,PD94bWwgdmVyc2lvbj0iMS4wIiBlbmNvZGluZz0idXRmL
TgiPz48c3ZnIHZlcnNpb249IjEuMSIgaWQ9IkkxheWVYXzEiIHhtbG5zPSJodHRwOi8vd3d3LnczL
m9yZy8yMDAwL3N2ZyIgeG1sbnM6eGxpbnM9Imh0dHA6Ly93d3cuZm91bmFibGUtYmFja2dyb
3VuZDpuZxcgMCawIDMyIDMyOyIgeG1sOnNwYWNlPSJwcmVzZXJ2ZSI+PHN0eWxlIHR5cGU9InRle
HQvY3NzIj4uc3Qwe2ZpbGw6I0ZGOTI5Mjt9LnN0MXtmaWxsOiNFODdFN0U7fS5zdDdJ7ZmlsbDojR
kZDM0Mz030uc3Qze2ZpbGw6IzIxRDRGRt9LnN0NHtmaWxsOiM2M0E3RDM7fS5zdDdV7ZmlsbDojQ
zFFN0ZGO30uc3Q2e2ZpbGw6I0ZGRkZGRjt9LnN0N3tmaWxsOiM4M0Q2Qjk7fS5zdDdH7ZmlsbDojN
DZDNjhDO30uc3Q5e2ZpbGw6I0JCRThEODt9LnN0MTB7ZmlsbDojRkVCMTdEO30uc3QxMXtmaWxsO
iNERDk2NjY7fS5zdDEye2ZpbGw6I0ZFRDRCNzt9LnN0MTN7ZmlsbDojRjJBMkEyO30uc3QxNHtma
WxsOiNGMUM1QzU7fS5zdDEle2ZpbGw6I0RCQjY2Mzt9LnN0MTZ7ZmlsbDojQ0VBMTUxO30uc3QxN
3tmaWxsOiNFQkQ3QTk7fS5zdDE4e2ZpbGw6I0NFQ0VDRt9LnN0MTl7ZmlsbDojQjdCN0I3O30uc
3QyMHTmaWxsOiNFNEU0RTQ7fS5zdDIxe2ZpbGw6IzY1QUFEMTt9LnN0MjJ7ZmlsbDojRTU3QTdBO
30uc3QyM3tmaWxsOiNFNkE2RTg7fS5zdDI0e2ZpbGw6I0Q2OEFENjt9LnN0MjV7ZmlsbDojRkZDQ
0ZF030uc3QyNntmaWxsOiM5OENFNUY7fS5zdDI3e2ZpbGw6IzhDQUYyQzt9LnN0Mjh7ZmlsbDojQ
zZFM0E3O30uc3QyOXtmaWxsOiNGRkI1Nzg7fS5zdDMwe2ZpbGw6I0VEOUY2NDt9LnN0MzF7Zmlsb
DojRkZENkI1O30uc3QzMntmaWxsOiNGNEExRUY7fS5zdDMze2ZpbGw6I0REODdERDt9LnN0MzR7Z
mlsbDojRjldQkY2O30uc3QzNXtmaWxsOiNBOEEYRjQ7fS5zdDM2e2ZpbGw6Izg4ODVfODt9LnN0M
zd7ZmlsbDojQ0ZDQ0Y4O30uc3QzOHTmaWxsOiNCQ0JDQkM7fS5zdDM5e2ZpbGw6I0E4QThBODt9L
nN0ND7ZmlsbDojREFEQURBO30uc3Q0MXtmaWxsOiM3N0NDREI7fS5zdDQye2ZpbGw6IzREQkNDM
Tt9LnN0NDN7ZmlsbDojQjRFM0VCO30uc3Q0NHtmaWxsOiNGRkI3QTQ7fS5zdDQle2ZpbGw6I0Y2O
UE3Qjt9LnN0NDZ7ZmlsbDojRkZEN0NEO30uc3Q0N3tmaWxsOiM3MUM4RjQ7fS5zdDQ4e2ZpbGw6I
zhEQzk3Nzt9LnN0NDl7ZmlsbDojN0NBODUxO30uc3Q1MHTvcGFjaXR5OjAuNDU7ZmlsbDojRkZGR
kZGO308L3N0eWxlPjxnpjxwYXRoIGNsYXNzPSJzdDM1IiBkPSJNMjEuNSwwLjVoLTE3Yy0xLjEsM
C0yLDAuOS0yLDJ2MjdjMCwLjEsMC45LDI5MiwyADIzYzEuMSwwLDItMC45LDI5MnYtMjFMMjEuN
SwwLjV6Ii8+PHBhdGggY2xhc3M9InN0MzYiIGQ9Ik0yNy41LDMyaC0yM0MzLjEsMzIsMiwxMC45L
DI5MjkuNXItMjddMiwxLjEsMy4xLDA5NC41LDBoMTcuMkwzM0wLjN2MjEuMkMzM0wLjEsMjIsMi
jksMzIsMjcuNSwzMnogTTQuNSwzQzMuNywxLDMsMS43LDMsMi41djI3QzMzMzAuMywzLjcsMzEsN
C41LDMxaDIzYzAuOCwwLDEuNS0wLjcsMS41LTEuNVY4LjddMjEuMywxSDQuNXoiLz48L2c+PGc+P
HBhdGggY2xhc3M9InN0MzciIGQ9Ik0yMS41LDAuNXI3YzAsMC42LDAuNCwwLDEsMWg3TDIxLjUsM
C41eiIvPjxwYXRoIGNsYXNzPSJzdDM2IiBkPSJNMjkuNSw5aC03QzIxLjcsOSwyMSw4LjMsMjEsN
y41di03YzAtMC4yLDAuMS0wLjQsMC4zLTAuNWMwLjItMC4xLDAuNCwwLDAuNSwwLjFsOCw4QzMwL
DguMywzM0wLjUsMzAsOC43QzI5LjksOC45LDI5LjcsOSwyOS41LdL6IE0yMiwxLjd2NS44QzIyL
DcuOCwyMi4yLDgsMjIuNSw4aDUuOEwyMiwxLjd6Ii8+PC9npjxnpjxwYXRoIGNsYXNzPSJzdDYiI
GQ9Ik0xMC4xLDE0LjhjMC40LDA5MC44LDAuMSwwLjEsMC4yYzAuMywwLjEsMC41LDAuMywwLjcsM
C42YzAuMiwwLjIsMC4zLDAuNSwwLjQsMC44YzAuMSwwLjMsMC4xLDAuNiwwLjEsMC45YzAsMC40L
TAuMSwwLjgtMC4yLDEuMmMtMC4xLDAuMy0wLjMsMC42LTAuNiwwLjhjLTAuMiwwLjItMC41LDAuN
C0wLjksMC41Yy0wLjMsMC4xLTAuNywwLjItMS4xLDAuMkg4Ljh2My40SDd2LTguNkgxMC4xeiBNO
S41LDE4LjddMjMC4zLDA5MC42LTAuMSwwLjgtMC4zc0wLjUsMC4zLTFjLjMC0wLjQtMC4xLTAuO
C0wLjMtMWMtMC4yLTAuMi0wLjUtMC4zLTAuOS0wLjNiOC44djIuNkg5LjV6Ii8+PHBhdGggY2xhc
3M9InN0NiIgdD0iITTE1LDE0Ljh2My4zaDJ2LTMuM2gxLjd2OC42SDE3di0zLjhoLTJ2My44aC0xL
jd2LTguNkgxNXoiLz48cGF0aCBjbGFzc0ic3Q2IiBkPSJNMjMsMTQuOGMwLjQsMCwwLjgsMC4xL
DEuMSwwLjJjMC4zLDAuMSwwLjUsMC4zLDAuNywwLjZjMC4yLDAuMiwwLjMsMC41LDAuNCwwLjhjM
C4xLDAuMywwLjEsMC42LDAuMSwwLjlljMCwwLjQtMC4xLDAuOC0wLjIsMS4yYy0wLjEsMC4zLTAuM
ywwLjYtMC42LDAuOGMtMC4yLDAuMi0wLjUsMC40LTAuOSwwLjVDMjMuNCwwOS45LDIzLDIwLDIyL
jYsMjBoLTAuOXIzLjRoLTEuN3YtOC42SDIzeiBNMjIuNCwwOC43YzAuMywwLDAuNi0wLjEsMC44L
TAuM3MwLjMtMC41LDAuMy0xYzAtMC40LTAuMS0wLjgtMC4zLTFjLjLTAuMi0wLjItMC41LTAuMy0wL
jktMC4zaC0wLjd2Mi42SDIyLjR6Ii8+PC9npjwvc3ZnPg==");
```



```
}

.e-filemanager .e-large-icons .e-fe-doc,
.e-filemanager .e-large-icons .e-fe-docx {
    background-image:
url("data:image/svg+xml;base64,PD94bWwgdmVyc2lvbj0iMS4wIiBlbmNvZGluZz0idXRmLTgiPz48c3ZnIHZlcnNpb249IjEuMSIGaWQ9IkxheWVyXzEiIHhtbG5zPSJodHRwOi8vd3d3LnczLm9yZy8yMDAwL3N2ZyIgeG1sbmM6eGxpbnMs9Imh0dHA6Ly93d3duc2Mub3JnLnZlE5OTkveGxpbnmsiIHg9IjBweCIgeT0iMHY4IiB2aWV3Qm94PSiwIDAgaWZlcmVzZXJ2ZSI+PHN0eWxlIHR5cGU9InRleHQvY3NzIj4uc3Qwe2ZpbGw6IOZGOTI5Mjt9LnN0MXtmaWxsOiNFODdFN0U7fS5zdDJ7ZmlsbDoJRkdDM0MzO30uc3Qze2ZpbGw6IzkxRDGRGTt9LnN0NHtmaWxsOiM2M0E3RDM7fS5zdDV7ZmlsbDoJQzFFN0ZGO30uc3Q2e2ZpbGw6IOZGRkZGRjt9LnN0N3tmaWxsOiM4M0Q2Qjk7fS5zdDh7ZmlsbDoJDZDNjhDO30uc3Q5e2ZpbGw6IOJCRTHeODt9LnN0MTB7ZmlsbDoJRkVCMTdEO30uc3QxMXtmaWxsOiNERdk2NjY7fS5zdDEye2ZpbGw6IOZFRDRCNzt9LnN0MTN7ZmlsbDoJRjJBMcKEyo30uc3QxNHtmaWxsOiNGMUM1QzU7fS5zdDEle2ZpbGw6IORCQjY2Mzt9LnN0MTZ7ZmlsbDoJQ0VBMTUxo30uc3QxN3tmaWxsOiNFQkQ3QTk7fS5zdDE4e2ZpbGw6IOVFQ0VDRTt9LnN0MTl7ZmlsbDoJJdcNOI3O30uc3QyMHTmaWxsOiNFNEU0RTQ7fS5zdDIxe2ZpbGw6IzYlQUFEMTt9LnN0MjJ7ZmlsbDoJRTU3QTDBo30uc3QyM3tmaWxsOiNFNKERTg7fS5zdDI0e2ZpbGw6IOQ2OEFENjt9LnN0MjV7ZmlsbDoJRkZDQ0ZF030uc3QyNntmaWxsOiM5OENFNUY7fS5zdDI3e2ZpbGw6IzhDQUYYQzt9LnN0Mjh7ZmlsbDoJQzZFM0E3O30uc3QyOXtmaWxsOiNGRKil1Nzg7fS5zdDMwe2ZpbGw6IOVEOUY2NDt9LnN0Mzf7ZmlsbDoJRkZENki1O30uc3QzMntmaWxsOiNGNEExRUy7fS5zdDMze2ZpbGw6IOREODDERDt9LnN0MZR7ZmlsbDoJRjldQKYZ2O30uc3QzNXtmaWxsOiNB0EEYrjQ7fS5zdDM2e2ZpbGw6Izg4ODVFODt9LnN0MZd7ZmlsbDoJQ0ZDQ0Y4O30uc3QzOHtmaWxsOiNCQ0JDQkm7fS5zdDM5e2ZpbGw6IOE4QThBODt9LnN0NDB7ZmlsbDoJREFEQURBO30uc3Q0MXtmaWxsOiM3N0NDREI7fS5zdDQye2ZpbGw6IzREQkNDMTt9LnN0NDN7ZmlsbDoJQjRFM0VCO30uc3Q0NHtmaWxsOiNGRKI3QTQ7fS5zdDQle2ZpbGw6IOY2OUE3Qjtz9LnN0NDZ7ZmlsbDoJRkZEN0NEO30uc3Q0N3tmaWxsOiM3MUM4RjQ7fS5zdDQ4e2ZpbGw6IzhEQzk3Nzt9LnN0NDl7ZmlsbDojN0NBODUXo30uc3Q1MhtvcGFjaXR5OjAuNDU7ZmlsbDoJRkZGRkZGO308L3N0eWxlPjxnPjxnPjxwYXR0IGNsYXNzPSJzdDMIIGQ9Ik0yMS41LDAuNWgtMTdjLTEuMSwwLTIsMC45LTIsmnYyN2MwLDEuMSswLjksMiwyLDJoMjNmS4xLDAsMi0wLjksMi0ydi0yMUwyMS41LDAuNXoiLz48cGF0aCBjbGFzc0ic3Q0IiBkPSJNMjcunSwzMmgmtMjNDMy4xLDMYLDIsmZAUOSwyLDI5LjV2LTI3QzIsMS4xLDMuMSswLDQuNSwwaDE3LjJMMZASOC4zdjIxLjJDMZASmZAUSwyC45LDMYLDI3LjUsMzJ6IE00LjUsMUMzLjcsMSswLDEuNywzLDIuNXYYNOMzLDMwLjMsMy43LDMXLdQuNSwzMWgyM2MwLjgsMCMwLjUtMC43LDEuNS0xLjVWOC43TDIxLjMsMuUG0LjV6Ii8+PC9nPjxnPjxwYXR0IGNsYXNzPSJzdDUiIGQ9Ik0yMS41LDAuNXYY2YzASMS4xLDAuOSwyLDIsMmg2TDIxLjUsMC4leiIVPjxwYXR0IGNsYXNzPSJzdDQiIGQ9Ik0yOS41LDloLTZDMjiuMSw5LDIXLDCuOSwyMSw2LjV2LTZjMCoWLjIsMC4xLTAuNCwLjMtMC41YzAuMi0wLjJEsMC40LDAuSMC41LDAuMWw4LdHDMZAsOC4zLDMwLdGuNSwzMCMw4LjdDMjkuOSw4LjksMjkuNyw5LDI5LjUsOXogTTIyLDEuN3Y0LjhDMjIsNy4zLDIyLjcsOCwyMy41LdhONC44TDIyLDEuN3oiLz48L2c+PGc+PHJlY3QgeD0iNyIgeT0iMjMiIGNsYXNzPSJzdDYiIHdpZHROPSixOCigaGVpZ2h0PSiyIi8+PC9nPjxnPjxyWN0IHg9IjciIHk9IjE3IiBjbGFzc0ic3Q2IiB3awRoada0IMTGgIghlaWdodD0iMiIVPjwvZz48Zz48cmVjdCB4PSI3Iib5PSixMSIgY2xhc3M9InN0NiIgd2lkdg9IjExIiBoZWlnaHQ9IjIiLz48L2c+PC9nPjwvc3Znpq==" );
}

.e-filemanager .e-large-icons .e-fe-xml {
    background-image:
url("data:image/svg+xml;base64,PD94bWwgdmVyc2lvbj0iMS4wIiBlbmNvZGluZz0idXRmLTgiPz48c3ZnIHZlcnNpb249IjEuMSIGaWQ9IkxheWVyXzEiIHhtbG5zPSJodHRwOi8vd3d3LnczLm9yZy8yMDAwL3N2ZyIgeG1sbmM6eGxpbnMs9Imh0dHA6Ly93d3duc2Mub3JnLnZlE5OTkveGxpbnmsiIHg9IjBweCIgeT0iMHY4IiB2aWV3Qm94PSiwIDAgaWZlcmVzZXJ2ZSI+PHN0eWxlIHR5cGU9InRleHQvY3NzIj4uc3Qwe2ZpbGw6IOZGOTI5Mjt9LnN0MXtmaWxsOiNFODdFN0U7fS5zdDJ7ZmlsbDoJRkdDM0MzO30uc3Qze2ZpbGw6IzkxRDGRGTt9LnN0NHtmaWxsOiM2M0E3RDM7fS5zdDV7ZmlsbDoJQzFFN0ZGO30uc3Q2e2ZpbGw6IOZGRkZGRjt9LnN0N3tmaWxsOiM4M0Q2Qjk7fS5zdDh7ZmlsbDoJDZDNjhDO30uc3Q5e2ZpbGw6IOJCRTHeODt9LnN0MTB7ZmlsbDoJRkVCMTdEO30uc3QxMXtmaWxsOiNERdk2NjY7fS5zdDEye2ZpbGw6IOZFRDRCNzt9LnN0MTN7ZmlsbDoJRjJBMcKEyo30uc3QxNHtmaWxsOiNGMUM1QzU7fS5zdDEle2ZpbGw6IORCQjY2Mzt9LnN0MTZ7ZmlsbDoJQ0VBMTUxo30uc3QxN3tmaWxsOiNFQkQ3QTk7fS5zdDE4e2ZpbGw6IOVFQ0VDRTt9LnN0MTl7ZmlsbDoJJdcNOI3O30uc3QyMHTmaWxsOiNFNEU0RTQ7fS5zdDIxe2ZpbGw6IzYlQUFEMTt9LnN0MjJ7ZmlsbDoJRTU3QTDBo30uc3QyM3tmaWxsOiNFNKERTg7fS5zdDI0e2ZpbGw6IOQ2OEFENjt9LnN0MjV7ZmlsbDoJRkZDQ0ZF030uc3QyNntmaWxsOiM5OENFNUY7fS5zdDI3e2ZpbGw6IzhDQUYYQzt9LnN0Mjh7ZmlsbDoJQzZFM0E3O30uc3QyOXtmaWxsOiNGRKil1Nzg7fS5zdDMwe2ZpbGw6IOVEOUY2NDt9LnN0Mzf7ZmlsbDoJRkZENki1O30uc3QzMntmaWxsOiNGNEExRUy7fS5zdDMze2ZpbGw6IOREODDERDt9LnN0MZR7ZmlsbDoJRjldQKYZ2O30uc3QzNXtmaWxsOiNB0EEYrjQ7fS5zdDM2e2ZpbGw6Izg4ODVFODt9LnN0MZd7ZmlsbDoJQ0ZDQ0Y4O30uc3QzOHtmaWxsOiNCQ0JDQkm7fS5zdDM5e2ZpbGw6IOE4QThBODt9LnN0NDB7ZmlsbDoJREFEQURBO30uc3Q0MXtmaWxsOiM3N0NDREI7fS5zdDQye2ZpbGw6IzREQkNDMTt9LnN0NDN7ZmlsbDoJQjRFM0VCO30uc3Q0NHtmaWxsOiNGRKI3QTQ7fS5zdDQle2ZpbGw6IOY2OUE3Qjtz9LnN0NDZ7ZmlsbDoJRkZEN0NEO30uc3Q0N3tmaWxsOiM3MUM4RjQ7fS5zdDQ4e2ZpbGw6IzhEQzk3Nzt9LnN0NDl7ZmlsbDojN0NBODUXo30uc3Q1MhtvcGFjaXR5OjAuNDU7ZmlsbDoJRkZGRkZGO308L3N0eWxlPjxnPjxnPjxwYXR0IGNsYXNzPSJzdDMIIGQ9Ik0yMS41LDAuNWgtMTdjLTEuMSwwLTIsMC45LTIsmnYyN2MwLDEuMSswLjksMiwyLDJoMjNmS4xLDAsMi0wLjksMi0ydi0yMUwyMS41LDAuNXoiLz48cGF0aCBjbGFzc0ic3Q0IiBkPSJNMjcunSwzMmgmtMjNDMy4xLDMYLDIsmZAUOSwyLDI5LjV2LTI3QzIsMS4xLDMuMSswLDQuNSwwaDE3LjJMMZASOC4zdjIxLjJDMZASmZAUSwyC45LDMYLDI3LjUsMzJ6IE00LjUsMUMzLjcsMSswLDEuNywzLDIuNXYYNOMzLDMwLjMsMy43LDMXLdQuNSwzMWgyM2MwLjgsMCMwLjUtMC43LDEuNS0xLjVWOC43TDIxLjMsMuUG0LjV6Ii8+PC9nPjxnPjxwYXR0IGNsYXNzPSJzdDUiIGQ9Ik0yMS41LDAuNXYY2YzASMS4xLDAuOSwyLDIsMmg2TDIxLjUsMC4leiIVPjxwYXR0IGNsYXNzPSJzdDQiIGQ9Ik0yOS41LDloLTZDMjiuMSw5LDIXLDCuOSwyMSw2LjV2LTZjMCoWLjIsMC4xLTAuNCwLjMtMC41YzAuMi0wLjJEsMC40LDAuSMC41LDAuMWw4LdHDMZAsOC4zLDMwLdGuNSwzMCMw4LjdDMjkuOSw4LjksMjkuNyw5LDI5LjUsOXogTTIyLDEuN3Y0LjhDMjIsNy4zLDIyLjcsOCwyMy41LdhONC44TDIyLDEuN3oiLz48L2c+PGc+PHJlY3QgeD0iNyIgeT0iMjMiIGNsYXNzPSJzdDYiIHdpZHROPSixOCigaGVpZ2h0PSiyIi8+PC9nPjxnPjxyWN0IHg9IjciIHk9IjE3IiBjbGFzc0ic3Q2IiB3awRoada0IMTGgIghlaWdodD0iMiIVPjwvZz48Zz48cmVjdCB4PSI3Iib5PSixMSIgY2xhc3M9InN0NiIgd2lkdg9IjExIiBoZWlnaHQ9IjIiLz48L2c+PC9nPjwvc3Znpq==" );
}
```

```
WxsOiNGMUM1QzU7fS5zdDE1e2ZpbGw6I0RCQjY2Mzt9LnN0MTZ7ZmlsbDojQ0VBMTUxO30uc3QxN
3tmaWxsOiNFQkQ3QTk7fS5zdDE4e2ZpbGw6I0NFQ0VDRTt9LnN0MTl7ZmlsbDojQjdCN0I3O30uc
3QyMHTmaWxsOiNFNEU0RTQ7fS5zdDIxe2ZpbGw6IzY1QUFEMTt9LnN0MjJ7ZmlsbDojRTU3QTdBO
30uc3QyM3tmaWxsOiNFNkE2RTg7fS5zdDI0e2ZpbGw6I0Q2OEFENjt9LnN0MjV7ZmlsbDojRkZDQ
0ZF030uc3QyNntmaWxsOiM5OENFNUY7fS5zdDI3e2ZpbGw6IzhDQUYyQzt9LnN0Mjh7ZmlsbDojQ
zZFM0E3O30uc3QyOXtmaWxsOiNGRkI1Nzg7fS5zdDMwe2ZpbGw6I0VEOUY2NDt9LnN0MzF7Zmlsb
DojRkZENkI1O30uc3QzMntmaWxsOiNGNEExRUY7fS5zdDMze2ZpbGw6I0REODdERDt9LnN0MzR7Z
mlsbDojRjldQkY2O30uc3QzNXtmaWxsOiNB0EEYRjQ7fS5zdDM2e2ZpbGw6Izg4ODVFODt9LnN0M
zd7ZmlsbDojQ0ZDQ0Y4O30uc3QzOHTmaWxsOiNCQ0JDQkM7fS5zdDM5e2ZpbGw6I0E4QThBODt9L
nN0NDB7ZmlsbDojREFEQURBO30uc3Q0MXtmaWxsOiM3N0NDREI7fS5zdDQye2ZpbGw6IzREQkNDM
Tt9LnN0NDN7ZmlsbDojQjRFM0VCO30uc3Q0NHTmaWxsOiNGRkI3QTQ7fS5zdDQle2ZpbGw6I0Y2O
UE3Qjt9LnN0NDZ7ZmlsbDojRkZEN0NEO30uc3Q0N3tmaWxsOiM3MUM4RjQ7fS5zdDQ4e2ZpbGw6I
zhEQzk3Nzt9LnN0NDl7ZmlsbDojN0NBODUxO30uc3Q1MHTvcGFjaXR5OjAuNDU7ZmlsbDojRkZGR
kZGO308L3N0eWxlPjxnPjxwYXR0IGNsYXNzPSJzdDQ4IiBkPSJNMjEuNSwwLjVoLTE3Yy0xLjEsM
C0yLDAuOS0yLDJ2MjdjMCwLjEsMCA45LDIsMiwyaDIzYzEuMSwwLDItMCA45LDItMnYtMjFMMjEuN
SwwLjV6Ii8+PHBhdGggY2xhc3M9InN0NDkiIGQ9Ik0yNy41LDMyaC0yM0MzLjEsMzIsMiwzMCA45L
DIsMjkuNXytMjddMiwxLjEsMy4xLDA5NC41LDBoMTcuMkwzMCA4LjN2MjEuMkMzMCA45LDI4L
jksMzIsMjcuNSwzMnogTTQuNSwxQzMuNywxLDMsMS43LDMsMi41djI3QzMsMzAuMywzLjcsMzEsN
C41LDMxaDIzYzAuOCwwLDEuNS0wLjcsMS41LTEuNVY4LjddMjEuMywzSDQuNXoiLz48L2c+PGc+P
HBhdGggY2xhc3M9InN0NTAiIGQ9Ik0yMS41LDAuNXI3YzAsMCA42LDAuNCwwLDEsMWg3TDIxLjUsM
C41eiIvPjxwYXR0IGNsYXNzPSJzdDQ5IiBkPSJNMjkuNSw5aC03QzIxLjcsOSwyMSw4LjMsMjEsN
y41di03YzAtMCA4yLDAuMS0wLjQsMCA4zLTAuNWwLjItMCA4xLDAuNCwwLDAuNSwwLjFsOCw4QzMwL
DguMywzMCA4LjUsMzAsOC43QzI5LjksOC45LDI5LjcsOSwyOS41LDl6IE0yMiwxLjd2NS44QzIyL
DcuOCwyMi4yLDgsMjIuNSw4aDUuOEwyMiwxLjd6Ii8+PC9nPjxnPjxwYXR0IGNsYXNzPSJzdDYiI
GQ9Ik03LjcsMTQuOGwxLjEsMi44bDEuMS0yLjhoMS45bC0yLDQuMmwyLjEsNC4zSDEwbC0xLjItM
i45bC0xLjIsMi45SDUuNkw3LjgsMTlsLTIiNC4ySDcuN3oiLz48cGF0aCBjbGFzc20ic3Q2IiBkP
SJNMTUuMiwxNC44bDEuMi42aDBsMS4yLTZoMi42djguNmgtMS42di02LjhoMGwtMS41LDYuOGgtM
S4zbC0xLjUtNi44aDB2Ni44aC0xLjZ2LTguNkgxNS4yeiIvPjxwYXR0IGNsYXNzPSJzdDYiIGQ9I
k0yMS41LDE0LjhoMS43VjIyaDMuMXIxLjRoLTQuOFYxNC44eiIvPjwvZz48L3N2Zz4=");
}
```

```
.e-filemanager .e-large-icons .e-fe-folder {
    background-image:
```

```
url('data:image/svg+xml;base64,PD94bWwgdmVyc2lvbj0iMS4wIiBlbmNvZGluZz0idXRmL
TgiPz48c3ZnIHZlcnNpb249IjEuMSIgaWQ9IkxheWVyXzEiIHhtbG5zPSJodHRwOi8vd3d3LnczL
m9yZy8yMDAwL3N2ZyIgeG1sbnM6eGxpbnM9Imh0dHA6Ly93d3cudzMub3JnLzE5OTkveGxpbnMsiI
Hg9IjBweCIgeT0iMHB4IiB2aWV3Qm94PSIwIDAgaWQ9IiBkPSJNMjkuNSw5aC03QzIxLjcsOSwyMSw4LjMsMjEsN
y41di03YzAtMCA4yLDAuMS0wLjQsMCA4zLTAuNWwLjItMCA4xLDAuNCwwLDAuNSwwLjFsOCw4QzMwL
DguMywzMCA4LjUsMzAsOC43QzI5LjksOC45LDI5LjcsOSwyOS41LDl6IE0yMiwxLjd2NS44QzIyL
DcuOCwyMi4yLDgsMjIuNSw4aDUuOEwyMiwxLjd6Ii8+PC9nPjxnPjxwYXR0IGNsYXNzPSJzdDYiI
GQ9Ik03LjcsMTQuOGwxLjEsMi44bDEuMS0yLjhoMS45bC0yLDQuMmwyLjEsNC4zSDEwbC0xLjItM
i45bC0xLjIsMi45SDUuNkw3LjgsMTlsLTIiNC4ySDcuN3oiLz48cGF0aCBjbGFzc20ic3Q2IiBkP
SJNMTUuMiwxNC44bDEuMi42aDBsMS4yLTZoMi42djguNmgtMS42di02LjhoMGwtMS41LDYuOGgtM
S4zbC0xLjUtNi44aDB2Ni44aC0xLjZ2LTguNkgxNS4yeiIvPjxwYXR0IGNsYXNzPSJzdDYiIGQ9I
k0yMS41LDE0LjhoMS43VjIyaDMuMXIxLjRoLTQuOFYxNC44eiIvPjwvZz48L3N2Zz4=");
}
```

```
C0yLTAuOS0yLTJ2LTE5YzAtMS4xLDAuOS0yLDItMmgxMC40bDMuNSwzLjFoMTMuMmMxLjEsMCwyL
DAuOSwyLDJ2MTUuOUMzMS41LDI2LjYsMzAuNiwyNy41LDI5LjUsMjcucXoiLz48cGF0aCBjbGFzc
z0ic3Q0IiBkPSJNMjkuNSwyOGgtMjdDMS4xLDI4LDAzMjYuOSwwLDI1LjV2LTE5QzAsNS4xLDEuM
Sw0LDIuNSw0aDEwLjZsMy41LDMuMWgxM2MxLjQsMCwyLjUsMS4xLDIuNSwyLjV2MTUuOUMzMiwNyN
i45LDMwLjksMjgsMjkuNSwyOHogTTIuNSw1QzEuNyw1LDEsNS43LDEsNi41djE5QzEsMjYuMywzL
jcsMjcsMi41LDI3aDI3YzAuOCwwLDEuNS0wLjcsMS41LTEuNVY5LjZjMC0wLjgtMC43LTEuNS0xL
jUtMS41SDE2LjJMMTIuNyw1SDIuNXoiLz48L2c+PC9zdmc+');
}

.e-filemanager .e-large-icons .e-fe-json {
background-image:
url("data:image/png;base64,iVBORw0KGgoAAAANSUheUgAAAOEAAADhCAMAAAAJbSJIAAAAK
FBMVEXp6eD///+Xd6jZ18qSb6Xr6+HW0NOgg6/t7uOUc6aScKTm5tvs5+/Qw9fOxc2OaqG3o8Oxn
LnBr8qmi7S/sMPBtMT49vmZeqrMwsy2pL3LvdPl493DssypkbTl3umjibDZz9/d2di1oMHY7vSnj
rLn4evfluTXzN3TzNHGusjcl9e0obytl7f39/SkiLOGXZqgHBjVAAAKqUleQVR4nO2da3vqIBKAN
QtN4FRjqzVeJ5dqg611//+/28QAgVwwoFg4y3zp04SM82bIADNE012Z/MDOrwj8kZqlJB3ZyV/iS
+WP1K67Ef78GmDnz3/uhihT9HuAKeHdECV6ftGFGeG9ECVqfu8pzAnvhGgz4X0QJUP+EZAQ3gXRb
sJ7IFpOeAdE2wlvR7Se8GZE+wlvrXSA8EZEFwhvQ3SC8CZENwhvQXSE8AZEwVj1EZ0h1EZ0h1AX0
SFCTUSXCPUQnSLUQnSLUAfrMUINRNci1RGdi1RGdi19QFdfBQkVEFwnVEJ0kVEJ0k1AF0VFCBURXC
dsjOkvYgtFdwraIDhO2RHSZsB2i04StEN0mbIPoOGLRNcJryM6T3gV0X3Ca4j/AOEVxH+BUI54b
0IYpaJ1ZXYhpLuU1AiliHclhNH8OD0tp3P17VTRZro8TSfziFz6R00k+1HvSRh9LQFGqeAnVUR4A
JcLwXKj1QPgQwjhN0ABEWUn0isRmEYaG+oeQ7jC1C8Aik6EfcCuxbuOBuIDCOGKOTC1sq9IOcnuT
oB2Gh3VPGEO5WwM4o2igZuYuxqM1BGNE0LRxG9VE6MXwF8/V7z8AYTRiYUKDMBU/UGC0ziNw1SH8
h0yTgJntI/iZf9rrhPwo/lXf0m1IOVgY5yQRgr8ls5L9LZPp9dFI6IGHGwjJkakg+mEQV7Pit0ox
SuNE37nhHhy2/b36Jg7EU2ti3xBWuNgWWcfEL54wrJ4Qk94TTxhW/GEzWIJYZbEkTdwmxBGh+lpe
pAxuk0IN2GWiWghJE/lNiFdfiDUvPpzmjDasdXfstF8lwnhV5EEAF+NrVwmPBZ5HHZ8FwmjUZGKQ
42rP5cJ+XyhpJXDhJ05l/NtHC8sJtyRGNKcXyKsqrh5AQ8P4Fq4bRDThJA8ZECSCYar3HqwarYeb
qinLSOknSsIJfLACEcBwCAYSaZtcI6udoZ6MUwYhQRQnmqL4FP/qyNvQpJtgcTrtWKWkd1izQMdk
au510iNqlKsXZgkhNGIPjxAo/hbUjanUx/wplRHNER4We8dziwXr5zlrAodd1Ivrg7RtfVkiWYIo
81o+pIU45xk0dBe5sXMACQv01HLqrcRwugYp+u9oDBocrsLM61cnS3VHx9baTVBCJ/4kmHaq9RLY
rUoSwBBb9y4EuHFBCgtVdDbfboPYBpvl6LiVkf3EcKtyIhssFct2AnUb50RwqlIGN4NMRQJW42MJ
gj5qHex5MbaIZVoJd65doOsEUL4FGRboApTNLZQ1ACoihuXbZJKWgUaQ+Mh7Bwmb6eCUBayaCvF6
iLle3mbHFr2ftOEwdyLovk3teke4TRigRRM59xmxWtiijA3ql/MSzW5CmH7ckDfhnpkpWgC2oW91
OVXnMISvilg68/PxCxh0bXkqzo4f5se5ZExOmt2eMOE8Ik6UWZ/2gojjGTbFtnOI7Cxa43fiVpEU
zgnjWSZDnqrQsvyNEWuTeIgSJ5WINlzAw9YMyobJ2xRe6JBpDnjbXW+1BN2PGEunrBZPKEnvCae8
NLGE3Y8oUw8oSe8Jq32tdG1hayN24R5Khs1ej1ue3bgJgEYJLLVu+OEQRcnbnXp/tNiwu/rUaTTI
tfGopF978zQJGCo8xIvpyfJ1dj33hPdyxSg09WUaKMOGHVowU71PeIHELKSH0pGR+VX6y4qDsdpw
kogypebJuSLiQiDs3p2f34uXiFtWTIUxDghv/UwtVB1R1MnCVmioTSxXC/mCYUtFC13FxtCMsH5U
6ixqcM8YdpPOSNveh8f6Gw8egChgBirfqC58NYq5T8CMJONEH01Yqz9nOIA8WYgPgGHEHai+ehS2
QehRizNXqhJZ60jzR0djyHMDvJ9HUCjnRfYIJyMRpNN6516ZXkQYSev7GvOaRSq9lV5HOFviSf0h
PaLJ/SE9osn9IT2iyf0hPaLJ/SE9osn9IT2iyf0hPaLJ/SE9osWIXRjtAifXBItwgS4I4kWofjCs
d0SekJPaL14Qk9ov3hCT2i/eEJP2CzoIsYBrooxwvPrRYbl4xjgcDgchkEMcN11AQJxkLY4JzGuv
T8IX6Si9yI1V5giRIO85aL0rUDBYk90jP/OznH1uvj0+U4afAxqbgJa9C5S5vi8HN1WEU0Tvgpbm
HFP1PNccjECr2OhwazSz8FzfQYHhMNDcrR6Sx5JiM5juSJ8fi83GJ9EEkZYutQKQnSu6FkL5oNB3
WfNkX7MCD+E41YQBhUPinraA/2Hif2REXaFb6ezgRB80Ks/FoPBYLEuuRCdukWD7WnQYzdkwZtdE
L7zTrSBkh7Scxjjyy82gl7CX8EyRjOQ/6RjzDrtpK+SSubYfeVwLCCkYfRvcecFa2Lq4h1zLE6IG
985X3OEY7sIiWxj+mE+QDuiesj/3CH1K/c0c4Q8jwWEKpfHrIEw/pu3nwlhBRMt4+IoT9gturkFh
EH+/7JhpkrKsL4JjzFXEUgLEYM6wnxLD9ddjHaEHtmRIGwy34Zwx7CRX0vpb5KKify44VvrcJn6
kQLCEE+IXuvTrYzswDixPr/MywFLG65M8AWUNIR4teHSJaNhmIyERNUSYkSNS5FhCYEXONqo8iW
uTnBtVTQ3pVifC/YvClgZAGk+54GZdBqIN3VUJyY/ZlwphanlhDyIa89BkqM9I5a41enI+j7P1lh
IDcltZMWkEY4D1T8LwVGAGBr1EUk5lhbTBARNclCNlBGMSfhYr3LRdyAGGvGSwLovL/mAShSwC2h
DCIt9wacZ9UJiolcZaewtWmZAC6xCdbCAPE4k0m23L4UPIhW1Fms1ZrCLNsG5eNGtBvXdV5DoXpu
kWE2bKveBzJVEUnlvLWJ8gqwtTKkCVN836qMx5e/iH3ah1bRpj2LzKJIeclcxqSoKvMaXKf0hEjP
NtGyDKHxzfLNealOSEmt2YfWkfIxxZiKeGtDBe0H+7qCdmIsbCPkHa/XFXz+pCE0sr6kBDsJXLXQ
kLyCOWVi8Y1Phnyqmt86u14zxtmI+FzSGNcqlLQEA+ap2H9WagVPJJwcY2QxHdynrPczLXxs5YmQ
sBNds0SivJmuCtif/EsLxf0pGcRtPumS9W0VIH192pmcI+iBD/HXAlWDykBrMG52EsGF8yia0ez
1zOmwtKd/5LeCqEdMQwTJiFivHnKY4BxhjErIzEPjHttvVxgFSuURY+lzUxS14vjBk6BV2I8CnIG
```

```
h/GxYLFICGru6xnsx6rMnU/ixh4OfjcGwzD8DygLbjKGWZxf9zbDofbovYkPJslhNyIYY4Q7+sbF
uMbrlYPxZILfTir8llbP+TnBsWIYY5wV9+O2zhQLQCnzhiH+Li2RPpZXwMWjjLl5ghR+FHTbMDd/
bDqw31ZC15WGwlKE7laQrA2Tpj9Xt+stM3gbyh8HJ6J5o/Ltmc3Ci/ERr2gbHJ9woMmY82O+BicF
x/UwOdZCCrnl709OT9eb2sSwlkjvKU6xh+vqKyjiZDO+ozPadJxIhnudsMQxHUVGISz/U7nYYjie
r68URxkOpKGNqh2Oln90fsTkg+TnW0nd9wQ5/cmekL7xRP+vxKe7hfqTAu3mUyFsF8diW0V0Nci7
K5ccSJaSShkhD+r5kmJRYLA6keTsNtdnyq5TeskOa2lDP8DUTgufjy2Bo0AAAAASUVORK5CYII="
);
}
</style>
```

```
{% previewsample "page.domainurl/code-snippet/file-manager/custom-thumbnail-cs1" %}
```

### Nested items in Vue File manager component

FileManager can be rendered inside the other components like Tab, Dialog, and more.

- [Adding file manager inside the dialog](#)
- [Adding file manager inside the tab](#)

### Adding file manager inside the dialog

The following example shows the file manager component rendered inside the dialog. Click the browse button in the Uploader element to open the File Manager inside the Dialog control.

### APP.VUE

```
<template>
  <div>
    <div class="control-section">
      <div id='container' class="fileupload">
        <ejs-uploader ref="uploadObj" id='defaultfileupload'
name="UploadFiles">
          </ejs-uploader>
          <ejs-button id="openBtn" v-
on:click.native="btnClick">Browse...</ejs-button>
        </div>
        <div id='target' class="control-section">
          <ejs-dialog ref="uploadDialog" id="dialog" v-
bind:visible="false" :header='dialogHeader'
:animationSettings='animationSettings' :showCloseIcon='showCloseIcon'
:open="dialogOpen" :close="dialogClose" :target='target'
:width='dialogWidth'>
            <ejs-filemanager ref="filemanagerObj" id="filemanager"
:ajaxSettings='ajaxSettings' v-bind:allowMultiSelection="false"
:fileOpen="onFileOpen" >
              </ejs-filemanager>
            </div>
          </div>
        </div>
      </div>
    </div>
  </template>
<script>
import Vue from "vue";
import { UploaderPlugin } from '@syncfusion/ej2-vue-inputs';
import { ButtonPlugin } from "@syncfusion/ej2-vue-buttons";
import { DialogPlugin } from '@syncfusion/ej2-vue-popups';
```

```

import { FileManagerPlugin, NavigationPane, Toolbar, DetailsView,
FileManagerComponent } from "@syncfusion/ej2-vue-filemanager";
Vue.use(FileManagerPlugin);
Vue.use(DialogPlugin);
Vue.use(UploaderPlugin);
Vue.use(ButtonPlugin);
let hostUrl = 'https://ej2-aspcore-service.azurewebsites.net/';
export default {
  data () {
    return {
      dialogHeader: 'Open',
      showCloseIcon: true,
      target: '#target',
      animationSettings: { effect: 'None' },
      dialogWidth: '850px',
      ajaxSettings: {
        url: hostUrl + 'api/FileManager/FileOperations',
        getImageUrl: hostUrl + 'api/FileManager/GetImage',
        uploadUrl: hostUrl + 'api/FileManager/Upload',
        downloadUrl: hostUrl + 'api/FileManager/Download'
      }
    };
  },
  provide: {
    filemanager: [DetailsView, NavigationPane, Toolbar]
  },
  methods: {
    btnClick: function(event) {
      this.$refs.uploadDialog.show();
    },
    // Uploader will be hidden, if Dialog is opened
    dialogOpen: function() {
      var fileObj = this.$refs.filemanagerObj.ej2Instances;
      fileObj.refreshLayout();
      document.getElementById('container').style.display = 'none';
    },
    // Uploader will be shown, if Dialog is closed
    dialogClose: function() {
      var fileObj = this.$refs.filemanagerObj.ej2Instances;
      fileObj.path = "/";
      document.getElementById('container').style.display = 'block';
    },
    // File Manager's fileOpen event function
    onFileOpen: function(args) {
      let file = args.fileDetails;
      if (file.isFile) {
        args.cancel = true;
        this.$refs.uploadObj.files = [{name: file.name, size:
file.size, type: file.type }];
        this.$refs.uploadDialog.hide();
      }
    }
  }
}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";

```

```

@import "../node_modules/@syncfusion/ej2-icons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-inputs/styles/material.css";
@import "../node_modules/@syncfusion/ej2-popups/styles/material.css";
@import "../node_modules/@syncfusion/ej2-buttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-splitbuttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-navigations/styles/material.css";
@import "../node_modules/@syncfusion/ej2-layouts/styles/material.css";
@import "../node_modules/@syncfusion/ej2-grids/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-filemanager/styles/material.css";

.fileupload {
    max-width: 500px;
    margin: auto;
}
#openBtn {
    position: absolute;
    top: 25px;
    margin-left: 13px;
}
#target {
    height: 550px;
}
#dialog {
    top: 20px !important;
    max-height: 500px !important;
    left: 30px !important;
}
</style>

```

{% previewsample "page.domainurl/code-snippet/file-manager/file-upload-cs1" %}

#### *Adding file manager inside the tab*

The following example demonstrate that the file manager component is placed inside the content area of tab element.

#### **APP.VUE**

```

<template>
  <div class="e-tab-section">
    <div class="col-lg-8 content-wrapper control-section">
      <div class="e-sample-resize-container">
        <ejs-tab ref="tabObj" id="tab_orientation" :showCloseButton=true
heightAdjustMode='Auto'>
          <e-tabitems>
            <e-tabitem :header='headerText0'
:content='OverviewTemplate'></e-tabitem>
            <e-tabitem :header='headerText1'
:content='FileManagerTemplate'></e-tabitem>
          </e-tabitems>
        </ejs-tab>
      </div>
    </div>
  </div>
</template>
<script>
import Vue from "vue";

```

```

import { FileManagerPlugin, DetailsView, NavigationPane, Toolbar } from
"@syncfusion/ej2-vue-filemanager";
import { TabPlugin } from "@syncfusion/ej2-vue-navigations";
Vue.use(TabPlugin);
Vue.use(FileManagerPlugin);
var Template1 = Vue.component("demo", {
  template: ` <div><div class="cnt-text">Overview</div><div>The file manager
component contains a context menu for performing file operations, large-
icons view for displaying the files and folders, and a breadcrumb for
navigation. However, these basic functionalities can be extended by using
the additional feature modules like toolbar, navigation pane, and details
view to simplify the navigation and file operations within the file
system</div></div>`,
  data() {
    return {
      data: {}
    };
  }
});
var Template2 = Vue.component("demo", {
  template: ` <div><div class="content-title">
    <div class="cnt-text">File manager with default
functionalities</div>
    </div>
    <ejs-filemanager id="file-manager"
:ajaxSettings="ajaxSettings">
    </ejs-filemanager></div>`,
  data() {
    return {
      ajaxSettings:
        {
          url: "https://ej2-aspcore-
service.azurewebsites.net/api/FileManager/FileOperations",
          getImageUrl: "https://ej2-aspcore-
service.azurewebsites.net/api/FileManager/GetImage",
          uploadUrl: "https://ej2-aspcore-
service.azurewebsites.net/api/FileManager/Upload",
          downloadUrl: "https://ej2-aspcore-
service.azurewebsites.net/api/FileManager/Download"
        },
    };
  },
  provide: {
    filemanager: [DetailsView, NavigationPane, Toolbar]
  }
});
export default {
  data () {
    return {
      headerText0: { text: 'Overview' },
      headerText1: { text: 'FileManager' },
      OverviewTemplate: function () {
        return {
          template : Template1
        }
      },
      FileManagerTemplate: function () {

```



```

        return {
            template : Template2
        }
    },
};
},
provide: {
    filemanager: [DetailsView, NavigationPane, Toolbar]
}
}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-icons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-inputs/styles/material.css";
@import "../node_modules/@syncfusion/ej2-popups/styles/material.css";
@import "../node_modules/@syncfusion/ej2-buttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-splitbuttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-navigations/styles/material.css";
@import "../node_modules/@syncfusion/ej2-layouts/styles/material.css";
@import "../node_modules/@syncfusion/ej2-grids/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-
filemanager/styles/material.css";
.content-title {
    height: 50px;
    display: table;
    margin: 0 auto;
}
.cnt-text {
    text-align: center;
    font-size: 18px;
    font-weight: 600;
    padding: 10px;
}
</style>

```

{% previewsample "page.domainurl/code-snippet/file-manager/file-tab-cs1" %}

### Create the custom file provider using NodeJS

Here we manipulate the Azure Blob Storage to supply the necessary data for the File Manager. We achieve this by utilizing NodeJS to fetch the required data from the Azure blob storage.

NodeJS acts as the bridge between the File Manager component and Azure Blob Storage, allowing seamless communication and data retrieval. Through this integration, the File Manager can access and interact with the data stored in Azure Blob Storage, enabling smooth file management operations.

### Prerequisites

- Valid Azure blob storage account. ( accountName, accountKey, endpointSuffix)
- Node version 14 above.



### *Introduction to Azure Blob Storage*

Azure Blob Storage is a cloud-based object storage service provided by Microsoft Azure. It is designed to store and manage unstructured data, also known as "blobs" in the cloud. Blobs can be any type of data, such as images, videos, documents, backups, logs, and more.

### *Key concepts of Azure Blob Storage*

**Containers:** In Azure Blob Storage, data is organized into containers. Containers are logical units that can hold one or more blobs. Think of them as directories or folders that help organize the data.

**Blobs:** Blobs are the actual data objects stored in Azure Blob Storage.

By understanding the fundamental concepts and use cases of Azure Blob Storage, you will be well-prepared to proceed with setting up and interacting with it using NodeJS in the custom File Provider.

### *Create NodeJS project*

Following the steps to create the NodeJS project.

Create a new directory for your project and run the following command to initialize a new NodeJS project. This will create a package.json file.

```
`ts
npm init
`
```

Install the following packages.

- express
- @azure/storage-blob
- archiver
- body-parser
- cors
- esm
- multer

Open your text editor or integrated development environment (IDE) and create the index.js file start writing your NodeJS code. This file will serve as the entry point of your application.

```
`ts
const express = require('express');
const app = express();
const port = 3000;
app.get('/', (req, res) => {
  res.send('Hello, NodeJS!');
});
app.listen(port, () => {
  console.log(Server running on http://localhost:${port});
});
```

To start your NodeJS application, simply run the following command in your terminal, pointing to the entry point file:

```
`ts
node index.js
```

#### *Initialize container client*

We need to first get the BlobServiceClient. By using the connection string, we can obtain the BlobServiceClient. So, format the connection string as shown below.

```
`ts
Const connectionString =
DefaultEndpointsProtocol=https;AccountName=${accountName};AccountKey=${accountKey};E
ndpointSuffix=${EndpointSuffix};
```

We can obtain the BlobServiceClient and the **containerClient** using this connection String and the BlobServiceClient. the **containerName** is the container from your Azure blob storage account that you need to access.

```
`ts
import { BlobServiceClient } from "@azure/storage-blob";
const blobServiceClient = BlobServiceClient.fromConnectionString(connectionString);
const containerClient = blobServiceClient.getContainerClient(containerName);
```

#### *File actions*

Need to provide the following action to creating a new folder, copying and moving of files or folders, deleting, uploading, and downloading the files or folders in the file system

#### *Read*

Specify the directory name that needs to be accessed.

```
`ts
const directoryName = 'Files';
```

Create the **app.post** method with URL **‘/fileManager’**.

To identify the action by use this condition **req.body.action === ‘read’**

The following table represents the request parameters of **read** operations.

Parameter	Type	Default	Explanation
action	String	read	Name of the file operation.

path	String	-	Relative path from which the data has to be read.
showHiddenItems	Boolean	-	Defines show or hide the hidden items.
data	[FileManagerDirectoryContent](#)	-	Details about the current path (directory).

*Example for request:*

```
`ts
{
  action: "read",
  path: "/Videos/",
  showHiddenItems: false,
  data: [
    0:{
      name:"Videos",
      size:0,
      dateModified:"2023-09-14T14:28:27.000Z",
      dateCreated: "2023-09-14T11:16:57.000Z",
      hasChild:true,
      isFile:false,
      type:"Directory",
      filterPath:"/",
      fmicon: "e-fe-folder",
      fmiconClass: "e-fe-folder",
      fmid: "fetree0",
      fmmmodified: "September 14, 2023 19:58"
    }
  ]
}
```

The following table represents the response parameters of **read** operations.

Parameter	Type	Default	Explanation
----	----	----	----
cwd	<a href="#">FileManagerDirectoryContent</a>		Path (Current Working Directory) details.
files	FileManagerDirectoryContent[]		Details of files and folders present in given path or directory.
error	<a href="#">ErrorDetails</a>		Error Details

<a id="file-manager-directory-content"></a>

The following table represents the contents of **FileManagerDirectoryContent** in the file manager request and response.

Parameter	Type	Default	Explanation	Is required
----	----	----	----	----
name	String	-	File name	Yes
dateCreated	String	-	Date in which file was created (UTC Date string).	Yes
dateModified	String	-	Date in which file was last modified (UTC Date string).	Yes
filterPath	String	-	Relative path to the file or folder.	Yes
hasChild	Boolean	-	Defines this folder has any child folder or not.	Yes
isFile	Boolean	-	Say whether the item is file or folder.	Yes
size	Number	-	File size	Yes
type	String	-	File extension	Yes
permission	<a href="#">AccessRules</a>	-	File extension	Optional
caseSensitive	Boolean	-	Defines search is case sensitive or not.	Optional
action	String	read	Name of the file operation.	Optional
names	String[]	-	Name list of the items to be downloaded.	Optional
data	FileManagerDirectoryContent	-	Details of the download item.	Optional
uploadFiles	<a href="#">IList&lt;IFormFile&gt;</a>	-	File that are uploaded.	Optional
newName	String	-	New name for the item.	Optional
searchString	String	-	String to be searched in the directory.	Optional
targetPath	String	-	Relative path where the items to be pasted are located.	Optional
targetData	FileManagerDirectoryContent	-	Details of the copied item.	Optional
renameFiles	String[]	-	Details of the renamed item.	Optional

<a id="access-rules"></a>

The following table represents the **AccessRules** properties available for file and folder:

Properties	Applicable for file	Applicable for folder	Description
---	---	---	---
Copy	Yes	Yes	Allows access to copy a file or folder.
Read	Yes	Yes	Allows access to read a file or folder.
Write	Yes	Yes	Allows permission to write a file or folder.
WriteContents	No	Yes	Allows permission to write the content of folder.
Download	Yes	Yes	Allows permission to download a file or folder.

Upload	No	Yes	Allows permission to upload to the folder.
Path	Yes	Yes	Specifies the path to apply the rules, which are defined.
Role	Yes	Yes	Specifies the role to which the rule is applied.
IsFile	Yes	Yes	Specifies whether the rule is specified for folder or file.

*Example for response:*

```
`ts
{
  cwd:
  {
    filterPath: "/",
    hasChild: true,
    name: "Videos",
    size: 0,
    type: "File Folder"
  },
  files:[
    0:{
      dateCreated: "2023-09-14T11:16:57.000Z"
      dateModified: "2023-09-14T11:16:57.000Z"
      filterPath: "/Videos/"
      hasChild: false
      isFile: true
      name: "about.txt"
      size: 29
      type: ".txt"
    }
  ],
  error:null
}
```

[Get image](#)

Create the **app.get** method with URL **‘/fileManager/GetImage’**.

The following table represents the request parameters of **GetImage** operations.

Parameter	Type	Default	Explanation
-----------	------	---------	-------------

----	----	----	----
------	------	------	------

path	String	-	Relative path to the image file
------	--------	---	---------------------------------

The req.query.path contains the exact path of the images. For example: `"/Jack.png"`.

Download the blob (image) from Azure Blob Storage using the blobClient and stores the result in the downloadResponse variable.

Pipe the readableStreamBody from the blob to the res response. It means the image data will be streamed from the Azure Blob Storage directly to the client's browser when the image URL is accessed.

Handle the exception if the image is not available in the given path.

#### Download

Create the **app.post** method with URL **`"/fileManager/Download"`**.

The following table represents the request parameters of *download* operations.

Parameter	Type	Default	Explanation
-----------	------	---------	-------------

----	----	----	----
------	------	------	------

action	String	download	Name of the file operation
--------	--------	----------	----------------------------

path	String	-	Relative path to location where the files to download are present.
------	--------	---	--------------------------------------------------------------------

names	String[]	-	Name list of the items to be downloaded.
-------	----------	---	------------------------------------------

data	<a href="#">FileManagerDirectoryContent</a>	-	Details of the download item.
------	---------------------------------------------	---	-------------------------------

*Example for request:*

```
`ts
{
  action: 'download',
  path: '/Downloads/Testing/',
  names: [ 'About.txt' ],
  data: [
    0:{
      name: 'About.txt',
      type: '.txt',
      isFile: true,
      size: 29,
      dateModified: '2023-09-14T06:03:52.000Z',
      hasChild: false,
      filterPath: '/Downloads/Testing/',
      fmcreated: null,
```

```

fmmodified: 'September 14, 2023 11:33',
fmiconClass: 'e-fe-txt',
fmicon: 'e-fe-txt'
}
]
}
,

```

The **req.body.downloadInput** must be parsed to get the **downloadObj**. Download the blob from Azure Blob Storage using the blobClient.

Download the blob from Azure Blob Storage using the blobClient and Pipe the readableStreamBody to the response object.

Create the archive file to download the multiple Files, Folders and single folders, then pipe the archive to the response.

#### Upload

Create the **app.post** method with URL **'/fileManager/Upload'**.

The following table represents the request parameters of *Upload* operations.

Parameter	Type	Default	Explanation
---- ---- ---- ----			
action	String	Save	Name of the file operation.
path	String	-	Relative path to the location where the file has to be uploaded.
uploadFiles	<code>IList&lt;IFormFile&gt;</code>	-	File that are uploaded.

*Example for request:*

```

`ts
{
path: '/Pictures/',
action: 'save',
data: [
0:{
name: 'Pictures',
type: 'File Folder',
isFile: true,
size: 0,
dateModified: '2023-09-14T06:03:52.000Z',
hasChild: true,

```

```

filterPath: "",
fmid: 'fetree1',
},
],
filename: 'bird (2).jpg'
},
,

```

Multer is a popular middleware used to handle file uploads in Express-based web applications. Create the Multer config to store the upload files in buffer.

```

`ts
const multerConfig = {
storage: memoryStorage()
};
,

```

Need to handle the 3 cases here.

- Save
- Keep Both (action name will be **keepboth**)
- Replace (action name will be **replace**)

Create the **getBlockBlobClient** with the **req.body.filename**. If the blob does not exist, then upload the data to that blob. If the blob already exists, then create an error message containing "File Already Exists" and send the response.

[Create a new folder](#)

The following table represents the request parameters of *create* operations.

Parameter	Type	Default	Explanation
----	----	----	----
action	String	create	Name of the file operation.
path	String	-	Relative path in which the folder has to be created.
name	String	-	Name of the folder to be created.
data	<a href="#">FileManagerDirectoryContent</a>	-	Details about the current path (directory).

*Example for request:*

```

`ts
action: "create",
data: [
0:{

```



```
filterPath: "/",
hasChild: true,
isFile: false,
name: "files",
nodeId: "fe_tree",
size: 0,
type: ""
},
name: "Hello",
path: "/test/"
,
```

Check the existence of the folder, If the folder exists then send the error message containing “Folder already exists”. If it does not exist, then create the folder. Create the folder by creating the file in that folder’s path.

The following table represents the response parameters of *create* operations.

Parameter	Type	Default	Explanation
files	FileManagerDirectoryContent[]	-	Details of the created folder
error	<a href="#">ErrorDetails</a>	-	Error Details

*Example for response:*

```
`ts
{
  cwd: null,
  files: [
    0:{
      dateCreated: "2023-09-14T10:52:25.000Z",
      dateModified: "2023-09-14T10:52:25.000Z",
      filterPath: null,
      hasChild: false,
      isFile: false,
      name: "New",
      size: 0,
      type: "Directory"
    }
  ]
}
```

```
}
],
details: null,
error: null
},
`
```

Rename

The following table represents the request parameters of *rename* operations.

Parameter	Type	Default	Explanation
----	----	----	----
action	String	rename	Name of the file operation.
path	String	-	Relative path in which the item is located.
name	String	-	Current name of the item to be renamed.
newName	String	-	New name for the item.
data	<a href="#">FileManagerDirectoryContent</a>	-	Details of the item to be renamed.

Example for request:

```
`ts
{
  action: "rename",
  data: [
    0:{
      dateCreated: "2023-09-14T10:41:17.000Z",
      filterPath: "/Pictures/Nature/",
      hasChild: false,
      iconClass: "e-fe-image",
      isFile: true,
      name: "seaviews.jpg",
      size: 95866,
      type: ".jpg"
    }
  ],
  newName: "seaview.jpg",
  name: "seaviews.jpg",

```

```
path: "/Pictures/Nature/"
}
```

Renaming can be done by copy the folder or file from the source blob instance to target blob instance. If the file exists, then send the error message as response.

The following table represents the response parameters of *rename* operations.

Parameter	Type	Default	Explanation
files	FileManagerDirectoryContent[]	-	Details of the renamed item.
error	<a href="#">ErrorDetails</a>	-	Error Details

*Example for response:*

```
`ts
{
  cwd:null,
  files:[
    0:{
      name:"seaview.jpg",
      size:95866,
      dateModified:"2023-09-14T11:16:57.000Z",
      dateCreated:"2023-09-14T10:41:17.000Z",
      hasChild:false,
      isFile:true,
      type:".jpg",
      filterPath:"/Pictures/Nature/"
    }
  ],
  error:null,
  details:null
}
```

### Delete

The following table represents the request parameters of *delete* operations.

Parameter	Type	Default	Explanation
----	----	----	----

action	String	delete	Name of the file operation.
path	String	-	Relative path where the items to be deleted are located.
names	String[]	-	List of the items to be deleted.
data	[FileManagerDirectoryContent](#)	-	Details of the item to be deleted.

*Example for request:*

```
`ts
{
  action: "delete",
  path: "/",
  names: ["bird.jpg"],
  data: [
    0:{
      dateModified: "2023-09-14T09:12:53.000Z",
      filterPath: "/",
      hasChild: false,
      iconClass: "e-fe-image",
      isFile: true,
      name: "bird.jpg",
      size: 102182,
      type: ".jpg"
    }
  ]
}
```

To delete the file, directly get the file instance and delete the file. To delete the folder, we need to get all files inside that folder and delete all those files.

Handle the null exception if file or folder is not available.

The following table represents the response parameters of *delete* operations.

Parameter	Type	Default	Explanation
files	<a href="#">FileManagerDirectoryContent[]</a>		Details about the deleted item(s).
error	<a href="#">ErrorDetails</a>		Error Details

*Example for response:*

```
`ts
{
  cwd: null,
  details: null,
  error: null,
  files: [
    0:{
      dateModified: "2023-09-14T09:12:53.000Z",
      filterPath: "/",
      hasChild: false,
      iconClass: "e-fe-image",
      isFile: true,
      name: "bird.jpg",
      size: 102182,
      type: ".jpg"
    }
  ]
}
```

### Details

The following table represents the request parameters of *details* operations.

Parameter	Type	Default	Explanation
----	----	----	----
action	String	details	Name of the file operation.
path	String	-	Relative path where the items are located.
names	String[]	-	List of the items to get details.
data	<a href="#">FileManagerDirectoryContent</a>	-	Details of the selected item.

*Example:*

```
`ts
{
  action: "details",
  path: "/FileContents/",
  names: ["bird.jpg"],
```

```
data: [  
  0:{  
    dateModified: "2023-09-14T09:12:53.000Z",  
    filterPath: "/",  
    hasChild: false,  
    iconClass: "e-fe-image",  
    isFile: true,  
    name: "bird.jpg",  
    size: 102182,  
    type: ".jpg"  
  }  
]  
}
```

To get the file and folder details, iterate the **req.body.names** to get the details of files and folders. If the data is file, then get the file instance and get the properties using the **getProperties** method. If the data is Folder, then get the blobs details under that folder using **listBlobsFlat** method. Get the required properties and send final response. Handled the null exception if the file or folder is not available.

The following table represents the response parameters of *details* operations.

Parameter	Type	Default	Explanation
-----	-----	-----	-----
details	<a href="#">FileManagerDirectoryContent</a>	-	Details of the requested item(s).
error	<a href="#">ErrorDetails</a>	-	Error Details

*Example:*

```
`ts  
{  
  cwd:null,  
  files:null,  
  error:null,  
  details:  
  {  
    created: "2023-09-15T06:04:12.000Z"  
    isFile: true  
    location: "Files/bird.jpg"
```

```
modified: "2023-09-15T06:04:12.000Z"
multipleFiles: false
name: "bird.jpg"
size: "100.0 KB"
}
}
,
```

### Search

The following table represents the request parameters of *search* operations.

Parameter	Type	Default	Explanation
----	----	----	----
action	String	search	Name of the file operation.
path	String	-	Relative path to the directory where the files should be searched.
showHiddenItems	Boolean	-	Defines show or hide the hidden items.
caseSensitive	Boolean	-	Defines search is case sensitive or not.
searchString	String	-	String to be searched in the directory.
data	<a href="#">FileManagerDirectoryContent</a>	-	Details of the searched item.

*Example for request:*

```
`ts
{
  action: "search",
  path: "/asia/",
  searchString: "nature",
  showHiddenItems: false,
  caseSensitive: false,
  data: [
    0:{
      filterPath: "/",
      hasChild: true,
      name: "asia",
      size: 0,
      type: "File Folder",
      fmid: "fetree1"
```

```
}  
]  
}  
,
```

Replace the '\*' in the **req.body.searchString** and assign the result to new variable. Get all blobs under this directory and check that path contains the search string

The following table represents the response parameters of *search* operations.

Parameter	Type	Default	Explanation
-----------	------	---------	-------------

----	----	----	----
------	------	------	------

cwd	<a href="#">FileManagerDirectoryContent</a>	-	Path (Current Working Directory) details.
-----	---------------------------------------------	---	-------------------------------------------

files	FileManagerDirectoryContent[]	-	Files and folders in the searched directory that matches the search input.
-------	-------------------------------	---	----------------------------------------------------------------------------

error	<a href="#">ErrorDetails</a>	-	Error Details
-------	------------------------------	---	---------------

*Example for response:*

```
`ts  
{  
  cwd:  
  {  
    name:"asia",  
    size:0,  
    dateModified:"2023-09-14T14:28:27.000Z",  
    dateCreated:"2023-09-14T11:16:57.000Z",  
    hasChild:true,  
    isFile:false,  
    type:"File Folder",  
    filterPath:"/"  
  },  
  files:[  
    0: {  
      dateModified: "2023-09-15T06:22:00.000Z",  
      filterPath: "/asia/",  
      hasChild: false,  
      isFile: true,  
      name: "about.txt",
```



```
size: 42,  
type: ".txt"  
}  
],  
error:null,  
details:null  
}  
,
```

### Copy and move

The following table represents the request parameters of *copy* operations.

Parameter	Type	Default	Explanation
----	----	----	----
action	String	copy	Name of the file operation.
path	String	-	Relative path to the directory where the files should be copied.
names	String[]	-	List of files to be copied.
targetPath	String	-	Relative path where the items to be pasted are located.
data	<a href="#">FileManagerDirectoryContent</a>	-	Details of the copied item.
targetData	<a href="#">FileManagerDirectoryContent</a>	-	Details of the copied item.
renameFiles	String[]	-	Details of the renamed item.

*Example for request:*

```
`ts  
{  
  action: "copy",  
  path: "/",  
  names: ["bird.jpg"],  
  renameFiles: [],  
  targetPath: "/asia/",  
  targetData: {  
    filterPath: "/",  
    hasChild: true,  
    name: "asia",  
    size: 0,  
    type: "File Folder",
```

```
fmid: "fetree1",
},
data: [
0:{
dateCreated: "2023-09-15T06:04:12.000Z",
dateModified: "2023-09-15T06:04:12.000Z",
filterPath: "/",
hasChild: false,
isFile: true,
name: "bird.jpg",
size: 102182,
type: ".jpg",
fmcreated: "September 15, 2023 11:34",
fmhtmlAttr: {class: "e-large-icon", title: "bird.jpg"},
fmiconClass: "e-fe-image",
fmimageAttr: {alt: "bird.jpg"},
fmimageUrl: "http://localhost:3000/GetImage?path=%2Fbird.jpg&time=1694760243307",
fmmodified: "September 15, 2023 11:34",
}
]
}
`
```

Action name will be **move** for move action.

The following table represents the response parameters of *copy* operations.

Parameter	Type	Default	Explanation
----	----	----	----
cwd	<a href="#">FileManagerDirectoryContent</a>	-	Path (Current Working Directory) details.
files	<a href="#">FileManagerDirectoryContent[]</a>	-	Details of copied files or folders
error	<a href="#">ErrorDetails</a>	-	Error Details

*Example for response:*

```
`ts
{
cwd:null,
```

```
files:[
  0:{
    dateCreated: "2023-09-15T06:55:03.000Z"
    dateModified: "2023-09-15T06:55:03.000Z"
    filterPath: "/asia/"
    hasChild: false
    isFile: true
    name: "bird.jpg"
    previousName: null
    size: 102182
    type: ".jpg"
  }
],
error:null,
details:null
},
`
```

Need to handle two cases.

- Directory copy and move.
- File copy and move.

Create the **isRename** variable to store the is request is rename or not. If the **isRename** is false then check the existence of the folders, and if folder is existing, then send the error message. If **isRename** is true, then don't check the existence of the folder.

To move or copy the folders you need to get all the blobs from that folder and create the new path for each blob and copy the data from the old path to the new path. To move or copy the files copy the data from the source blob client to target client. If the action is move then delete the old blob.

**Note:** To get the complete project, refer to this [link](#)

## FloatingActionButton

### Getting Started with the Vue Floating action button Component in Vue 2

This article provides a step-by-step guide for setting up a Vue 2 project using [Vue-CLI](#) and integrating the Syncfusion Vue Floating action button component using the [Composition API](#) / [Options API](#).

Prerequisites

[System requirements for Syncfusion Vue UI components](#)

### Dependencies

The list of dependencies required to use the Floating Action Button component in your application is given as follows:

```
`js
|-- @syncfusion/ej2-vue-buttons
|-- @syncfusion/ej2-base
|-- @syncfusion/ej2-buttons
|-- @syncfusion/ej2-vue-base
\
```

### Setting up the Vue 2 project

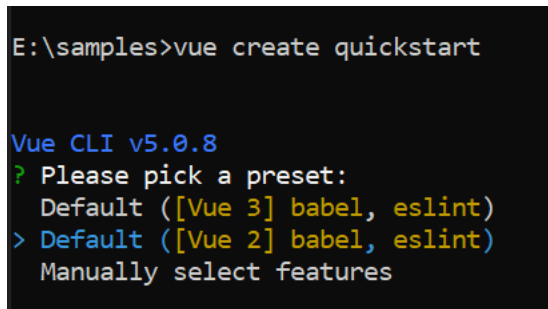
To generate a Vue 2 project using Vue-CLI, use the [vue create](#) command. Follow these steps to install Vue CLI and create a new project:

```
`bash
npm install -g @vue/cli
vue create quickstart
cd quickstart
npm run serve
\
```

or

```
`bash
yarn global add @vue/cli
vue create quickstart
cd quickstart
yarn run serve
\
```

When creating a new project, choose the option **Default ([Vue 2] babel, eslint)** from the menu.



```
E:\samples>vue create quickstart

Vue CLI v5.0.8
? Please pick a preset:
  Default ([Vue 3] babel, eslint)
> Default ([Vue 2] babel, eslint)
  Manually select features
```

Once the **quickstart** project is set up with default settings, proceed to add Syncfusion components to the project.

### Add Syncfusion Vue packages

Syncfusion packages are available at [npmjs.com](https://www.npmjs.com). To use Vue components, install the required npm package.

This article uses the [Vue Floating action button component](#) as an example. Install the `@syncfusion/ej2-vue-buttons` package by running the following command:

```
`bash
```

```
npm install @syncfusion/ej2-vue-buttons --save
```

```
,
```

or

```
`bash
```

```
yarn add @syncfusion/ej2-vue-buttons
```

```
,
```

### Import Syncfusion CSS styles

You can import themes for the Syncfusion Vue component in various ways, such as using CSS or SASS styles from npm packages, CDN, [CRG](#) and [Theme Studio](#). Refer to [themes topic](#) to know more about built-in themes and different ways to refer to themes in a Vue project.

In this article, the `Material` theme is applied using CSS styles, which are available in installed packages. The necessary `Material` CSS styles for the Floating action button component and its dependents were imported into the `<style>` section of `src/App.vue` file.

#### ~/SRC/APP.VUE

```
<style>
@import '../node_modules/@syncfusion/ej2-base/styles/material.css';
@import '../node_modules/@syncfusion/ej2-buttons/styles/material.css';
</style>
```

### Add Syncfusion Vue component

Follow the below steps to add the Vue Floating action button component using `Composition API` or `Options API`:

1\ First, import and register the Floating action button component in the `script` section of the `src/App.vue` file. If you are using the `Composition API`, you should add the `setup` attribute to the `script` tag to indicate that Vue will be using the `Composition API`.

#### COMPOSITION API (~/SRC/APP.VUE)

```
<script setup>
import { FabComponent as EjsFab } from '@syncfusion/ej2-vue-buttons';
</script>
```

#### OPTIONS API (~/SRC/APP.VUE)

```
<script>
import { FabComponent } from '@syncfusion/ej2-vue-buttons';
export default {
```

```

components: {
  'ejs-fab': FabComponent
}
}
</script>

```

2\ In the **template** section, define the Floating action button component with the [content](#) property.

#### **COMPOSITION API (~SRC/APP.VUE)**

```

<template>
<div>
<div id="targetElement" style="position:relative;min-height:350px;border:1px
solid;"></div>
<!-- To render Floating Action Button -->
<ejs-fab id='fab' content='Add' target='#targetElement'></ejs-fab>
</div>
</template>

```

Here is the summarized code for the above steps in the **src/App.vue** file:

#### **COMPOSITION API (~SRC/APP.VUE)**

```

<template>
  <div>
    <div id="targetElement" style="position:relative;min-
height:350px;border:1px solid;"></div>
    <!-- To render Floating Action Button -->
    <ejs-fab id='fab' content='Add' target='#targetElement'></ejs-fab>
  </div>
</template>
<script setup>
  import { FabComponent as EjsFab } from "@syncfusion/ej2-vue-buttons";
  import { enableRipple } from '@syncfusion/ej2-base';
  enableRipple(true);
</script>
<style>
  @import '../node_modules/@syncfusion/ej2-base/styles/material.css';
  @import '../node_modules/@syncfusion/ej2-buttons/styles/material.css';
</style>

```

#### **OPTIONS API (~SRC/APP.VUE)**

```

<template>
  <div>
    <div id="targetElement" style="position:relative;min-
height:350px;border:1px solid;"></div>
    <!-- To render Floating Action Button -->
    <ejs-fab id='fab' content='Add' target='#targetElement'></ejs-fab>
  </div>
</template>
<script>
  import { FabComponent } from "@syncfusion/ej2-vue-buttons";
  import { enableRipple } from '@syncfusion/ej2-base';
  enableRipple(true);
  export default {

```

```

        components: {
            'ejs-fab': FabComponent
        }
    }
</script>
<style>
    @import '../node_modules/@syncfusion/ej2-base/styles/material.css';
    @import '../node_modules/@syncfusion/ej2-buttons/styles/material.css';
</style>

```

### Run the project

To run the project, use the following command:

```
`bash
```

```
npm run serve
```

```
,
```

or

```
`bash
```

```
yarn run serve
```

```
,
```

```
{% previewsample "page.domainurl/code-snippet/floating-action-button/getting-started-cs1" %}
```

### Click event

The floating action button control triggers the `onclick` event when you click on the floating action button. You can use this event to perform the required action.

### COMPOSITION API (~SRC/APP.VUE)

```

<template>
  <div>
    <div id="targetElement" style="position:relative;min-height:350px;border:1px solid;"></div>
    <!-- To render Floating Action Button -->
    <ejs-fab id='fab' iconCss='e-icons e-edit' content='Edit' v-on:click.native="onClick" target='#targetElement'>
      </ejs-fab>
    </div>
  </template>
<script setup>
import { FabComponent as EjsFab } from "@syncfusion/ej2-vue-buttons";
const onClick = function () {
  alert("Edit is clicked!");
};
</script>
<style>
  @import '../node_modules/@syncfusion/ej2-base/styles/material.css';
  @import '../node_modules/@syncfusion/ej2-buttons/styles/material.css';
</style>

```

### OPTIONS API (~SRC/APP.VUE)

```
<template>
  <div>
    <div id="targetElement" style="position:relative;min-
height:350px;border:1px solid;"></div>
    <!-- To render Floating Action Button -->
    <ejs-fab id='fab' iconCss='e-icons e-edit' content='Edit' v-
on:click.native="onClick" target='#targetElement'>
    </ejs-fab>
  </div>
</template>
<script>
import { FabComponent } from "@syncfusion/ej2-vue-buttons";
import { enableRipple } from '@syncfusion/ej2-base';
enableRipple(true);
export default {
  components: {
    'ejs-fab': FabComponent
  },
  methods: {
    onClick: function () {
      alert("Edit is clicked!");
    }
  }
}
</script>
<style>
  @import '../node_modules/@syncfusion/ej2-base/styles/material.css';
  @import '../node_modules/@syncfusion/ej2-buttons/styles/material.css';
</style>
```

{% previewsample "page.domainurl/code-snippet/floating-action-button/events-cs2" %}

## Getting Started with Syncfusion Floating Action Button Component in Vue 3

This article provides a step-by-step guide for setting up a [Vite](#) project with a JavaScript environment and integrating the Syncfusion Vue Floating Action Button component using the [Composition API](#) / [Options API](#).

The **Composition API** is a new feature introduced in Vue.js 3 that provides an alternative way to organize and reuse component logic. It allows developers to write components as functions that use smaller, reusable functions called composition functions to manage their properties and behavior.

The [Link to the Video](#) is the traditional way of writing Vue.js components, where the component logic is organized into a series of options that define the component's properties and behavior. These options include data, methods, computed properties, watchers, lifecycle hooks, and more.

To get started quickly with Vue 3 Floating Action Button, you can check out this video:

### Prerequisites

#### [System requirements for Syncfusion Vue UI components](#)

#### Set up the Vite project

A recommended approach for beginning with Vue is to scaffold a project using [Vite](#). To create a new Vite project, use one of the commands that are specific to either NPM or Yarn.

```
`bash
```



```
npm create vite@latest
```

```
,
```

```
or
```

```
`bash
```

```
yarn create vite
```

```
,
```

Using one of the above commands will lead you to set up additional configurations for the project as below:

1. Define the project name: We can specify the name of the project directly. Let's specify the name of the project as **my-project** for this article.

```
`bash
```

```
? Project name: » my-project
```

```
,
```

2. Select **Vue** as the framework. It will create a Vue 3 project.

```
`bash
```

```
? Select a framework: » - Use arrow-keys. Return to submit.
```

```
Vanilla
```

```
Vue
```

```
React
```

```
Preact
```

```
Lit
```

```
Svelte
```

```
Others
```

```
,
```

3. Choose **JavaScript** as the framework variant to build this Vite project using JavaScript and Vue.

```
`bash
```

```
? Select a variant: » - Use arrow-keys. Return to submit.
```

```
JavaScript
```

```
TypeScript
```

```
Customize with create-vue ↗
```

```
Nuxt ↗
```

```
,
```

4. Upon completing the aforementioned steps to create the **my-project**, run the following command to install its dependencies:

```
`bash
cd my-project
npm install
`
or
`bash
cd my-project
yarn install
`
```

Now that `my-project` is ready to run with default settings, let's add Syncfusion components to the project.

#### Add Syncfusion Vue packages

Syncfusion Vue component packages are available at [npmjs.com](https://www.npmjs.com). To use Syncfusion Vue components in the project, install the corresponding npm package.

This article uses the [Vue Floating Action Button component](#) as an example. To use the Vue Floating Action Button component in the project, the `@syncfusion/ej2-vue-buttons` package needs to be installed using the following command:

```
`bash
npm install @syncfusion/ej2-vue-buttons --save
`
or
`bash
yarn add @syncfusion/ej2-vue-buttons
`
```

#### Import Syncfusion CSS styles

You can import themes for the Syncfusion Vue component in various ways, such as using CSS or SASS styles from npm packages, CDN, [CRG](#) and [Theme Studio](#). Refer to [themes topic](#) to know more about built-in themes and different ways to refer to themes in a Vue project.

In this article, `Material` theme is applied using CSS styles, which are available in installed packages. The necessary `Material` CSS styles for the Button component and its dependents were imported into the `<style>` section of `src/App.vue` file.

#### ~/SRC/APP.VUE

```
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-buttons/styles/material.css";
</style>
```

The order of importing CSS styles should be in line with its dependency graph.

### Adding Syncfusion Vue Floating Action Button Component in the Application

Follow the below steps to add the Vue Floating Action Button component using **Composition API** or **Options API**:

1. First, import and register the Floating Action Button component in the **script** section of the **src/App.vue** file. If you are using the **Composition API**, you should add the **setup** attribute to the **script** tag to indicate that Vue will be using the **Composition API**.

#### COMPOSITION API (~SRC/APP.VUE)

```
<script setup>
import { FabComponent as EjsFab } from "@syncfusion/ej2-vue-buttons";
</script>
```

#### OPTIONS API (~SRC/APP.VUE)

```
<script>
import { FabComponent } from "@syncfusion/ej2-vue-buttons";
//Component registration
export default {
  name: "App",
  components: {
    "ejs-fab": FabComponent
  }
}
</script>
```

2. Add the component definition in **template** section.

#### ~/SRC/APP.VUE

```
<template>
<ejs-fab content='Add'></ejs-fab>
</template>
```

Here is the summarized code for the above steps in the **src/App.vue** file:

#### COMPOSITION API (~SRC/APP.VUE)

```
<template>
<div id="targetElement" style="position:relative;min-height:350px;min-width:
350px;border:1px solid;min-width: 300px;">
<ejs-fab id='fab' ref="fabIn" content='Add' target='#targetElement'></ejs-
fab>
</div>
</template>
<script setup>
import { FabComponent as EjsFab } from "@syncfusion/ej2-vue-buttons";
</script>
<style>
```

```
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-buttons/styles/material.css";
</style>
```

### **OPTIONS API (~SRC/APP.VUE)**

```
<template>
<div id="targetElement" style="position:relative;min-height:350px;border:1px
solid;">
<ejs-fab id='fab' content='Add' target='#targetElement'></ejs-fab>
</div>
</template>
<script>
import { FabComponent } from "@syncfusion/ej2-vue-buttons";
export default {
name: "App",
components: {
"ejs-fab": FabComponent
}
}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-buttons/styles/material.css";
</style>
```

### [Run the project](#)

To run the project, use the following command:

```
`bash
```

```
npm run dev
```

```
,
```

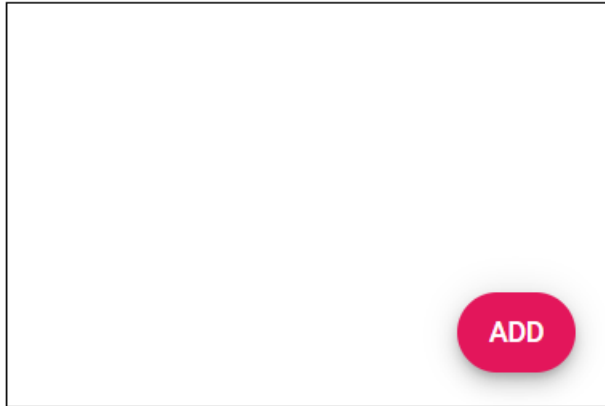
or

```
`bash
```

```
yarn run dev
```

```
,
```

The output will appear as follows:



For migrating from Vue 2 to Vue 3, refer to the [migration](#) documentation.

See also

- [Getting Started with Vue UI Components using Composition API and TypeScript](#)
- [Getting Started with Vue UI Components using Options API and TypeScript](#)

### Icons in Vue Floating action button component

You can customize the icon and text of Vue Floating Action Button(FAB) using [iconCss](#) and [content](#) properties.

#### FAB with icon

You can show icon only in Floating Action Button by setting [iconCss](#) property. You can show tooltip on hover to show additional details to end-user by setting [title](#) attribute.

```
<template>
<!-- To render Floating Action Button with icon only -->
<ejs-fab id='fab' iconCss='fab-icons fab-icon-people'></ejs-fab>
</template>
<script>
import Vue from 'vue';
import { FabPlugin } from "@syncfusion/ej2-vue-buttons";
Vue.use(FabPlugin);
export default {}
</script>
<style>
@import '../node_modules/@syncfusion/ej2-base/styles/material.css';
@import '../node_modules/@syncfusion/ej2-buttons/styles/material.css';
</style>
```

,

### FAB with icon and text

You can show icon along with text in Floating Action Button by setting [iconCss](#) and [content](#) properties.

,

```
<template>
<!-- To render Floating Action Button with icon and text -->
<ejs-fab id='fab' iconCss='fab-icons fab-icon-people' content='Contacts'></ejs-fab>
</template>
<script>
import Vue from 'vue';
import { FabPlugin } from "@syncfusion/ej2-vue-buttons";
Vue.use(FabPlugin);
export default {}
</script>
<style>
@import '../node_modules/@syncfusion/ej2-base/styles/material.css';
@import '../node_modules/@syncfusion/ej2-buttons/styles/material.css';
</style>
```

,

### Icon position

You can change the position of icon when showing along with content by setting [iconPosition](#) property. By default, the icon is positioned on the left side together with text.

,

```
<template>
<!-- To render Floating Action Button with icon position -->
<ejs-fab id='fab' iconCss='fab-icons fab-icon-people' content='Contacts' iconPosition='Right'></ejs-fab>
</template>
<script>
import Vue from 'vue';
import { FabPlugin } from "@syncfusion/ej2-vue-buttons";
Vue.use(FabPlugin);
export default {}
</script>
<style>
```

```
@import '../node_modules/@syncfusion/ej2-base/styles/material.css';
@import '../node_modules/@syncfusion/ej2-buttons/styles/material.css';
</style>
```

Below example demonstrates a FAB with icon and text.

**APP.VUE**

```
<template>  
  <div>  
    <div id="targetElement" style="position:relative;min-height:350px;border:1px solid;"></div>  
    <!-- To render Floating Action Button with icon and text -->  
    <ejs-fab id='fab' iconCss='fab-icons fab-icon-people'  
content='Contacts' iconPosition='Right' target='#targetElement'></ejs-fab>  
  </div>  
</template>  
  
<script>  
import Vue from 'vue';  
import { FabPlugin } from "@syncfusion/ej2-vue-buttons";  
import { enableRipple } from '@syncfusion/ej2-base';  
enableRipple(true);  
Vue.use(FabPlugin);  
export default {}  
</script>  
  
<style>  
@import '../node_modules/@syncfusion/ej2-base/styles/material.css';  
@import '../node_modules/@syncfusion/ej2-buttons/styles/material.css';  
@font-face {  
  font-family: 'fab-icons';  
  src: url(data:application/x-font-ttf;charset=utf-8;base64,AEEAAAAKAIAAAwAgTlMvMjl1tSgIAAAEoAAAAMVnTYXDnYOfNAAABYAAAAFZnbHlmkfZLRAAAAKGAAABoaGVhZCF5f3EAAADQAAAAANmhoZWElUQQTA AAAArAAAACRobXR4SAAAAAAAAAYAAAAIBzG9jYTvenNR4AAAIgAAAAJmlheHABIQIXAAABC AAAACBuYW1lkWSegQAAGTAAAAIxcG9zdKKfPWYAABtkAAAAzQABAAAEAAAAAFWEAAAAAAD9AABAABAAAAA AAAAAAAAAAAAAEAABAAAAAQAbzQqW18PPPUACwQAAAAAN8znUAAAAAAN3zOdQAAAAAAD9AP0AAAAACA AAAAAAAAAAAAAEAAAAASAgABQAAAAAAGAAAAAoCgAAAP8AAAAAAAAAAQQA ZAABQAAAokCzAAAAII8CiQLMAAAB6wAyAQgAAAAIABQM AAAAAA AAAAAAAAAAAAAAAAAAAAAUGZFZA B5wDnEAQAAAAAXAQAAAAAAAABAAAAAABAAAAAQAAAA EAAAAABAAAAAQAAAAEAAAAABAAAAAQAAAAEAAAAABAAAAAQAAAAEAAAAABAAAAAQAAAAEAAAAA AgAAAAAUA MAUAQAABQABABCAAAABAAEAAEOcQ//8AAOcA//8AAAABAAQAAAABA IA AwAEA UABgAHAAgACQAKAAsADA ANAA4ADwAQABEAAAAAAAAABP APMBFQEggSaBK4FQAX sBrgGL1afcCB AI5gj8CR4KPA t0AAAABAAAAAAD9APoAD0AwC5AXAA ACUF CRUPDiMvDjU/Dh8CAR8LFQ8OIy8PPw4XAR8LHQEPDI8OPQE/DDMXJw8KHQEFagUVcSSBDw4VHW4zPwcBDWedAR8NPw41Lw4 jDwkBNzUvASufCTM/DT0BLw4rAQ8BA5AGBgYFB AQDA gIBA QECawMFBAUGBgCHBWgHCacIBwCHBG YGBQQEA wICA QEBAgMDB QQGBQYHBwCKwoJCv0nBwcGBgYFB AQDA wIBA QECagQBQUFBGChBwCHCAC ICACHBgYGBQQEA wMBAQEBAQIDA wQEBgUGBwKICQoLatwHBwCGBUFBAMCagEBAgMDBAUFBG yGBwg HBWG IBwC IBwYGBQUFBAMCagEBAgMDBAUFBQC GCQk JCQxMDAsKCQGHBgUDAgEBAQP+HwYHCagICQk KDQ4ODQ4NDAwMCwoJCACHBAQCAQIDBQYHCagKCwwMDQoLCgoREBAPdg0MCgHDagICagQGBGGJCQSMEBESERMSDQwMCwsJCAGGBQQCAQIDBAYGCakJCwwMDQ0NDQ4NDgoKCQkJCAcH/h4CAQMB4gcOBwgJCQkKCwoLEBAQEAsLCwkIBWyFAwMcAUFBwgJCgsLDA0KCgoKEREQ2QQEBQYGBgYHBwCHBwCHBWyHBQFUFBAMDAG IBAQE CAgQEBAU FBG YHBwCHBwCHBwyHBgYFB AQEA WICAQE CAWFvAgMEBAUGBQC GBWC HBwCHBwCHBwYFBQUEAWICAQEBAgidBAQFBQYGBwCHBwCHBwYHBgYGBQUEBAMDAQEBAV0DAwQE BQU GBgCHBGGHBwCHBGCGBGUEBQMDAwEBAQEBAgieAU EBgyGBgCHBwCHBwYFBYFBQUEBAMDAGEsbwCICQoLCwwKCQkKCQkJCeOHBGYFBQUEAWMDAgMEBQC HCAk LCgw NDA0NDA0MDAwLCWKJBwCGBAMCAOE
```

CBQYICQsM/v8IERAJCAGNCwsLCQkIBwgFAwEBawQFBggICQoLDAwMDA0NDA0MDAsLCQkIBwYEBAl  
BAgIDBAQFBgYHBwEDDg0NDewHDAUFBAQDAwIBAQMEBwYICakKCwwMDA0NDA0MDAwLCwkJCAYGBAM  
CAQMEAAAABQAAAAAD7wP0AD8AfwEQAVACCgAAARUPDSsBLw4/DjsBHW0FFQ8NkWEvDT0BPw07AR8  
NAR8QFQcvAiMPDx8HDwcjLwc/By8PDwMvAj8QHw8/DycVDw0rAS8NPQE/DTsBHW0nDxUdAR8DDwg  
fDjsBPwIfDjsBPw4fAjSBPw4vCD8CPQEvLSMPDgOxAgIDBAUFBwYHCAgJCQkKCQoJCAgIBwcGBgQ  
EAwMBAQEBAwMEBAYGBwcICAgJCgkKCQkICAgHbGcFBQQDAgL9WQEDAwQFBQYHBwgICakKCQoJCQk  
ICaCHBgUFBAMCAgICAwQFBQYHBwgICQkJCgkKCQgICaCHBgUFBAMDAQGEDg0aGRcWFBMSDw4MCQg  
GAwEDDAwMDBAPDw4ODQwLCgkIBwUEAgEBAGMFBGcIDRAaGhwCHR4eHh4dHR0bGxoQDQgHBgUDAgE  
BAgQFBwgJCgsMDQ4ODw8QDAwMDAECAQEDBQcJCgwODxETExULFkQBgyGBggICQkKCgsLDAwMDA4  
NDQwMDAsKCggJBwcFBQ6AgIDBAUFBgCHCAgICQoJCgkJCQgIBwcGBQUEAwICAgIDBAUFBgCHCAg  
JCQkKCQoJCAgIBwcGBQUEAwIC+AEXDw8ODhoZGBYUEhAOcwoEAwMCAgEBAGMDBgoJCAyFBAMBAQM  
EBQYICQoLDA0ODg8QEA4NDg8PEBAQERIREhISExMSFBMTExITEhITERIREREREBAPFg8ODQ4QEA8  
ODg0MCwoJCAyFBAMBAQMEBQYICQoGBQICAgIDAwQEBQYGBwgICQkKCgsMDAwNDQ4ODw8PEBAQGGQ  
EBQCHCAkKCwsLDQ0NDQ4PDg4NDQwLCwkJCQcGBQQAoJCQkICaCHBgUFBAMCAgICAwQFBQYHBwg  
ICQkJCgkKCQgICaCHBgUFBAMDAQEDAwQFBQYHBwgICakKCQoJCQkICaCHBgUFBAMCAgICAwQFBQY  
HBwgICQkJCgkKCQgICaCHBgUFBAMDAQEDAwQFBQYHBwgICakKAigGBg4PERMUFhYZGRobHBwdHRs  
bAwICAQIEBQcICQoLDA0NDw8PEA8PDg0NDQsPCw8NCwkHBAICBAcJCwwPCxALDQ0NDg8PEA8PDw0  
NDAsKCQgHBQQCAQEBAGMKFhYbGxsaGhkYFxyVFBMRCA8OBxALCgkJCQgHBgyFBAMDAQEBAGIEBQY  
GCAGJCgoLCwwOJgoJCQkICaCHBgUFBAMCAgICAwQFBQYHBwgICQkJCgkKCQgICaCHBgUFBAMDAQE  
DAwQFBQYHBwgICakKDgcKBwgICRMVFXgZGxwdHx8PEBEQEBEQExMTEwwFCwwNDg4PDxAQE8ODg0  
MCwoJCAyFBAMCAwUMCwoJCQgHBwYFBAQDAgEBAGMDBAUGBgICakJCxAFawIDBAUHBwkKDAwNDQ8  
OEBAQDw8PDQ0MCwUYFRUWEhESERERERARE8QDw8ODg4NDQ0MDAsKCgoJCAgIBgkQDQwMCwoKCAg  
HBwUEAwIBAGMEBgYICaOKCwsNDA4AAAMAAAAA/QD9AAHACoAUAAACQEVmWERIRElMx8ODwcBBzU  
BPwUnByERIRE/CDUvDg8GApn+D8MB7fzyAyMIBwcGBwYFBgyEAwMCAQEBAQEBAgMDBAX9rmkCVwY  
GBwCHB1o1/TUDiJkJBwYGBAMBAQIDAwUGCAGMCwsMDQ0ODg0ODQ0MCwsDWP4QwAhz/a8DD10BAgI  
DBAQFBgyYFBGcGBGcGBwYGBgyYFBv2oAWgCVgUEAwICARY1/HYCzjoKCgsLDAwMDAwMDAwLCwoKDAg  
HBgUDAgEBAGMFBGcIAAADAAAAAAP0A/QAAGAGABkaADclJzcXASc3Bxc/AzUvBw8CDAEk6jvpAdP  
qqW7pcgUEAgIEBaYICQkKCQkJDDrQOukB0umpbulYCaKKCQoJCKYHBAMBAQMEAAAAAAEAAAAA/Q  
D9AALAAABIRUhETMRITUhESMBwv5KAbZ8Abb+SncPnz+SgG2fAG2AAEAAAAA/MDDwAFAAATFwk  
BNwEMLAHKAcYs/g4BHSwBxv48LQHvAAABAAAAAAP0A/QAgQAAEW8TFR8cPwx1LxMPDS8NB+0JCRM  
TEhMREhAQDg0LCgQEAwIDAQICAwMFBQYGCAGKCsMDg4PEBETExUWFxgZNzxBQTW3GRgXFhUTEExE  
QDw4ODAsKCggIBgyYFBQMDAgIBAwIDBAQKcW0OEBASERMSExMfFx8TEhEQDw4ODQwMDBcXfxcMDAw  
NDg4PEBESEyAXA/EBAGUGCQsMDxETFBcZGw4PDxAQETsQERARERISEhISExMUExQUFRQVfHwUfHc  
XFy8wMjIwLxcXFxyWFRYVFBuUFBMUExMSEhISEhEREBEQOxEQE8PDhsZFxQTEQ8MCwkGBQUBAQE  
DBAUGBwgJCQsLGRwcGQsLCQkIBwYFBAMBAQEAAAAAAGAAAAADOWP0AEAA1AAAA0BDw4vDz8Phw4  
FFxUfBgEbAT8HNS8dKwEPHQKiAwQFBwgJCwsNDQ4PEBARERAQDw4NDQsLCQgHBQQDAQEDBAUHCAk  
LCw0NDg8QEBEQEA8PDg4MDAsJCAcGBAP+IgEDBAYHCAoLAQqJhAoJCAcFAGQBAQIDAwQFBgyHCAg  
JCQoLCgWMDA0NDg4ODw8PEBAQEBAQEBAQDw8ODg4NDQwMCwsLCgkICAgHbGyYFBAMDAgECvAkIDw8  
PDgWMCwoJCAcFBAIBAQIEBQcICQoLDAwODw8PERAPDw8ODAwLCgkIBwUEAgEBAGQGBggKCgsMDQ0  
PDw8GCwsVFRQUEXMR/eYBEAEKERMTFBQKFRYbDw8PDw8ODQ4NDQwMCwsLCgkJCQcIBgyGBAUDAwE  
CAgEDAwUEBgYGCACJCQkKCwsLDAwNDQ4NDg8PDw8PAAACAAAAAOTA/QAdAc2AAAlFSMVITUjNT8  
eNSMPFS8VixUfHQMRHw8/DxEvDw8OAdGJAXeKEhISEhEREBAPEA4ODg0NDAsLCwkJCQcHBgyEBAM  
CAlcAgMEBAUFBgCHCAgSFRYXGhocHh4fHx4eHBoaFxyVEggIBwcGBQUEBAMCAlcAgMEBAYGBwg  
ICQoKCwwMDQ0ODw4QEBARERESExJ2AQIFBgkCgWNDw8QERESExMSEREQDw8NDAoKBACFBAIBAgU  
GCAkKDA0OEBARERMTExMSERAPDw0MCwkHBgUCuFtRUVsCAwMEBQUGBgICAgJCgoKCwsMDAwNDQ4  
NDg8ODw8PEA0MCwwLDAsKCwoKCRIRDw0MCgcGBAEBBAYHCgWNDxESCQoKCwoLDAsMCwwNEA8PDw4  
PDg0ODQ0MDAwLCwoKCgkICAgHbgyYFBQQDAwKk/rGQDw4ODQ0LCwkJBwcEBAIBAQIEBQYICQkLDAw  
HDg4UFAYUHAwGJ/oMBNXYBNQE1dgE1/on3AnXq/opQ6gF96gF9AAAAAIAAAAAA/QDvQB3AO4AAAE  
VHxMPFSEvFT8VLw8PDicfFA8VIT8KLw8/Dy80Iw8OAcSAwQFBgyICAOKCgWMDQ8GBgICAgEBAGI  
DAwQeNRkZFxcVFRMSEA8ODAOC8QoMDg8QEHMVFRcXGRkaGxsDBwcAgEBAQMDAwQUDQwLCwoKCAg  
GBgUEAwEBAQMEBgJCgsNDQ4QEBEREhISEhAQDg4NCwoJBwYFAvgBAQMEBQYGCAGKCsLDA0QBgu  
DAQEBAQICAwMEHjUZGRcXFRUTEhAPDgWKAQMMDAwNDg4dHh8gIQ0MCwoJCQgHBwUFBAMCAQEBAgM  
FBgyICQoLCw0NDg4PBgcHCAgJCQkKCgsLCwsLDBIRERAQDg0NCwoJBwYEAwJWEXISERAPDg4NDAw  
KCgkICQYHBQQFBAUEBAQDAgwXCw0NDg4QERIUFRIYGHkYFxyUTEhIQDw4NDQsLCwoCBQgEBQQGBQU  
FAwQLCAkKCwsMDQ0PDw8RERISFhYVFRMSEQ8ODQsJBwUEAQEEBQcKCg0OEBASFBQVfncTExERE8  
PDQ0MDAoKCQkJBQcFBAUFBAUEAwMDDBYMDA0ODw8REhQVfHkaDQ0MDAsKFBMQEA8KDAwMDQ4ODhA



```

PEBARERISFRQUBMSEREQDw8NDAsKCQwLCgoJCQgIBgYFBQMCAGEDBQgJCwwOEBESExQVFgAAAAE
AAAAAAvAD9AAkAAABERSBES8PIw8OAQ/t9QECawUFBwgICgoLDAwNDQ7WDg0NDawLCgoICacFBQM
CA238nwEW/uoDYQ4NDQ0LCwsJCQgHBQUDAgEBAgMFBQcICQkLCwsNDQ0AAAAAAwAAAAAD9AP0AD8
AfwC1AAALHw8/Dy8PDw4FHw8/Dy8PDw4DMxMPAhUfDiE1IS8ENyE/BhM/AjUvBiEnIwLIAQECBAQ
FBQCHCAGJCQoKCgoKCgkJCAGHBwUFBAQCAQEBAQIEBAUFBwcICakJCgoKCgoKCQkICacGBgUEBAI
B/gSBAQIEBAUFBwcICakJCgoKCgoKCQkICacHBQUEBAIBAQEBAgQEBQUHBwgICQkKCgoKCgoJCQg
IBwYGBQQEAgHJZLVICAECAGQEBQUHBwgICQkJCgsCW/27AwMCAQEsAXEPDQwMCggIvAMEAgIDBQc
ICQr9FSumcAoKCgkJCAGHBgYFBQCAQEBAQIEBAUFBwcICakJCgoKCgoKCQkICacHBQUEBAIBAQE
BAgQEBQUHBwgICQkKCgoKCgoJCQgIBwYGBQQEAgEBAQECBAQFBQCHCAGJCQoKCgoKCgkJCAGHBwU
FBAQCAQEBAQIEBAUFBwcICakJCgoDFv6DdyYMCgoKCQkICacGBgUEBAIBAQBQMCCVQBAGUFBwk
KAVADBwUQCgkIBwUDAmQAAAAACAAAAA02A/QAAGAFAAAJAQsBCQEDRf0+ATcDbPySAhL+XQMm/Hc
CCAHgAAQAAAAAAygd9AADAACACwAPAAABESMIREjEQEzESMBMxEjAu5e/uBeAUPT0/6D09MDUfy
OA3L8jgNy/FMD6PwYA+gAAwAAAAADuAP0ABEAZgD6AAALDwgvBxMzHwcVMx8SDwIfCCU/CS8CPxE
zNT8HBxUPFBuFag8NFwUfDjsBPw4FNy8MPwI1LyUjDw4CHQEBAgMEBAUFBUFBQDQDawICKAQECAY
GBQQBAjQLFBAQERERCAgJCAcFBQQDAGEBAG0CAGICBAUGBwkL/ZUMCQcGBQQCAGEBAG0BAQECAwQ
FBQcICQcQEBAQDw4WOAECBAUGAwcIYh8UFBYLCwsLCwoKDQkIBwUFAwMBAQIGBwEDAwMEBAYGBxA
PDhIRBQFXAQICBAQEBgYGBwgHCQgJCQkJCAGIBwCGBGUFBAQDAGEBWQUXEW8PDwYGBAQEAgICAQw
CAQICBAQGBggJCw0LCwsMCwwLFhUTEwEBAwMFBQYGCACJCQkKCgsYCgoKCQgJBwgGBGUFBAICYwE
FBAQEAWIBAQEBAgMEBAQGA1IBAgUFBwgECUMDCACkDA8RCgsSERERERARERAhIqc4HBIQEA4ODQ0
OAQ8NDQ4OEBASEhUtpyIhEBEQERARERESChIQDQoJBwg/CQgIBwUDAwIoBQoICw4ICQoLDA0OFxQ
UFBMTExMSEyUnRm0qGwwKCgkICacLCQYHBT4BCQkICAGHBgcFBQUDAwICAgICBAQFBQYGBwCICAg
NAT4HCACkDQgICQoKDA0dL58nJRMSExMTExQUFBQSDQwMCgkICAwJBwYMCgsKCQkJBwCGBGQEAWE
BAQEDAwQGBQCHCAkJCQoKAAACAAAAAAP0A/QAQAEMAAABFQ8PLw8/Dx8OARUPBS8EDwcdAR8DDwY
rAQ8FHQEfBjsBHWUPBB8IPwQfBh0BHWU7Aj8FPQE/BR8EPwgvBD8GOWE/BT0BLwYrAS8FPwQvBys
BDwMvBj0BLwUrAg8FAsGBAwUHCAoMDQ4QEBISExQUFBQTEhIQEA4NDAoIBwUDAQEDBQcICgWNDhA
QEHITFBQUFBMSEhAQDg0MCggHBQP+6A4cHA0ODVoDBAQFBAUDA1ICAgICAgJbCA4GBGUEBH0GBQQ
DAWICAQEDAwQEBAV9BAkMBwCIWwMCAQEBAQIDUGIEBQQFBAQDWg0aDg0ODw8CagQDBQQFcAYFBAQ
CAwEOHBwNDg1aAwQEBQQFBAJSAGIBAQEBAgJbCA4GBGUEBH0GBQQDAWICAQEDAwQEBAV9BAkMBwC
IWwMCAQEBAQIDUQMEBAUFBAQDWg0aDg0ODw8CagMEBAUFcAUEBQMEAgICAAoKFBMSEhAQDg0MCgg
HBwBQAQMFbWgKDA0SEBASEhMUFBQUExISEBAODQwKCACFAwEBawUHCAoMDQ4QEBISExQBx30ECQw
HBwhbAgIBAQEBAgJSAgQEBQEBANXDRoODQ4PDwICAwQEBQVwBgUEAwMCAg4cHA0ND1oEBAUEBQQ
FA1ICAgYGBQECALsIDgYGBQQEfQYFBAQMDAgICAwMDBQQFfQJDAcHCFsCAGEBQECaK4DBAQFBQQ
EA1oNGg0ODg8PAgIDBAQFBHEGBAUDAwICDhwcDQ0NwWQEBQQFBAUDUGICAgICALsIDgYGBQQEfQY
FBAMDAgICAgMDBQQAaaaaaBIA3gABAAAAAaaaaaEAAAAAaaaaaAABAgAAQABAAAAAACAACACQA
BAAAAAADAAgAEAAABAAAAAEEAgAGAABAAAAAFAAAsAIAABAAAAAAGAAgAKwABAAAAAAKACw
AMwABAAAAAALABIAXwADAAEECQAAAAIAcQADAAEECQABABAAcWADAAEECQACAA4AgwADAAEECQA
DABAAKQADAAEECQAEABAAoQADAAEECQAFABYAsQADAAEECQAGABAAxwADAAEECQAKAFgAlwADAAE
ECQALACQBLyBzYilpY29uc1JlZ3VsYXJzYilpY29uc3NiLWljY25zVmVyc2lubiAxLjBzYilpY29
uc0ZvbnQgZ2VuZXJhdGVkIHVzaW5nIFN5bmNmdXNpb24gTWV0cm8gU3R1ZGlvd3d3LnN5bmNmdXN
pb24uY29tACAACwBiAC0AaQBJAG8AbgBzAFIAZQBnAHUAbABhAHIAcWBiAC0AaQBJAG8AbgBzAHM
AYgAtAGkAYwBvAG4AcwBWAGUAcgBzAGkAbwBuACAAMQAuADAACwBiAC0AaQBJAG8AbgBzAEYAbwB
uAHQAIAbNAGUAbgBIAHIAyQB0AGUAZAAgAHUAcWBPAG4AZwAgAFMAEQBuAGMAZgB1AHMAAQBVAG4
AIABNAGUAdABYAG8AIABTahQAdQBkAGkAbwB3AHcAdwAuAHMAEQBuAGMAZgB1AHMAAQBVAG4ALgB
jAG8AbQAAAAACAAAAAaaaaaAaaaaaAaaaaaAaaaaaAaaaaaAaaaaaAaaaaaAaaaaaAaaaaaAaaaaa
JAQoBCwEMAQ0BDgEPARABEQESARMABXNoYXJlCHNoYXJlLTaxBGVkaXQHZWRpdC0wMQNhZGQhdxB
hcnJvdwVoZWZyYdAntYXAMdm9pY2Utc2VhcmNoCWZhdmd9yaXRlcwtjaGF0LXB1cnNvbGhib29rbWF
yawhzaG9wcGluZwRwbGF5BxBhdXNlCHJlbWluZGVyCHNldHRpbmdzAAAAA=)
format('truetype');
    font-weight: normal;
    font-style: normal;
}
[class^="fab-icon-"],
[class*=" fab-icon-"] {
    font-family: 'fab-icons' !important;
    speak: none;
    font-size: 55px;
    font-style: normal;

```

```

        font-weight: normal;
        font-variant: normal;
        text-transform: none;
        line-height: 1;
        -webkit-font-smoothing: antialiased;
        -moz-osx-font-smoothing: grayscale;
    }
    .fab-icon-people:before {
        content: "\e70a";
    }
</style>

```

{% previewsample "page.domainurl/code-snippet/floating-action-button/icons-cs1" %}

### Styles in Vue Floating action button component

This section explains the different styles of Floating Action Button.

#### FAB styles

The Vue Floating Action Button supports the following predefined styles that can be defined using the [cssClass](#) property. You can customize by replacing the `cssClass` property with the below defined class.

cssClass	Description
-----	-----
e-primary	Used to represent a primary action.
e-outline	Used to represent an appearance of button with outline.
e-info	Used to represent an informative action.
e-success	Used to represent a positive action.
e-warning	Used to represent an action with caution.
e-danger	Used to represent a negative action.

#### APP.VUE

```

<template>
  <div>
    <div id="targetElement" style="position:relative;min-height:350px;border:1px solid;"></div>
    <!-- To render Floating Action Button with applied warning style -->
    <ejs-fab id='fab' iconCss= 'e-icons e-edit' cssClass= 'e-warning'
target='#targetElement'></ejs-fab>
  </div>
</template>
<script>
  import Vue from 'vue';
  import { FabPlugin } from "@syncfusion/ej2-vue-buttons";
  import { enableRipple } from '@syncfusion/ej2-base';
  enableRipple(true);
  Vue.use(FabPlugin);
  export default {}
</script>
<style>
  @import '../node_modules/@syncfusion/ej2-base/styles/material.css';

```

```
@import '../node_modules/@syncfusion/ej2-buttons/styles/material.css';
</style>
```

```
{% previewsample "page.domainurl/code-snippet/floating-action-button/styles-cs1" %}
```

Predefined Floating Action Button styles provide only the visual indication. So, Floating Action Button [content](#) property should define the Floating Action Button style for the users of assistive technologies such as screen readers.

### Styles customization

To modify the Floating Action Button appearance, you need to override the default CSS of Floating Action Button component. Please find the list of CSS classes and its corresponding section in Floating Action Button component. Also, you have an option to create your own custom theme for the components using our [Theme Studio](#).

CSS Class	Purpose of Class
.e-fab.e-btn	To customize the FAB.
.e-fab.e-btn:hover	To customize the FAB on hover.
.e-fab.e-btn:focus	To customize the FAB on focus.
.e-fab.e-btn:active	To customize the FAB on active.
.e-fab.e-btn-icon	To customize the style of FAB icon.

### Show text on hover

By using [cssClass](#), you can customize the Floating Action Button to show text on hover with applied transition effect. For detailed information, refer `app.vue` file below.

The content will behave the same, when the `enableHtmlSanitizer` is enabled. Since we are adding only the valid tags in content, sanitizing the content will not affect it.

### APP.VUE

```
<template>
  <div>
    <div id="targetElement" style="position:relative;min-
height:350px;border:1px solid;"></div>
    <!-- To render Floating Action Button -->
    <ejs-fab id='fab' iconCss='e-icons e-edit' content='<span
class="text-container"><span class="textEle">Edit</span></span>'
cssClass='fab-hover' target='#targetElement'></ejs-fab>
  </div>
</template>
<script>
  import Vue from 'vue';
  import { FabPlugin } from "@syncfusion/ej2-vue-buttons";
  import { enableRipple } from '@syncfusion/ej2-base';
  enableRipple(true);
  Vue.use(FabPlugin);
  export default {}
</script>
<style>
```

```

@import '../node_modules/@syncfusion/ej2-base/styles/material.css';
@import '../node_modules/@syncfusion/ej2-buttons/styles/material.css';
/* start of onhover customization */
.e-fab.e-btn.fab-hover {
    padding: 6px 0px 10px 10px;
}
.fab-hover .text-container {
    overflow: hidden;
    width: 0;
    margin: 0;
    transition: width .5s linear 0s, margin .2s linear .5s;
}
.fab-hover:hover .text-container {
    width: 35px;
    margin: 0 5px;
    transition: width .5s linear .2s, margin .2s linear 0s;
}
/* end of onhover customization */
</style>

```

{% previewsample "page.domainurl/code-snippet/floating-action-button/styles-cs2" %}

### Positions in Vue Floating action button component

The floating action button can be positioned anywhere on the [target](#) using the [position](#) property. If the [target](#) is not defined, then FAB is positioned based on the browser viewport.

The position values of Floating Action Button are as follows:

- TopLeft
- TopCenter
- TopRight
- MiddleLeft
- MiddleCenter
- MiddleRight
- BottomLeft
- BottomCenter
- BottomRight

<template>

<!-- To render Floating Action Button in BottomLeft position -->

<ejs-fab id='fab' content='Add' position='BottomLeft'></ejs-fab>

</template>

<script>

import Vue from 'vue';

import { FabPlugin } from "@syncfusion/ej2-vue-buttons";

import { enableRipple } from '@syncfusion/ej2-base';

```

enableRipple(true);
Vue.use(FabPlugin);
export default {}
</script>

<style>
@import '../node_modules/@syncfusion/ej2-base/styles/material.css';
@import '../node_modules/@syncfusion/ej2-buttons/styles/material.css';
</style>
`

```

Below example demonstrates different supported positions of FAB.

### APP.VUE

```

<template>
  <div>
    <div id="target" style="position:relative;min-height:350px;border:1px solid;"></div>
    <!-- To render Floating Action Button in TopLeft position -->
    <ejs-fab id="fab1" iconCss='fab-icons fab-icon-people'
position='TopLeft' target='#target'></ejs-fab>
    <!-- To render Floating Action Button in TopCenter position -->
    <ejs-fab id="fab2" iconCss='fab-icons fab-icon-people'
position='TopCenter' target='#target'></ejs-fab>
    <!-- To render Floating Action Button in TopRight position -->
    <ejs-fab id="fab3" iconCss='fab-icons fab-icon-people'
position='TopRight' target='#target'></ejs-fab>
    <!-- To render Floating Action Button in MiddleLeft position -->
    <ejs-fab id="fab4" iconCss='fab-icons fab-icon-people'
position='MiddleLeft' target='#target'></ejs-fab>
    <!-- To render Floating Action Button in MiddleCenter position -->
    <ejs-fab id="fab5" iconCss='fab-icons fab-icon-people'
position='MiddleCenter' target='#target'></ejs-fab>
    <!-- To render Floating Action Button in MiddleRight position -->
    <ejs-fab id="fab6" iconCss='fab-icons fab-icon-people'
position='MiddleRight' target='#target'></ejs-fab>
    <!-- To render Floating Action Button in BottomLeft position -->
    <ejs-fab id="fab7" iconCss='fab-icons fab-icon-people'
position='BottomLeft' target='#target'></ejs-fab>
    <!-- To render Floating Action Button in BottomCenter position -->
    <ejs-fab id="fab8" iconCss='fab-icons fab-icon-people'
position='BottomCenter' target='#target'></ejs-fab>
    <!-- To render Floating Action Button in BottomRight position -->
    <ejs-fab id="fab9" iconCss='fab-icons fab-icon-people'
position='BottomRight' target='#target'></ejs-fab>
  </div>
</template>
<script>
  import Vue from 'vue';
  import { FabPlugin } from "@syncfusion/ej2-vue-buttons";
  import { enableRipple } from '@syncfusion/ej2-base';
  enableRipple(true);

```

2882

MCwoJCAYFBAMCAwUMCwoJCQgHBwYFBAQDAGEBAGMDBAUGBgCICakJCxAFawIDBAUHBwkKDAwNDQ8  
OEBAQDw8PDQ0MCwUYFRUWEhESERERERAREA8QDw8ODg4NDQ0MDAsKCgoJCAgIBgkQDQwMCwoKCAg  
HBwUEAwIBAgMEBgYICAoKCwsNDA4AAAMAAAAA/QD9AAHACoAUAACQEVmWERIRElMx8ODwcBBzU  
BPwUnByERIRE/CDUvDg8GApn+D8MB7fzyAyMIBwCGBwYFBgYEAwMCAQEBAQEBAgMDBAX9rmkCVwY  
GBwCHB1o1/TUDiJkJBwYGBAMBAQIDAwUGCAGMCwsMDQ0ODg0ODQ0MCwsDWP4QwAHZ/a8DD10BAgI  
DBAQFBgYFBGcGBGcGBwYGBgYFBv2oAWgCVgUEAwICARY1/HYCZjoKCgsLDAwMDAwMDAwLcwoKDAg  
HBgUDAgEBAGMFBGcIAAADAAAAAAP0A/QAAGAGABkAADclJzcXASc3Bxc/AzUvBw8CDAEk6jvpAdP  
qqW7pcgUEAgIEBaYICQkKCQkJDDRqOukB0umpbulYCaKkCQoJCKYHBAMBAQMEAAAAAEEAAAAA/Q  
D9AALAAABIRUhETMRITUheSMBwv5KAbZ8Abb+SncPnz+SgG2fAG2AAEAAAAA/MDDwAFAAATFwk  
BNwEMLAHKAcYs/g4BHSwBxv48LQHvAAABAAAAAAP0A/QAGQAAEW8TFR8cPwx1LxMPDS8NB+0JCRM  
TEhMREhAQDg0LCgQEAWIDAQICAwMFBQYGCAGKCgsMDg4PEBETExUWFxgZNzxBQTW3GRgXFhUTEExE  
QDw4ODAsKCGgIBgYFBQMDAgIBAwIDBAQKcW0OEBASERMSExmFfx8TEhEQDw4ODQwMDBcXFxcMDAw  
NDg4PEBESEYAXA/EBAGUGCQsMDxETFBcZGw4PDxOaQETsQERARERISEhISExmUEXQUFRQVfHwUWFhc  
XFy8wMjIwLxcXFxYWFYVFBuUFBMUEXmSEhISEhEREBEQOxEQEAE8PDhsZFxQTEQ8MCwkGBQUBAQE  
DBAUGBwgJCQsLGRwcQsLCQkIBwYFBAMBAQEAAAAAagAAAAADowP0AEAA1AAAA0BDw4vDz8Phw4  
FFxUfBgEbAT8HNSdKwEPHQKiAwQFBwgJCwsNDQ4PEBARERAQDw4NDQsLCQgHBQDQAQEDBAUHCAk  
LCw0NDg8QEBEQEA8PDg4MDAsJCACGBAP+IgEDBAYHCAoLAQqJhAoJCACFAGQBAQIDAwQFBgYHCAG  
JCQoLCgWMDA0NDg4ODw8PEBAQEBAQEBAQDw8ODg4NDQwMCwsLCgkJCAGHBgYFBAMDAgECvAkIDw8  
PDgWMCwoJCACFBAIBAQIEBQcICQoLDAwODw8PERAPDw8ODAwLCgkIBwUEAgEBAGQGBggKCgsMDQ0  
PDw8GCwsVFRQUEXMR/eYBEAEKERMTFBQKFRYbDw8PDw8ODQ4NDQwMCwsLCgkJCQcIBgYGBAUDAwE  
CAGEDAwUEBgYGCACJCQkKCwsLDAwNDQ4NDg8PDw8PAAACAAAAAOTA/QAdAC2AAAlFSMVITUjNT8  
eNSMPFS8VixUfHQMRHw8/DxEvDw8OAdGJAXeKEhISEhEREBAPEA4ODg0NDAsLCwkJCQcHBgYEBAM  
CA1cCAGMEBAUFBgCHCAGSFRYXGhocHh4fHx4eHBoaFxFYVEggIBwCGBQUEBAMCA1cCAGMEBAYGBwg  
ICQoKCwWMDQ0ODw4QEBARERESExJ2AQIFBgkCgWNDw8QERESExMSEREQDw8NDAoKBACFBAIBAgu  
GCAkKDA0OEBARERMTExMSERAPDw0MCwkHBgUCuFtRUVsCAwMEBQUGBgCICAgJCgoKCwsMDAwNDQ4  
NDg8ODw8PEA0MCwWLDAsKCwoKCRIRDw0MCgcGBAEBBAYHCgWNDxESCQoKCwoLDAsMCwWNEA8PDw4  
PDg0ODQ0MDAwLcwoKCgkICAGHBgYFBQDQAwKk/rqQDw4ODQ0LCwkJBwCEBAIBAQIEBQYICQkLDAw  
HDQ4ODwFQEA8ODg0NCwsJCQcGBQQAQECBAUGBwkJCwsNDQ4ODwAAAAABAAAAAAP0A/IACQAAASE  
FAyUFAYUhAwGJ/OMBXNYBNQElDgE1/on3AnXq/opq6gF96gF9AAAAAIAAAAAA/QDvQB3AO4AAAE  
VHxMPFSEvFT8VLw8PDicfFA8VIT8KLw8/Dy8OIw8OAcSawQFBgYICAoKCgWMDQ8GBgICAQEBAgI  
DAwQENRkZFxcVFRMSEA8ODAOC8QoMDg8QEhMVFRcXGRkaGxsDBwCAGEBAQMDAwQUDQwLcwoKCAg  
GBgUEAwEBAGMEBgCJCgsNDQ4QEBEREhISEhAQDg4NCwoJBwYFAvgBAQMEBQYGCAGKCgsLDA0QBgu  
DAQEBAQICAwMEHjUZGRcXFRUTEhAPDgWKAQMMDAwNDg4dHh8gIQ0MCwoJCQgHBwUFBAUCAQEBAgM  
FBgYICQoLCw0NDg4PBgcHCAgJCQkKCgsLCwsLDBIRERAQDg0NCwoJBwYEAwJWExISERAPDg4NDAw  
KCgkICQYHBQQAUEBAQDAGwXCw0NDg4QERIUFRRYYGhkYFxEhIQDw4NDQsLCwoCBQgEBQQAQGBQU  
FAwQLCAkKCwsMDQ0PDw8RERISFhYVFRMSEQ8ODQsJBwUEAQEEBQcKCg0OEBASFBQVFnCTExEREAE8  
PDQ0MDAOCQKJBQcFBAUFBAUEAwMDDBYMDA0ODw8REhQVfHkaDQ0MDAsKFBMQEA8KDAwMDQ4ODhA  
PEBARERISFRQUBMSEREQDw8NDAsKCQwLCgoJCQgIBgYFBQMCAGEDBQgJCwWOEBESExQVFgAAAAE  
AAAAAAvAD9AAkAAABERSBES8PIW8OAQ/t9QECawUFBwgICgoLDAwNDQ7WDg0NDAwLCgoICACFBQM  
CA238nwEW/uoDYQ4NDQ0LCwsJCQgHBQUDAgEBAGMFBQcICQkLCwsNDQ0AAAAAAwAAAAAD9AP0AD8  
AfwC1AAALHw8/Dy8PDw4FHw8/Dy8PDw4DMxMPAhUfDiE1IS8ENyE/BhM/AjUvBiEnIwLIAQECCBAQ  
FBQCHCAGJCQoKCgoKCgkJCAGHBwUFBAQCAQEBAQIEBAUFBwCICakJCgoKCgoKCQkICACGBgUEBAI  
B/gSBAQIEBAUFBwCICakJCgoKCgoKCQkICACHBQUEBAIBAQEBAgQEBQUHBwgICQkKCgoKCgoJCQg  
IBwYGBQQAQgHJZLVICAECAGQEBQUHBwgICQkJCgsCW/27AwMCAQEsAXEPDQwMCgIvAMEAgIDBQc  
ICQR9FSumcAoKCgkJCAGHBgYFBAQCAQEBAQIEBAUFBwCICakJCgoKCgoKCQkICACHBQUEBAIBAQE  
BAGQEBQUHBwgICQkKCgoKCgoJCQgIBwYGBQQAQgEBAGQECBAQFBQCHCAGJCQoKCgoKCgkJCAGHBwU  
FBAQCAQEBAQIEBAUFBwCICakJCgoDFv6DdyYMCgoKCQkICACGBgUEBAIBAWQBAQMCVQBAGUFBwk  
KAVADBwUQCgkIBwUDAmQAAAAACAAAAA02A/QAAGAFAAAJAQsBCQEDrf0+ATcDbPySAhL+XQMm/Hc  
CCAHAQAQAAAAAAYgD9AADAACwAPAAABESMRIREjEQEzESMBMxEjAu5e/uBeAUPT0/6D09MDufy  
OA3L8jgNy/FMD6PyYA+gAAwAAAAADuAP0ABEAZgD6AAALDwgvBxMzHwCVMx8SDWfCCU/CS8CPxE  
zNT8HBxUPFBUfAg8NFwUfDjsBPw4FNy8MPwI1LyUjDw4CHQEBAgMEBAUFBUFBQQAQAwICKAQECAY  
GBQQAjQLFBAQERERCAgJCACFBQQAQgEBAQ0CAGICBAUGBwkL/ZUMCQcGBQQAQgEBAg0BAQECAwQ  
FBQcICQcQEBAQDw4WOAECBAUGAwCIYh8UFBYLCwsLCwoKDQkIBwUFAwMBAQIGBwEDAwMEBAYGBxA  
PDhIRBQFXAQICBAQEBgYGBwgHCQgJCQkJCAGIBwCGBgUFBAQDAGEBWQUXEW8PDwYGBAQEAQICAQW  
CAQICBAQGBggJCw0LCwsMCwWLFhUTEwEBAwMFBQYGCACJCQkKCgsYCgoKCQgJBwgGBgUFBAICYwE  
FBAQEAWIBAQEBAgMEBAQGA1IBAGUFBwgECUMDCACkDA8RCgsSERERERAEQERAhIqc4HBIQEA4ODQ0  
OAQ8NDQ4OEBASEhUtpyIhEBEQERARERESChIQDQoJBwg/CQgIBwUDAwIoBQoICw4ICQoLDA0OFxQ

```

UFBMTExMSEyUnRm0qGwwKCgkICAcLCQYHBT4BCQkICAgHBgcFBQUDAwICAgICBAQFBQYGBwcICAg
NAT4HCACkDQgICQoKDA0dL58nJRMSExMTExQUFBQSDQwMCgkICAwJBwYMCgsKCQkJBwcGBgQEAW
BAQEDAwQGBQcHCAkJCQoKAAACAAAAAP0A/QAQAEMAAABFQ8PLw8/Dx8OARUPBS8EDwcdAR8DDwY
rAQ8FHQEfBjsBHWUPBB8IPwQfBh0BHwU7Aj8FPQE/BR8EPwgvBD8GOWE/BT0BLwYrAS8FPwQvByS
BDwMvBj0BLwUrAg8FasgBAwUHCAoMDQ4QEBISExQUFBQTEhIQEA4NDAoIBwUDAQEDBQcICgwNDhA
QEHITFBQUFBMSEhAQDg0MCggHBQP+6A4cHA0ODVoDBAQFBAUDA1ICAgICAgJbCA4GBgUEBH0GBQQ
DAwICAQEDAwQEBAV9BAkMBwcIWwMCAQEBAQIDUgIEBQQFBAQDWg0aDg0ODw8CagQDBQQFcAYFBAQ
CAwEOHBwNDg1aAwQEBQQFBAJSAGIBAQEBAgJbCA4GBgUEBH0GBQQDAwICAQEDAwQEBAV9BAkMBwc
IWwMCAQEBAQIDUQMEBAUFBAQDWg0aDg0ODw8CagMEBAUFcAUEBQMEAgICAAoKFBMSEhAQDg0MCgg
HBQMBAMFBwGKDA0OEBASEhMUFBQUExISEBAODQwKCACFAwEBawUHCAoMDQ4QEBISExQBx30ECQw
HBwhbAgIBAQEBAgJSAGQEBQUEBANXDRoODQ4PDwICAwQEBQVwBgUEAwMCAg4cHA0ND1oEBAUEBQQ
FA1ICAgEBAQECALsIDgYGBQQEfQYFBAMDAGICAwMDBQQFfQQJDAcHCFsCagEBAQECAk4DBAQFBQQ
EA1oNGg0ODg8PAgIDBAQFBHEGBAUDAwICDhwcDQ0NWwQEBQQFBAUDUgICAgICALsIDgYGBQQEfQY
FBAMDAGICAgMDBQQAAAAAABIA3gABAAAAAEEEEAAAAAABAAAAAABAAgAAQABAAAAAACAAcACQA
BAAAAAADAAgAEAAABAAAAAEEAAGAGAABAAAAAFAAAsAIAABAAAAAAGAAGAKwABAAAAAAKACw
AMwABAAAAAALABIAXwADAAEECQAAAAIACQADAAEECQABABAAcWADAAEECQACAA4AgwADAAEECQA
DABAAKQADAAEECQAEABAAoQADAAEECQAFABYASQADAAEECQAGABAAxwADAAEECQAKAFgAlwADAAE
ECQALACQBLyBzYi1pY29uc1JlZ3VsYXJzYi1pY29uc3NiLWljY25zVmVyc2lubiAxLjBzYi1pY29
uc0ZvbnQgZ2VuZXJhdGvKIHVzaW5nIFN5bmNmdXNpb24gTWV0cm8gU3R1ZGlvd3d3LnN5bmNmdXN
pb24uY29tACAACwBiAC0AaQbJAG8AbgBzAFIAZQBnAHUAbABhAHIAcWBiAC0AaQbJAG8AbgBzAHM
AYgAtAGkAYwBvAG4AcwBWAGUAcgBzAGkAbwBuACAAMQAuADAACwBiAC0AaQbJAG8AbgBzAEYAbwB
uAHQAIABnAGUAbgB1AHIAZQB0AGUAZAAgAHUAcWBPAG4AZwAgAFMAeQBuaGMAZgB1AHMAaQBvAG4
AIABNAGUAdABYAG8AIABTAHQAdQBkAGkAbwB3AHcAdwAuAHMAeQBuaGMAZgB1AHMAaQBvAG4ALgB
jAG8AbQAAAAACAAAAAa
JAQoBCwEMAQ0BDgEPARABEQESARMABXNoYXJlCHNoYXJlLTAxBGVkaXQhZWZwZGQwMQNhZGQhXb
hcnJvdwVoZWZyYdAntYXAMdm9pY2Utc2VhcmNoCWZhdmd9yaXRlcwtjaGF0LXB1cnNvbGhib29rbWF
yawhzaG9wcGluZwRwbGF5BxBhdXNlCHJlbWluZGVyCHNldHRpbmdzAAAAA=)
format('trueype');
    font-weight: normal;
    font-style: normal;
}
[class^="fab-icon-"],
[class*=" fab-icon-"] {
    font-family: 'fab-icons' !important;
    speak: none;
    font-size: 55px;
    font-style: normal;
    font-weight: normal;
    font-variant: normal;
    text-transform: none;
    line-height: 1;
    -webkit-font-smoothing: antialiased;
    -moz-osx-font-smoothing: grayscale;
}
.fab-icon-people:before {
    content: "\e70a";
}
</style>

```

{% previewsample "page.domainurl/code-snippet/floating-action-button/positions-cs1" %}

### Custom position

You can define the custom position of the Floating Action Button by override the **top**, **left**, **right**, and **bottom** CSS properties using [cssClass](#). For detailed information, refer **app.vue** file below.

### APP.VUE



```

<template>
  <div>
    <div id="targetElement" style="position:relative;min-height:350px;border:1px solid;"></div>
    <!-- To render Floating Action Button -->
    <ejs-fab id='fab' iconCss='e-icons e-edit' cssClass='custom-position' target='#targetElement'></ejs-fab>
  </div>
</template>
<script>
  import Vue from 'vue';
  import { FabPlugin } from "@syncfusion/ej2-vue-buttons";
  import { enableRipple } from '@syncfusion/ej2-base';
  enableRipple(true);
  Vue.use(FabPlugin);
  export default {}
</script>
<style>
  @import '../node_modules/@syncfusion/ej2-base/styles/material.css';
  @import '../node_modules/@syncfusion/ej2-buttons/styles/material.css';
  .e-fab.e-btn.custom-position {
    left: 40px;
    top: 40px;
    bottom: unset;
    right: unset;
  }
</style>

```

{% previewsample "page.domainurl/code-snippet/floating-action-button/positions-cs2" %}

### Events in Vue Floating action button component

This section explains the available events in Floating Action Button component.

#### created

Event triggers after the creation of Floating Action Button.

,

```

<template>
  <!-- To render Floating Action Button -->
  <ejs-fab id='fab' iconCss='e-icons e-edit' content='Edit' :created="onCreate"></ejs-fab>
</template>
<script>
import Vue from 'vue';
import { FabPlugin } from "@syncfusion/ej2-vue-buttons";
Vue.use(FabPlugin);
export default {
  methods: {
    onCreate: function () {

```

```
//Your required action here
```

```
}
```

```
}
```

```
}
```

```
</script>
```

```
<style>
```

```
@import '../node_modules/@syncfusion/ej2-base/styles/material.css';
```

```
@import '../node_modules/@syncfusion/ej2-buttons/styles/material.css';
```

```
</style>
```

```
,
```

[onclick](#)

Event triggers when the Floating Action Button is clicked.

```
,
```

```
<template>
```

```
<!-- To render Floating Action Button -->
```

```
<ejs-fab id='fab' iconCss='e-icons e-edit' content='Edit' v-on:click.native="onClick"></ejs-fab>
```

```
</template>
```

```
<script>
```

```
import Vue from 'vue';
```

```
import { FabPlugin } from "@syncfusion/ej2-vue-buttons";
```

```
Vue.use(FabPlugin);
```

```
export default {
```

```
  methods: {
```

```
    onClick: function () {
```

```
      //Your required action here
```

```
    }
```

```
  }
```

```
}
```

```
</script>
```

```
<style>
```

```
@import '../node_modules/@syncfusion/ej2-base/styles/material.css';
```

```
@import '../node_modules/@syncfusion/ej2-buttons/styles/material.css';
```

```
</style>
```

Below example demonstrates the click event of the Floating Action Button.

#### APP.VUE

```
<template>
  <div>
    <div id="targetElement" style="position:relative;min-height:350px;border:1px solid;"></div>
    <!-- To render Floating Action Button -->
    <ejs-fab id='fab' iconCss='e-icons e-edit' content='Edit' v-on:click.native="onClick" target='#targetElement'>
      </ejs-fab>
    </div>
  </template>
<script>
  import Vue from 'vue';
  import { FabPlugin } from "@syncfusion/ej2-vue-buttons";
  import { enableRipple } from '@syncfusion/ej2-base';
  enableRipple(true);
  Vue.use(FabPlugin);
  export default {
    methods: {
      onClick: function () {
        alert("Edit is clicked!");
      }
    }
  }
</script>
<style>
  @import '../node_modules/@syncfusion/ej2-base/styles/material.css';
  @import '../node_modules/@syncfusion/ej2-buttons/styles/material.css';
</style>
```

{% previewsample "page.domainurl/code-snippet/floating-action-button/events-cs1" %}

## Form Validator

### Validation rules in Vue Form validator component

#### Default Rules

The **FormValidator** has following default validation rules, which are used to validate the form.

Rules	Description	Example
required	The form input element must have any input values	a or 1 or -
email	The form input element must have valid email format values	form@syncfusion.com
url	The form input element must have valid url format values	<a href="http://syncfusion.com/">http://syncfusion.com/</a>
date	The form input element must have valid date format values	05/15/2017
dateISO	The form input element must have valid dateISO format values	2017-05-15

| **number** | The form input element must have valid **number** format values | 1.0 or 1 |

| **digit** | The form input element must have valid **digit** values | 1 |

| **maxLength** | Input value must have less than or equal to **maxLength** character length | if  
**maxLength**: 5, **check** is valid and **checking** is invalid |

| **minLength** | Input value must have greater than or equal to **minLength** character length | if  
**minLength**: 5, **testing** is valid and **test** is invalid |

| **rangeLength** | Input value must have value between **rangeLength** character length | if  
**rangeLength**: [4,5], **test** is valid and **key** is invalid |

| **range** | Input value must have value between **range** number | if **range**: [4,5], **4** is valid and **6** is  
invalid |

| **max** | Input value must have less than or equal to **max** number | if **max**: 3, **3** is valid and **4** is invalid |

| **min** | Input value must have greater than or equal to **min** number | if **min**: 4, **5** is valid and **2** is invalid  
|

### Error messages in Vue Form validator component

The **FormValidator** provides default error messages for all default validation rules.

It is tabulated as follows

Rules	message
-----	-----
<b>required</b>	This field is required.
<b>email</b>	Please enter a valid email address.
<b>url</b>	Please enter a valid URL.
<b>date</b>	Please enter a valid date.
<b>dateIso</b>	Please enter a valid date ( ISO ).
<b>number</b>	Please enter a valid number.
<b>digit</b>	Please enter only digits.
<b>maxLength</b>	Please enter no more than {0} characters.
<b>minLength</b>	Please enter at least {0} characters.
<b>rangeLength</b>	Please enter a value between {0} and {1} characters long.
<b>range</b>	Please enter a value between {0} and {1}.
<b>max</b>	Please enter a value less than or equal to {0}.
<b>min</b>	Please enter a value greater than or equal to {0}.
<b>regex</b>	Please enter a correct value.

### Customizing Error Messages

The default error message for a rule can be customizable by defining it along with concern rule object as follows.

#### APP.VUE

```
<template>
  <div class="wrap">
    <form id="form-element" class="form-horizontal">
      <div class="form-group">
        <div class="col-sm-3 control-label">Select Date</div>
        <div class="col-sm-6">
          <ejs-datepicker
            id="datepicker"
            name="date"
            class="form-control"
            placeholder="Select a date"
          ></ejs-datepicker>
          <div class="error-element"></div>
        </div>
      </div>
    </form>
  </template>
<script>
import Vue from "vue";
import { DatePickerPlugin } from "@syncfusion/ej2-vue-calendars";
import { FormValidator } from "@syncfusion/ej2-vue-inputs";
import { queryParams } from "@syncfusion/ej2-base";
import { throws } from "assert";
Vue.use(DatePickerPlugin);
export default {
  name: "App",
  mounted() {
    var options = {
      rules: {
        date: { required: [true, "Select any value"] }
      },
      customPlacement: function(element, errorElement) {
        document.querySelector(".error-element").appendChild(errorElement);
      },
    };
    // defines FormValidator to validate the TimePicker
    var formObject = new FormValidator("#form-element", options);
  }
}
</script>
<style>
@import "../../node_modules/@syncfusion/ej2-vue-inputs/styles/material.css";
.wrap {
  margin-top: 10px;
}
.error-element {
  margin-top: 5px;
}
#list_event {
  margin-top: 60px;
  padding-left: 5px;
}
```

```

    min-width: 200px;
  }
  #evt {
    border: 1px solid #dcdcdc;
    padding: 10px;
    min-width: 10px;
  }
  .eventarea {
    min-width: 250px;
  }
  #EventLog b {
    color: #388e3c;
  }
  .evtbtn {
    margin-top: 40px;
    margin-left: 70px;
  }
}
</style>

```

{% previewsample "page.domainurl/code-snippet/form-validator/form-validation-events-cs1" %}

### Validation events in Vue Form validator component

Validation events have two types of events. These are

- validationBegin
- validationComplete

#### Validation Begin

**validationBegin** event triggers before the validation starts and also it invoke for each rules in validation process. Please find the below code snippet.

```

`ts
validationBegin: function(args) {
// ("Validation begin");
},
`

```

The following values are passed in **args**. You can use this values in event validation.

Fields	Description
----- -----	
<b>element</b>	Specifies the input element
<b>name</b>	Specifies the event name (validationBegin)
<b>param</b>	Specifies the param value (true/false)
<b>value</b>	Specifies the input value

### Validation Complete

**validationComplete** event triggered after validation is completed and also it invoke for each rules in validation process. Please find the below code snippet.

```
`ts
validationComplete: function(args) {
// ("Validation complete");
}
`
```

The following values are passed in **args**. You can use this values in event validation.

Fields	Description
<b>element</b>	Specifies the input element
<b>name</b>	Specifies the event name (validationComplete)
<b>param</b>	Specifies the param value (true/false)
<b>value</b>	Specifies the input value
<b>status</b>	Specifies the status (success/failure)
<b>inputName</b>	Specifies the type of input

Please find the simple sample for validation events.

### APP.VUE

```
<template>
  <div class="wrap">
    <form id="form-element" class="form-horizontal">
      <div class="form-group">
        <div class="col-sm-3 control-label">Select Date</div>
        <div class="col-sm-6">
          <ejs-datepicker
            id="datepicker"
            name="date"
            class="form-control"
            placeholder="Select a date"
          ></ejs-datepicker>
          <div class="error-element"></div>
        </div>
      </div>
      <div id="list_event">
        <h4>
          <b>Event Trace</b>
        </h4>
        <div id="evt">
          <ul id="eList"></ul>
        </div>
      </div>
    </form>
  </div>
</template>
```

```

    <div class="evtbtn">
      <ejs-button id="clear" @click.native="onClick">Clear</ejs-button>
    </div>
  </div>
</template>
<script>
import Vue from "vue";
import { DatePickerPlugin } from "@syncfusion/ej2-vue-calendars";
import { FormValidator } from "@syncfusion/ej2-vue-inputs";
import { queryParams } from "@syncfusion/ej2-base";
import { ButtonPlugin } from "@syncfusion/ej2-vue-buttons";
import { throws } from "assert";
Vue.use(DatePickerPlugin);
Vue.use(ButtonPlugin);
export default {
  name: "App",
  methods: {
    onClick: function(event) {
      document.getElementById("eList").innerHTML = "";
    },
  },
  mounted() {
    var options = {
      rules: {
        date: { required: true }
      },
      validationBegin: function(args) {
        this.appendElement("Validation begin <hr>");
      },
      appendElement: function(html) {
        let span = document.createElement("span");
        span.innerHTML = html;
        let log = document.getElementById("eList");
        log.insertBefore(span, log.firstChild);
      },
      customPlacement: function(element, errorElement) {
        document.querySelector(".error-element").appendChild(errorElement);
      },
      validationComplete: function(args) {
        if (args.status === "success") {
          this.appendElement("Validation complete success <hr>");
        } else {
          this.appendElement("Validation complete failure <hr>");
        }
      },
    };
    // defines FormValidator to validate the TimePicker
    var formObject = new FormValidator("#form-element", options);
  }
};
</script>
<style>
.wrap {
  margin-top: 10px;
}
.error-element {
  margin-top: 5px;
}

```



```

}
#list_event {
  margin-top: 60px;
  padding-left: 5px;
  min-width: 200px;
}
#evt {
  border: 1px solid #dcdcdc;
  padding: 10px;
  min-width: 10px;
}
.eventarea {
  min-width: 250px;
}
#EventLog b {
  color: #388e3c;
}
.evtbtn {
  margin-top: 40px;
  margin-left: 70px;
}
</style>

```

{% previewsample "page.domainurl/code-snippet/form-validator/form-validation-events-cs2" %}

In the above code example, after completion of validation, the status of the validation process is passed as arguments to validationComplete method and the values are assigned according to the status of the validation.

## Gantt

### Getting Started with the Vue Gantt Component in Vue 2

This article provides a step-by-step guide for setting up a Vue 2 project using [Vue-CLILink to the Video](#) and integrating the Syncfusion Vue Gantt component

To get start quickly with Vue Gantt Chart, you can check on this video:

#### Prerequisites

[System requirements for Syncfusion Vue UI components](#)

#### Dependencies

Following is the list of dependencies to use the Gantt with all features:

```
`javascript
```

```
|-- @syncfusion/ej2-gantt
|-- @syncfusion/ej2-base
|-- @syncfusion/ej2-buttons
|-- @syncfusion/ej2-calendars
|-- @syncfusion/ej2-data
|-- @syncfusion/ej2-dropdowns
```

```
|-- @syncfusion/ej2-grids
|-- @syncfusion/ej2-inputs
|-- @syncfusion/ej2-layouts
|-- @syncfusion/ej2-lists
|-- @syncfusion/ej2-navigations
|-- @syncfusion/ej2-popups
|-- @syncfusion/ej2-richtexteditor
|-- @syncfusion/ej2-treegrid
\
```

### Setting up the Vue 2 project

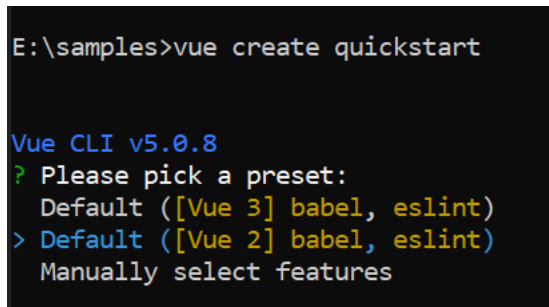
To generate a Vue 2 project using Vue-CLI, use the [vue create](#) command. Follow these steps to install Vue CLI and create a new project:

```
`bash
npm install -g @vue/cli
vue create quickstart
cd quickstart
npm run serve
\
```

or

```
`bash
yarn global add @vue/cli
vue create quickstart
cd quickstart
yarn run serve
\
```

When creating a new project, choose the option **Default ([Vue 2] babel, eslint)** from the menu.



```
E:\samples>vue create quickstart

Vue CLI v5.0.8
? Please pick a preset:
  Default ([Vue 3] babel, eslint)
> Default ([Vue 2] babel, eslint)
  Manually select features
```

Once the **quickstart** project is set up with default settings, proceed to add Syncfusion components to the project.

### Add Syncfusion Vue packages

Syncfusion packages are available at [npmjs.com](https://www.npmjs.com). To use Vue components, install the required npm package.

This article uses the [Vue gantt component](#) as an example. Install the `@syncfusion/ej2-vue-gantt` package by running the following command:

```
`bash
```

```
npm install @syncfusion/ej2-vue-gantt --save
```

```
`
```

or

```
`bash
```

```
yarn add @syncfusion/ej2-vue-gantt
```

```
`
```

### Import Syncfusion CSS styles

You can import themes for the Syncfusion Vue component in various ways, such as using CSS or SASS styles from npm packages, CDN, [CRG](#) and [Theme Studio](#). Refer to [themes topic](#) to know more about built-in themes and different ways to refer to themes in a Vue project.

In this article, the `Material` theme is applied using CSS styles, which are available in installed packages. The necessary `Material` CSS styles for the Gantt component and its dependents were imported into the `<style>` section of `src/App.vue` file.

#### ~/SRC/APP.VUE

```
<style>
@import '../node_modules/@syncfusion/ej2-base/styles/material.css';
@import '../node_modules/@syncfusion/ej2-buttons/styles/material.css';
@import '../node_modules/@syncfusion/ej2-calendars/styles/material.css';
@import '../node_modules/@syncfusion/ej2-dropdowns/styles/material.css';
@import '../node_modules/@syncfusion/ej2-inputs/styles/material.css';
@import '../node_modules/@syncfusion/ej2-navigations/styles/material.css';
@import '../node_modules/@syncfusion/ej2-lists/styles/material.css';
@import '../node_modules/@syncfusion/ej2-layouts/styles/material.css';
@import '../node_modules/@syncfusion/ej2-popups/styles/material.css';
@import '../node_modules/@syncfusion/ej2-splitbuttons/styles/material.css';
@import '../node_modules/@syncfusion/ej2-grids/styles/material.css';
@import '../node_modules/@syncfusion/ej2-richtexteditor/styles/material.css';
@import '../node_modules/@syncfusion/ej2-treegrid/styles/material.css';
@import "../node_modules/@syncfusion/ej2-vue-gantt/styles/material.css";
</style>
```

**Note:** Use above CSS styles in `<style> </style>` tag for rendering all code snippets.

### Add Syncfusion Vue component

Follow the below steps to add the Vue Gantt component:

1\ First, import and register the Gantt component in the `script` section of the `src/App.vue` file.

### ~/SRC/APP.VUE

```
<script>
import { GanttComponent } from "@syncfusion/ej2-vue-gantt";
export default {
  components: {
    'ejs-gantt': GanttComponent
  }
}
</script>
```

2\ In the **template** section, define the Gantt component.

### ~/SRC/APP.VUE

```
<template>
<div id="app">
<ejs-gantt></ejs-gantt>
</div>
</template>
```

### Module injection

The Gantt components was segregated into individual feature-wise modules. To use its feature, you need to inject its feature service in the AppModule.

Find the relevant feature modules and descriptions as follows:

- [Edit](#) : Inject this module to use the editing feature.
- [Filter](#) : Inject this module to use the filtering feature.
- [Sort](#) : Inject this module to use the sorting feature.
- [Selection](#) : Inject this module to use the selection feature.
- [Toolbar](#) : Inject this module to use the toolbar items.
- [DayMarkers](#) : Inject this module to highlight the days.

Now, import the above-mentioned modules from the Gantt package and inject them using **providelike** following code:

### ~/SRC/APP.VUE

```
<script>
import { GanttComponent , Edit, Filter, Sort } from "@syncfusion/ej2-vue-gantt";
export default {
  provide: {
    gantt: [ Edit, Filter, Sort ]
  }
};
</script>
```

### Binding Gantt with data

Bind data with the Gantt component by using the [dataSource](#) property. It accepts an array of JavaScript object or the DataManager instance.

~/SRC/APP.VUE

```

<template>
<div>
<ejs-gantt ref='gantt' :dataSource="data"></ejs-gantt>
</div>
</template>
<script>
import { GanttComponent } from "@syncfusion/ej2-vue-gantt";
export default {
  components: {
    'ejs-gantt': GanttComponent
  },
  data: function() {
    return{
      data: [
        {
          TaskID: 1,
          TaskName: 'Project Initiation',
          StartDate: new Date('04/02/2019'),
          EndDate: new Date('04/21/2019'),
          subtasks: [
            { TaskID: 2, TaskName: 'Identify Site location', StartDate: new
              Date('04/02/2019'), Duration: 4, Progress: 50 },
            { TaskID: 3, TaskName: 'Perform Soil test', StartDate: new
              Date('04/02/2019'), Duration: 4, Progress: 50 },
            { TaskID: 4, TaskName: 'Soil test approval', StartDate: new
              Date('04/02/2019'), Duration: 4, Progress: 50 },
          ]
        },
        {
          TaskID: 5,
          TaskName: 'Project Estimation',
          StartDate: new Date('04/02/2019'),
          EndDate: new Date('04/21/2019'),
          subtasks: [
            { TaskID: 6, TaskName: 'Develop floor plan for estimation', StartDate: new
              Date('04/04/2019'), Duration: 3, Progress: 50 },
            { TaskID: 7, TaskName: 'List materials', StartDate: new Date('04/04/2019'),
              Duration: 3, Progress: 50 },
            { TaskID: 8, TaskName: 'Estimation approval', StartDate: new
              Date('04/04/2019'), Duration: 3, Progress: 50 }
          ]
        },
      ],
    };
  },
};
</script>

```

## Mapping task fields

The data source fields that are required to render the tasks are mapped to the Gantt control using the [taskFields](#) property.

~/SRC/APP.VUE

```

<template>
  <div>
    <ejs-gantt ref='gantt' :dataSource="data" id="GanttContainer"
    :taskFields = "taskFields" :height = "height"></ejs-gantt>
  </div>
</template>
<script>
import { GanttComponent } from "@syncfusion/ej2-vue-gantt";
export default {
  components: {
    'ejs-gantt': GanttComponent
  },
  data: function() {
    return{
      data: [
        {
          TaskID: 1,
          TaskName: 'Project Initiation',
          StartDate: new Date('04/02/2019'),
          EndDate: new Date('04/21/2019'),
          subtasks: [
            { TaskID: 2, TaskName: 'Identify Site location',
StartDate: new Date('04/02/2019'), Duration: 4, Progress: 50 },
            { TaskID: 3, TaskName: 'Perform Soil test', StartDate:
new Date('04/02/2019'), Duration: 4, Progress: 50 },
            { TaskID: 4, TaskName: 'Soil test approval', StartDate:
new Date('04/02/2019'), Duration: 4, Progress: 50 },
          ]
        },
        {
          TaskID: 5,
          TaskName: 'Project Estimation',
          StartDate: new Date('04/02/2019'),
          EndDate: new Date('04/21/2019'),
          subtasks: [
            { TaskID: 6, TaskName: 'Develop floor plan for
estimation', StartDate: new Date('04/04/2019'), Duration: 3, Progress: 50 },
            { TaskID: 7, TaskName: 'List materials', StartDate: new
Date('04/04/2019'), Duration: 3, Progress: 50 },
            { TaskID: 8, TaskName: 'Estimation approval', StartDate:
new Date('04/04/2019'), Duration: 3, Progress: 50 }
          ]
        },
      ],
      height: '450px',
      taskFields: {
        id: 'TaskID',
        name: 'TaskName',
        startDate: 'StartDate',
        endDate: 'EndDate',
        duration: 'Duration',
        progress: 'Progress',
        child: 'subtasks'
      },
    };
  },
};

```

```

};
</script>
<style>
@import '../node_modules/@syncfusion/ej2-base/styles/material.css';
@import '../node_modules/@syncfusion/ej2-buttons/styles/material.css';
@import '../node_modules/@syncfusion/ej2-calendars/styles/material.css';
@import '../node_modules/@syncfusion/ej2-dropdowns/styles/material.css';
@import '../node_modules/@syncfusion/ej2-inputs/styles/material.css';
@import '../node_modules/@syncfusion/ej2-navigations/styles/material.css';
@import '../node_modules/@syncfusion/ej2-lists/styles/material.css';
@import '../node_modules/@syncfusion/ej2-layouts/styles/material.css';
@import '../node_modules/@syncfusion/ej2-popups/styles/material.css';
@import '../node_modules/@syncfusion/ej2-splitbuttons/styles/material.css';
@import '../node_modules/@syncfusion/ej2-grids/styles/material.css';
@import '../node_modules/@syncfusion/ej2-richtexteditor/styles/material.css';
@import '../node_modules/@syncfusion/ej2-treegrid/styles/material.css';
@import "../node_modules/@syncfusion/ej2-vue-gantt/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/gantt/getting-started-cs1" %}

Note : While creating a Syncfusion Vue components using [Direct Scripts](#), camelCased property (dataSource) names need to specify in the [kebab-cased](#) (data-source) equivalents.

### Defining timeline

The Gantt has an option to define timeline using [timelineSettings](#) property with various options. Using this property we can customize the Gantt timeline.

### ~/SRC/APP.VUE

```

<template>
  <div>
    <ejs-gantt id="GanttContainer" :dataSource="data" :taskFields =
      "taskFields" :height = "height" :timelineSettings="timelineSettings"></ejs-
      gantt>
    </div>
  </template>
<script>
import { GanttComponent } from "@syncfusion/ej2-vue-gantt";
export default {
  components: {
    'ejs-gantt': GanttComponent
  },
  data: function() {
    return{
      data: [
        {
          TaskID: 1,
          TaskName: 'Project Initiation',
          StartDate: new Date('04/02/2019'),
          EndDate: new Date('04/21/2019'),
          subtasks: [
            { TaskID: 2, TaskName: 'Identify Site location',
              StartDate: new Date('04/02/2019'), Duration: 4, Progress: 50 },

```

```

        { TaskID: 3, TaskName: 'Perform Soil test', StartDate:
new Date('04/02/2019'), Duration: 4, Progress: 50 },
        { TaskID: 4, TaskName: 'Soil test approval', StartDate:
new Date('04/02/2019'), Duration: 4, Progress: 50 },
    ]
},
{
    TaskID: 5,
    TaskName: 'Project Estimation',
    StartDate: new Date('04/02/2019'),
    EndDate: new Date('04/21/2019'),
    subtasks: [
        { TaskID: 6, TaskName: 'Develop floor plan for
estimation', StartDate: new Date('04/04/2019'), Duration: 3, Progress: 50 },
        { TaskID: 7, TaskName: 'List materials', StartDate: new
Date('04/04/2019'), Duration: 3, Progress: 50 },
        { TaskID: 8, TaskName: 'Estimation approval', StartDate:
new Date('04/04/2019'), Duration: 3, Progress: 50 }
    ]
},
],
height: '450px',
taskFields: {
    id: 'TaskID',
    name: 'TaskName',
    startDate: 'StartDate',
    endDate: 'EndDate',
    duration: 'Duration',
    progress: 'Progress',
    dependency: 'Predecessor',
    child: 'subtasks'
},
timelineSettings: {
    topTier: {
        format: 'MMM dd, yyyy',
        unit: 'Week',
    },
    bottomTier: {
        unit: 'Day',
    },
},
},
};
},
};
</script>
<style>
@import '../node_modules/@syncfusion/ej2-base/styles/material.css';
@import '../node_modules/@syncfusion/ej2-buttons/styles/material.css';
@import '../node_modules/@syncfusion/ej2-calendars/styles/material.css';
@import '../node_modules/@syncfusion/ej2-dropdowns/styles/material.css';
@import '../node_modules/@syncfusion/ej2-inputs/styles/material.css';
@import '../node_modules/@syncfusion/ej2-navigations/styles/material.css';
@import '../node_modules/@syncfusion/ej2-lists/styles/material.css';
@import '../node_modules/@syncfusion/ej2-layouts/styles/material.css';
@import '../node_modules/@syncfusion/ej2-popups/styles/material.css';
@import '../node_modules/@syncfusion/ej2-splitbuttons/styles/material.css';
@import '../node_modules/@syncfusion/ej2-grids/styles/material.css';

```



```
@import '../node_modules/@syncfusion/ej2-richtexteditor/styles/material.css';
@import '../node_modules/@syncfusion/ej2-treegrid/styles/material.css';
@import "../node_modules/@syncfusion/ej2-vue-gantt/styles/material.css";
</style>
```

```
{% previewsample "page.domainurl/code-snippet/gantt/getting-started-cs2" %}
```

### Enabling toolbar

The [toolbar](#) property is used to add the toolbar items like Add, Remove, Edit, Update, Delete, Expand All, Collapse All in Gantt.

To use toolbar, inject the **Toolbar** module in the **provide** section.

### ~/SRC/APP.VUE

```
<template>
  <div>
    <ejs-gantt ref='gantt' id="GanttContainer" :dataSource="data"
      :taskFields = "taskFields" :height = "height" :toolbar="toolbar"
      :editSettings= "editSettings"></ejs-gantt>
  </div>
</template>
<script>
import { GanttComponent, Edit, Selection, Toolbar, DayMarkers } from
"@syncfusion/ej2-vue-gantt";
export default {
  components: {
    'ejs-gantt': GanttComponent
  },
  data: function() {
    return {
      data: [
        {
          TaskID: 1,
          TaskName: 'Project Initiation',
          StartDate: new Date('04/02/2019'),
          EndDate: new Date('04/21/2019'),
          subtasks: [
            { TaskID: 2, TaskName: 'Identify Site location',
              StartDate: new Date('04/02/2019'), Duration: 4, Progress: 50 },
            { TaskID: 3, TaskName: 'Perform Soil test', StartDate:
              new Date('04/02/2019'), Duration: 4, Progress: 50 },
            { TaskID: 4, TaskName: 'Soil test approval', StartDate:
              new Date('04/02/2019'), Duration: 4, Progress: 50 },
          ]
        },
        {
          TaskID: 5,
          TaskName: 'Project Estimation',
          StartDate: new Date('04/02/2019'),
          EndDate: new Date('04/21/2019'),
          subtasks: [
            { TaskID: 6, TaskName: 'Develop floor plan for
              estimation', StartDate: new Date('04/04/2019'), Duration: 3, Progress: 50 },
```

```

        { TaskID: 7, TaskName: 'List materials', StartDate: new
Date('04/04/2019'), Duration: 3, Progress: 50 },
        { TaskID: 8, TaskName: 'Estimation approval', StartDate:
new Date('04/04/2019'), Duration: 3, Progress: 50 }
    ]
  },
],
  height: '450px',
  taskFields: {
    id: 'TaskID',
    name: 'TaskName',
    startDate: 'StartDate',
    endDate: 'EndDate',
    duration: 'Duration',
    progress: 'Progress',
    dependency: 'Predecessor',
    child: 'subtasks'
  },
  toolbar: ['Add', 'Edit', 'Update', 'Delete', 'Cancel',
'ExpandAll', 'CollapseAll'],
  editSettings: {
    allowAdding: true,
    allowEditing: true,
    allowDeleting: true,
    allowTaskbarEditing: true,
    showDeleteConfirmDialog: true
  },
},
};
},
provide: {
  gantt: [ Edit, Selection, Toolbar, DayMarkers ]
}
};
</script>
<style>
@import '../node_modules/@syncfusion/ej2-base/styles/material.css';
@import '../node_modules/@syncfusion/ej2-buttons/styles/material.css';
@import '../node_modules/@syncfusion/ej2-calendars/styles/material.css';
@import '../node_modules/@syncfusion/ej2-dropdowns/styles/material.css';
@import '../node_modules/@syncfusion/ej2-inputs/styles/material.css';
@import '../node_modules/@syncfusion/ej2-navigations/styles/material.css';
@import '../node_modules/@syncfusion/ej2-lists/styles/material.css';
@import '../node_modules/@syncfusion/ej2-layouts/styles/material.css';
@import '../node_modules/@syncfusion/ej2-popups/styles/material.css';
@import '../node_modules/@syncfusion/ej2-splitbuttons/styles/material.css';
@import '../node_modules/@syncfusion/ej2-grids/styles/material.css';
@import '../node_modules/@syncfusion/ej2-richtexteditor/styles/material.css';
@import '../node_modules/@syncfusion/ej2-treegrid/styles/material.css';
@import "../node_modules/@syncfusion/ej2-vue-gantt/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/gantt/getting-started-cs3" %}

### Enabling editing

The editing feature enables you to edit the tasks in Gantt component. It can be enabled by using the [editSettings.allowEditing](#) and [editSettings.allowTaskbarEditing](#) properties.

To use Editing, inject the [Edit](#) module in the `provide` section.

The following editing options are available to update the tasks in Gantt:

- Cell
- Dialog
- Taskbar
- Connector line

### Cell editing

Modify the task details through cell editing by setting the edit [mode](#) property as `Auto`. To enable edit support [Edit](#) module should be injected in Gantt. If [Edit](#) module is not injected, you cannot do any editing action in Gantt.

### ~/SRC/APP.VUE

```
<template>
  <div>
    <ejs-gantt ref='gantt' id="GanttContainer" :dataSource="data"
    :taskFields = "taskFields" :height = "height" :toolbar="toolbar" :columns =
    "columns" :editSettings= "editSettings"></ejs-gantt>
  </div>
</template>
<script>
import { GanttComponent, Edit } from "@syncfusion/ej2-vue-gantt";
export default {
  components: {
    'ejs-gantt': GanttComponent
  },
  data: function() {
    return {
      data: [
        {
          TaskID: 1,
          TaskName: 'Project Initiation',
          StartDate: new Date('04/02/2019'),
          EndDate: new Date('04/21/2019'),
          subtasks: [
            { TaskID: 2, TaskName: 'Identify Site location',
              StartDate: new Date('04/02/2019'), Duration: 4, Progress: 50 },
            { TaskID: 3, TaskName: 'Perform Soil test', StartDate:
              new Date('04/02/2019'), Duration: 4, Progress: 50 },
            { TaskID: 4, TaskName: 'Soil test approval', StartDate:
              new Date('04/02/2019'), Duration: 4, Progress: 50 },
          ]
        },
        {
          TaskID: 5,
          TaskName: 'Project Estimation',
          StartDate: new Date('04/02/2019'),
          EndDate: new Date('04/21/2019'),
        }
      ]
    }
  }
}
```

```

        subtasks: [
            { TaskID: 6, TaskName: 'Develop floor plan for
estimation', StartDate: new Date('04/04/2019'), Duration: 3, Progress: 50 },
            { TaskID: 7, TaskName: 'List materials', StartDate: new
Date('04/04/2019'), Duration: 3, Progress: 50 },
            { TaskID: 8, TaskName: 'Estimation approval', StartDate:
new Date('04/04/2019'), Duration: 3, Progress: 50 }
        ]
    },
],
height: '450px',
taskFields: {
    id: 'TaskID',
    name: 'TaskName',
    startDate: 'StartDate',
    endDate: 'EndDate',
    duration: 'Duration',
    progress: 'Progress',
    child: 'subtasks'
},
columns: [
    { field: 'TaskID', headerText: 'Task ID', textAlign: 'Left',
width: '100' },
    { field: 'TaskName', headerText: 'Task Name', width: '250'
},
    { field: 'StartDate', headerText: 'Start Date', width: '150'
},
    { field: 'Duration', headerText: 'Duration', width: '150' },
    { field: 'Progress', headerText: 'Progress', width: '150' },
],
toolbar: ['Edit'],
editSettings: {
    allowEditing: true,
    mode: "Auto"
},
};
},
provide: {
    gantt: [ Edit ]
}
};
</script>
<style>
@import '../node_modules/@syncfusion/ej2-base/styles/material.css';
@import '../node_modules/@syncfusion/ej2-buttons/styles/material.css';
@import '../node_modules/@syncfusion/ej2-calendars/styles/material.css';
@import '../node_modules/@syncfusion/ej2-dropdowns/styles/material.css';
@import '../node_modules/@syncfusion/ej2-inputs/styles/material.css';
@import '../node_modules/@syncfusion/ej2-navigations/styles/material.css';
@import '../node_modules/@syncfusion/ej2-lists/styles/material.css';
@import '../node_modules/@syncfusion/ej2-layouts/styles/material.css';
@import '../node_modules/@syncfusion/ej2-popups/styles/material.css';
@import '../node_modules/@syncfusion/ej2-splitbuttons/styles/material.css';
@import '../node_modules/@syncfusion/ej2-grids/styles/material.css';
@import '../node_modules/@syncfusion/ej2-
richtexteditor/styles/material.css';
@import '../node_modules/@syncfusion/ej2-treegrid/styles/material.css';

```

```
@import "../node_modules/@syncfusion/ej2-vue-gantt/styles/material.css";
</style>
```

```
{% previewsample "page.domainurl/code-snippet/gantt/getting-started-cs4" %}
```

**Note:** When the edit mode is set to **Auto**, you can change the cells to editable mode by double-clicking anywhere at the TreeGrid and edit the task details in the edit dialog by double-clicking anywhere at the chart.

### Dialog editing

Modify the task details through dialog by setting edit [mode](#) property as **Dialog**.

### ~/SRC/APP.VUE

```
<template>
  <div>
    <ejs-gantt ref='gantt' id="GanttContainer" :dataSource="data"
    :taskFields = "taskFields" :height = "height" :toolbar="toolbar" :columns =
    "columns" :editSettings= "editSettings"></ejs-gantt>
  </div>
</template>
<script>
import { GanttComponent, Edit } from "@syncfusion/ej2-vue-gantt";
export default {
  components: {
    'ejs-gantt': GanttComponent
  },
  data: function() {
    return{
      data: [
        {
          TaskID: 1,
          TaskName: 'Project Initiation',
          StartDate: new Date('04/02/2019'),
          EndDate: new Date('04/21/2019'),
          subtasks: [
            { TaskID: 2, TaskName: 'Identify Site location',
              StartDate: new Date('04/02/2019'), Duration: 4, Progress: 50 },
            { TaskID: 3, TaskName: 'Perform Soil test', StartDate:
              new Date('04/02/2019'), Duration: 4, Progress: 50 },
            { TaskID: 4, TaskName: 'Soil test approval', StartDate:
              new Date('04/02/2019'), Duration: 4, Progress: 50 },
          ]
        },
        {
          TaskID: 5,
          TaskName: 'Project Estimation',
          StartDate: new Date('04/02/2019'),
          EndDate: new Date('04/21/2019'),
          subtasks: [
            { TaskID: 6, TaskName: 'Develop floor plan for
              estimation', StartDate: new Date('04/04/2019'), Duration: 3, Progress: 50 },
            { TaskID: 7, TaskName: 'List materials', StartDate: new
              Date('04/04/2019'), Duration: 3, Progress: 50 },
            { TaskID: 8, TaskName: 'Estimation approval', StartDate:
              new Date('04/04/2019'), Duration: 3, Progress: 50 }
          ]
        }
      ]
    }
  }
}
```

```

    ],
    height: '450px',
    taskFields: {
      id: 'TaskID',
      name: 'TaskName',
      startDate: 'StartDate',
      endDate: 'EndDate',
      duration: 'Duration',
      progress: 'Progress',
      child: 'subtasks'
    },
    columns: [
      { field: 'TaskID', headerText: 'Task ID', textAlign: 'Left',
width: '100' },
      { field: 'TaskName', headerText: 'Task Name', width: '250' },
      { field: 'StartDate', headerText: 'Start Date', width: '150' },
      { field: 'Duration', headerText: 'Duration', width: '150' },
      { field: 'Progress', headerText: 'Progress', width: '150' }
    ],
    toolbar: ['Edit'],
    editSettings: {
      allowEditing: true,
      mode: "Dialog"
    },
  },
  provide: {
    gantt: [ Edit ]
  }
};
</script>
<style>
@import '../node_modules/@syncfusion/ej2-base/styles/material.css';
@import '../node_modules/@syncfusion/ej2-buttons/styles/material.css';
@import '../node_modules/@syncfusion/ej2-calendars/styles/material.css';
@import '../node_modules/@syncfusion/ej2-dropdowns/styles/material.css';
@import '../node_modules/@syncfusion/ej2-inputs/styles/material.css';
@import '../node_modules/@syncfusion/ej2-navigations/styles/material.css';
@import '../node_modules/@syncfusion/ej2-lists/styles/material.css';
@import '../node_modules/@syncfusion/ej2-layouts/styles/material.css';
@import '../node_modules/@syncfusion/ej2-popups/styles/material.css';
@import '../node_modules/@syncfusion/ej2-splitbuttons/styles/material.css';
@import '../node_modules/@syncfusion/ej2-grids/styles/material.css';
@import '../node_modules/@syncfusion/ej2-richtexteditor/styles/material.css';
@import '../node_modules/@syncfusion/ej2-treegrid/styles/material.css';
@import "../node_modules/@syncfusion/ej2-vue-gantt/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/gantt/getting-started-cs5" %}

**Note:** In dialog editing mode, the edit dialog will appear while performing double click action in both TreeGrid and chart sides.

### Taskbar editing

Modify the task details through user interaction such as resizing and dragging the taskbar by enabling the [allowTaskbarEditing](#) property.

### ~/SRC/APP.VUE

```
<template>
  <div>
    <ejs-gantt ref='gantt' id="GanttContainer" :dataSource="data"
    :taskFields = "taskFields" :height = "height" :toolbar="toolbar" :columns =
    "columns" :editSettings= "editSettings"></ejs-gantt>
  </div>
</template>
<script>
import { GanttComponent, Edit } from "@syncfusion/ej2-vue-gantt";
export default {
  components: {
    'ejs-gantt': GanttComponent
  },
  data: function() {
    return {
      data: [
        {
          TaskID: 1,
          TaskName: 'Project Initiation',
          StartDate: new Date('04/02/2019'),
          EndDate: new Date('04/21/2019'),
          subtasks: [
            { TaskID: 2, TaskName: 'Identify Site location',
              StartDate: new Date('04/02/2019'), Duration: 4, Progress: 50 },
            { TaskID: 3, TaskName: 'Perform Soil test', StartDate:
              new Date('04/02/2019'), Duration: 4, Progress: 50 },
            { TaskID: 4, TaskName: 'Soil test approval', StartDate:
              new Date('04/02/2019'), Duration: 4, Progress: 50 },
          ]
        },
        {
          TaskID: 5,
          TaskName: 'Project Estimation',
          StartDate: new Date('04/02/2019'),
          EndDate: new Date('04/21/2019'),
          subtasks: [
            { TaskID: 6, TaskName: 'Develop floor plan for
              estimation', StartDate: new Date('04/04/2019'), Duration: 3, Progress: 50 },
            { TaskID: 7, TaskName: 'List materials', StartDate: new
              Date('04/04/2019'), Duration: 3, Progress: 50 },
            { TaskID: 8, TaskName: 'Estimation approval', StartDate:
              new Date('04/04/2019'), Duration: 3, Progress: 50 }
          ]
        }
      ],
      height: '450px',
      taskFields: {
        id: 'TaskID',

```

```

        name: 'TaskName',
        startDate: 'StartDate',
        endDate: 'EndDate',
        duration: 'Duration',
        progress: 'Progress',
        child: 'subtasks'
    },
    columns: [
        { field: 'TaskID', headerText: 'Task ID', textAlign: 'Left',
width: '100' },
        { field: 'TaskName', headerText: 'Task Name', width: '250'
},
        { field: 'StartDate', headerText: 'Start Date', width: '150'
},
        { field: 'Duration', headerText: 'Duration', width: '150' },
        { field: 'Progress', headerText: 'Progress', width: '150' },
    ],
    toolbar: ['Edit'],
    editSettings: {
        allowTaskbarEditing: true
    },
    };
    },
    provide: {
        gantt: [ Edit ]
    }
    };
</script>
<style>
@import '../node_modules/@syncfusion/ej2-base/styles/material.css';
@import '../node_modules/@syncfusion/ej2-buttons/styles/material.css';
@import '../node_modules/@syncfusion/ej2-calendars/styles/material.css';
@import '../node_modules/@syncfusion/ej2-dropdowns/styles/material.css';
@import '../node_modules/@syncfusion/ej2-inputs/styles/material.css';
@import '../node_modules/@syncfusion/ej2-navigations/styles/material.css';
@import '../node_modules/@syncfusion/ej2-lists/styles/material.css';
@import '../node_modules/@syncfusion/ej2-layouts/styles/material.css';
@import '../node_modules/@syncfusion/ej2-popups/styles/material.css';
@import '../node_modules/@syncfusion/ej2-splitbuttons/styles/material.css';
@import '../node_modules/@syncfusion/ej2-grids/styles/material.css';
@import '../node_modules/@syncfusion/ej2-richtexteditor/styles/material.css';
@import '../node_modules/@syncfusion/ej2-treegrid/styles/material.css';
@import "../node_modules/@syncfusion/ej2-vue-gantt/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/gantt/getting-started-cs6" %}

#### Dependency editing

Modify the task dependencies using mouse interactions by enabling the [allowTaskbarEditing](#) property along with mapping the task dependency data source field to the [dependency](#) property.

#### ~/SRC/APP.VUE

```

<template>
  <div>

```



```

    <ejs-gantt ref='gantt' id="GanttContainer" :dataSource="data"
:taskFields = "taskFields" :height = "height" :toolbar="toolbar" :columns =
"columns" :editSettings= "editSettings"></ejs-gantt>
  </div>
</template>
<script>
import { GanttComponent, Edit } from "@syncfusion/ej2-vue-gantt";
export default {
  components: {
    'ejs-gantt': GanttComponent
  },
  data: function() {
    return{
      data: [
        {
          TaskID: 1,
          TaskName: 'Project Initiation',
          StartDate: new Date('04/02/2019'),
          EndDate: new Date('04/21/2019'),
          subtasks: [
            { TaskID: 2, TaskName: 'Identify Site location', StartDate:
new Date('04/02/2019'), Duration: 0, Progress: 50 },
            { TaskID: 3, TaskName: 'Perform Soil test', StartDate: new
Date('04/02/2019'), Duration: 4, Progress: 50 },
            { TaskID: 4, TaskName: 'Soil test approval', StartDate: new
Date('04/02/2019'), Duration: 4,Predecessor:"2FS", Progress: 50 },
          ]
        },
        {
          TaskID: 5,
          TaskName: 'Project Estimation',
          StartDate: new Date('04/02/2019'),
          EndDate: new Date('04/21/2019'),
          subtasks: [
            { TaskID: 6, TaskName: 'Develop floor plan for estimation',
StartDate: new Date('04/04/2019'), Duration: 3, Progress: 50 },
            { TaskID: 7, TaskName: 'List materials', StartDate: new
Date('04/04/2019'), Duration: 3, Progress: 50 },
            { TaskID: 8, TaskName: 'Estimation approval', StartDate: new
Date('04/04/2019'), Duration: 0,Predecessor:"6SS", Progress: 50 }
          ]
        },
      ],
      height: '450px',
      taskFields: {
        id: 'TaskID',
        name: 'TaskName',
        startDate: 'StartDate',
        endDate: 'EndDate',
        duration: 'Duration',
        progress: 'Progress',
        dependency: 'Predecessor',
        child: 'subtasks'
      },
      columns: [
        { field: 'TaskID', headerText: 'Task ID', textAlign: 'Left',
width: '100' },

```

```

        { field: 'TaskName', headerText: 'Task Name', width: '250'
    },
        { field: 'StartDate', headerText: 'Start Date', width: '150'
    },
        { field: 'Duration', headerText: 'Duration', width: '150' },
        { field: 'Progress', headerText: 'Progress', width: '150' },
    ],
    toolbar: ['Edit'],
    editSettings: {
        allowTaskbarEditing: true
    },
    };
    },
    provide: {
        gantt: [ Edit ]
    }
};
</script>
<style>
@import '../node_modules/@syncfusion/ej2-base/styles/material.css';
@import '../node_modules/@syncfusion/ej2-buttons/styles/material.css';
@import '../node_modules/@syncfusion/ej2-calendars/styles/material.css';
@import '../node_modules/@syncfusion/ej2-dropdowns/styles/material.css';
@import '../node_modules/@syncfusion/ej2-inputs/styles/material.css';
@import '../node_modules/@syncfusion/ej2-navigations/styles/material.css';
@import '../node_modules/@syncfusion/ej2-lists/styles/material.css';
@import '../node_modules/@syncfusion/ej2-layouts/styles/material.css';
@import '../node_modules/@syncfusion/ej2-popups/styles/material.css';
@import '../node_modules/@syncfusion/ej2-splitbuttons/styles/material.css';
@import '../node_modules/@syncfusion/ej2-grids/styles/material.css';
@import '../node_modules/@syncfusion/ej2-richtexteditor/styles/material.css';
@import '../node_modules/@syncfusion/ej2-treegrid/styles/material.css';
@import "../node_modules/@syncfusion/ej2-vue-gantt/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/gantt/getting-started-cs7" %}

### Enabling predecessors or task relationships

Predecessor or task dependency in the Gantt component is used to depict the relationship between the tasks.

Start to Start (SS) : You cannot start a task until the dependent task starts.

Start to Finish (SF) : You cannot finish a task until the dependent task finishes.

Finish to Start (FS) : You cannot start a task until the dependent task completes.

Finish to Finish (FF) : You cannot finish a task until the dependent task completes.

You can show the relationship in tasks, by using the [dependency](#) property as shown in the following code example:

### ~/SRC/APP.VUE

```

<template>
  <div>

```

```

    <ejs-gantt ref='gantt' id="GanttContainer" :dataSource="data"
    :taskFields = "taskFields" :height = "height"
    highlightWeekends='true'></ejs-gantt>
  </div>
</template>
<script>
import { GanttComponent } from "@syncfusion/ej2-vue-gantt";
export default {
  components: {
    'ejs-gantt': GanttComponent
  },
  data: function() {
    return{
      data: [
        {
          TaskID: 1,
          TaskName: 'Project Initiation',
          StartDate: new Date('04/02/2019'),
          EndDate: new Date('04/21/2019'),
          subtasks: [
            { TaskID: 2, TaskName: 'Identify Site location', StartDate:
new Date('04/02/2019'), Duration: 0, Progress: 50 },
            { TaskID: 3, TaskName: 'Perform Soil test', StartDate: new
Date('04/02/2019'), Duration: 4, Progress: 50 },
            { TaskID: 4, TaskName: 'Soil test approval', StartDate: new
Date('04/02/2019'), Duration: 4,Predecessor:"2FS", Progress: 50 },
          ]
        },
        {
          TaskID: 5,
          TaskName: 'Project Estimation',
          StartDate: new Date('04/02/2019'),
          EndDate: new Date('04/21/2019'),
          subtasks: [
            { TaskID: 6, TaskName: 'Develop floor plan for estimation',
StartDate: new Date('04/04/2019'), Duration: 3, Progress: 50 },
            { TaskID: 7, TaskName: 'List materials', StartDate: new
Date('04/04/2019'), Duration: 3, Progress: 50 },
            { TaskID: 8, TaskName: 'Estimation approval', StartDate: new
Date('04/04/2019'), Duration: 0,Predecessor:"6SS", Progress: 50 }
          ]
        },
      ],
      height: '450px',
      taskFields: {
        id: 'TaskID',
        name: 'TaskName',
        startDate: 'StartDate',
        endDate: 'EndDate',
        duration: 'Duration',
        progress: 'Progress',
        dependency: 'Predecessor',
        child: 'subtasks'
      },
    };
  },
};

```

```

</script>
<style>
@import '../node_modules/@syncfusion/ej2-base/styles/material.css';
@import '../node_modules/@syncfusion/ej2-buttons/styles/material.css';
@import '../node_modules/@syncfusion/ej2-calendars/styles/material.css';
@import '../node_modules/@syncfusion/ej2-dropdowns/styles/material.css';
@import '../node_modules/@syncfusion/ej2-inputs/styles/material.css';
@import '../node_modules/@syncfusion/ej2-navigations/styles/material.css';
@import '../node_modules/@syncfusion/ej2-lists/styles/material.css';
@import '../node_modules/@syncfusion/ej2-layouts/styles/material.css';
@import '../node_modules/@syncfusion/ej2-popups/styles/material.css';
@import '../node_modules/@syncfusion/ej2-splitbuttons/styles/material.css';
@import '../node_modules/@syncfusion/ej2-grids/styles/material.css';
@import '../node_modules/@syncfusion/ej2-richtexteditor/styles/material.css';
@import '../node_modules/@syncfusion/ej2-treegrid/styles/material.css';
@import "../node_modules/@syncfusion/ej2-vue-gantt/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/gantt/getting-started-cs8" %}

### Assigning resources

You can display and assign the resource for each task in the Gantt control. Create a collection of JSON object, which contains id, name, unit and group of the resources and assign it to the [resources](#) property. Map these fields to the Gantt control using the [resourceFields](#) property.

### ~/SRC/APP.VUE

```

<template>
  <div>
    <ejs-gantt ref='gantt' id="GanttContainer" :dataSource="data"
      :taskFields= "taskFields" :height= "height" :treeColumnIndex= "1"
      :resourceFields= "resourceFields" :resources= "resources"
      :highlightWeekends= "true" :labelSettings= "labelSettings"></ejs-gantt>
    </div>
  </template>
<script>
import { GanttComponent } from "@syncfusion/ej2-vue-gantt";
export default {
  components: {
    'ejs-gantt': GanttComponent
  },
  data: function() {
    return {
      data: [
        {
          TaskID: 1,
          TaskName: 'Project initiation',
          StartDate: new Date('04/02/2019'),
          EndDate: new Date('04/21/2019'),
          subtasks: [
            {TaskID: 2, TaskName: 'Identify site location',
              StartDate: new Date('04/02/2019'), Duration: 0, Progress: 50, resources:
                [1]},

```

```

        {TaskID: 3, TaskName: 'Perform soil test',
StartDate: new Date('04/02/2019'), Duration: 4, Predecessor: '2', Progress:
50, resources: [2, 3, 5]},
        {TaskID: 4, TaskName: 'Soil test approval',
StartDate: new Date('04/02/2019'), Duration: 0, Predecessor: '3', Progress:
50 },
    ],
    {
        TaskID: 5,
        TaskName: 'Project estimation',
        StartDate: new Date('04/02/2019'),
        EndDate: new Date('04/21/2019'),
        subtasks: [
            {TaskID: 6, TaskName: 'Develop floor plan for
estimation', StartDate: new Date('04/04/2019'), Duration: 3, Predecessor:
'4', Progress: 50, resources: [4]},
            {TaskID: 7, TaskName: 'List materials', StartDate:
new Date('04/04/2019'), Duration: 3, Predecessor: '6', resources: [4,
8], Progress: 50},
            {TaskID: 8, TaskName: 'Estimation approval',
StartDate: new Date('04/04/2019'), Duration: 0, Predecessor: '7', resources:
[12, 5]}
        ]
    },
    ],
    taskFields: {
        id: 'TaskID',
        name: 'TaskName',
        startDate: 'StartDate',
        endDate: 'EndDate',
        duration: 'Duration',
        progress: 'Progress',
        dependency: 'Predecessor',
        child: 'subtasks',
        resourceInfo: 'resources'
    },
    height: '450px',
    resourceFields: {
        id: 'resourceId',
        name: 'resourceName',
    },
    resources: [
        { resourceId: 1, resourceName: 'Martin Tamer' },
        { resourceId: 2, resourceName: 'Rose Fuller' },
        { resourceId: 3, resourceName: 'Margaret Buchanan' },
        { resourceId: 4, resourceName: 'Fuller King' },
        { resourceId: 5, resourceName: 'Davolio Fuller' },
        { resourceId: 6, resourceName: 'Van Jack' },
        { resourceId: 7, resourceName: 'Fuller Buchanan' },
        { resourceId: 8, resourceName: 'Jack Davolio' },
        { resourceId: 9, resourceName: 'Tamer Vinet' },
        { resourceId: 10, resourceName: 'Vinet Fuller' },
        { resourceId: 11, resourceName: 'Bergs Anton' },
        { resourceId: 12, resourceName: 'Construction Supervisor' }
    ],
    labelSettings: {

```

```

        leftLabel: 'TaskName',
        rightLabel: 'resources'
      },
    ];
  }
};
</script>
<style>
@import '../node_modules/@syncfusion/ej2-base/styles/material.css';
@import '../node_modules/@syncfusion/ej2-buttons/styles/material.css';
@import '../node_modules/@syncfusion/ej2-calendars/styles/material.css';
@import '../node_modules/@syncfusion/ej2-dropdowns/styles/material.css';
@import '../node_modules/@syncfusion/ej2-inputs/styles/material.css';
@import '../node_modules/@syncfusion/ej2-navigations/styles/material.css';
@import '../node_modules/@syncfusion/ej2-lists/styles/material.css';
@import '../node_modules/@syncfusion/ej2-layouts/styles/material.css';
@import '../node_modules/@syncfusion/ej2-popups/styles/material.css';
@import '../node_modules/@syncfusion/ej2-splitbuttons/styles/material.css';
@import '../node_modules/@syncfusion/ej2-grids/styles/material.css';
@import '../node_modules/@syncfusion/ej2-richtexteditor/styles/material.css';
@import '../node_modules/@syncfusion/ej2-treegrid/styles/material.css';
@import "../node_modules/@syncfusion/ej2-vue-gantt/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/gantt/getting-started-cs9" %}

### Enable filtering

The filtering feature enables you to view reduced amount of records based on filter criteria. Gantt provides support for menu filtering support for each columns. It can be enabled by setting the [allowFiltering](#) property to `true` along with injecting the `Filter` module module as shown in the following code example. Filtering feature can also be customized using the [filterSettings](#) property.

To use Filtering, inject the [Filter](#) module in the `provide` section.

### ~/SRC/APP.VUE

```

<template>
  <div>
    <ejs-gantt ref='gantt' id="GanttContainer" :dataSource="data"
      :taskFields = "taskFields" :height = "height" :columns="columns"
      :toolbar="toolbar" :allowFiltering= "true"
      :timelineSettings="timelineSettings" :splitterSettings= "splitterSettings"
      :labelSettings= "labelSettings" :projectStartDate="projectStartDate"
      :projectEndDate= "projectEndDate"></ejs-gantt>
    </div>
  </template>
<script>
import { GanttComponent, Filter, Toolbar } from "@syncfusion/ej2-vue-gantt";
export default {
  components: {
    'ejs-gantt': GanttComponent
  },
  data: function() {
    return {
      data: [

```

```

    {
      TaskID: 1,
      TaskName: 'Project Initiation',
      StartDate: new Date('04/02/2019'),
      EndDate: new Date('04/21/2019'),
      subtasks: [
        { TaskID: 2, TaskName: 'Identify Site location',
StartDate: new Date('04/02/2019'), Duration: 4, Progress: 50 },
        { TaskID: 3, TaskName: 'Perform Soil test', StartDate:
new Date('04/02/2019'), Duration: 4, Progress: 50 },
        { TaskID: 4, TaskName: 'Soil test approval', StartDate:
new Date('04/02/2019'), Duration: 4, Progress: 50 },
      ]
    },
    {
      TaskID: 5,
      TaskName: 'Project Estimation',
      StartDate: new Date('04/02/2019'),
      EndDate: new Date('04/21/2019'),
      subtasks: [
        { TaskID: 6, TaskName: 'Develop floor plan for
estimation', StartDate: new Date('04/04/2019'), Duration: 3, Progress: 50 },
        { TaskID: 7, TaskName: 'List materials', StartDate: new
Date('04/04/2019'), Duration: 3, Progress: 50 },
        { TaskID: 8, TaskName: 'Estimation approval', StartDate:
new Date('04/04/2019'), Duration: 3, Progress: 50 }
      ]
    },
  ],
  height: '450px',
  taskFields: {
    id: 'TaskID',
    name: 'TaskName',
    startDate: 'StartDate',
    endDate: 'EndDate',
    duration: 'Duration',
    dependency: 'Predecessor',
    child: 'subtasks',
  },
  columns: [
    { field: 'TaskName', headerText: 'Task Name', width: '250' ,
clipMode: 'EllipsisWithTooltip' },
    { field: 'StartDate', headerText: 'Start Date' },
    { field: 'Duration', headerText: 'Duration' },
    { field: 'EndDate', headerText: 'End Date' },
    { field: 'Predecessor', headerText: 'Predecessor' }
  ],
  toolbar: ['Search'],
  timelineSettings: {
    timelineUnitSize: 60,
    topTier: {
      format: 'MMM dd, yyyy',
      unit: 'Day',
    },
    bottomTier: {
      unit: 'Hour',
      format: 'h.mm a'
    }
  }
}

```

```

    },
    },
    splitterSettings: {
      columnIndex: 3
    },
    labelSettings: {
      rightLabel: 'TaskName',
    },
    },
    projectStartDate: new Date('07/16/1969 01:00:00 AM'),
    projectEndDate: new Date('07/25/1969')
  };
},
provide: {
  gantt: [ Filter, Toolbar ]
}
};
</script>
<style>
@import '../node_modules/@syncfusion/ej2-base/styles/material.css';
@import '../node_modules/@syncfusion/ej2-buttons/styles/material.css';
@import '../node_modules/@syncfusion/ej2-calendars/styles/material.css';
@import '../node_modules/@syncfusion/ej2-dropdowns/styles/material.css';
@import '../node_modules/@syncfusion/ej2-inputs/styles/material.css';
@import '../node_modules/@syncfusion/ej2-navigations/styles/material.css';
@import '../node_modules/@syncfusion/ej2-lists/styles/material.css';
@import '../node_modules/@syncfusion/ej2-layouts/styles/material.css';
@import '../node_modules/@syncfusion/ej2-popups/styles/material.css';
@import '../node_modules/@syncfusion/ej2-splitbuttons/styles/material.css';
@import '../node_modules/@syncfusion/ej2-grids/styles/material.css';
@import '../node_modules/@syncfusion/ej2-richtexteditor/styles/material.css';
@import '../node_modules/@syncfusion/ej2-treegrid/styles/material.css';
@import "../node_modules/@syncfusion/ej2-vue-gantt/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/gantt/getting-started-cs10" %}

### Enable sorting

The sorting feature enables you to order the records. It can be enabled by setting the [allowSorting](#) property to true. Provide the [Sort](#) module as follows. If [Sort](#) module is not provided, you cannot sort when a header is clicked. The sorting feature can be customized using the [sortSettings](#)(../api/gantt/sortSettings/) property.

To use Sorting, inject the [Sort](#) module in the `provide` section.

### ~/SRC/APP.VUE

```

<template>
  <div>
    <ejs-gantt ref='gantt' id="GanttContainer" :dataSource="data"
      :taskFields = "taskFields" :height = "height" :columns="columns"
      :splitterSettings= "splitterSettings" :allowSorting= 'true'></ejs-gantt>
    </div>
  </template>
<script>
import { GanttComponent, Sort } from "@syncfusion/ej2-vue-gantt";

```



```

export default {
  components: {
    'ejs-gantt': GanttComponent
  },
  data: function() {
    return {
      data: [
        {
          TaskID: 1,
          TaskName: 'Project Initiation',
          StartDate: new Date('04/02/2019'),
          EndDate: new Date('04/21/2019'),
          subtasks: [
            { TaskID: 2, TaskName: 'Identify Site location',
              StartDate: new Date('04/02/2019'), Duration: 4, Progress: 50 },
            { TaskID: 3, TaskName: 'Perform Soil test', StartDate:
              new Date('04/02/2019'), Duration: 4, Progress: 50 },
            { TaskID: 4, TaskName: 'Soil test approval', StartDate:
              new Date('04/02/2019'), Duration: 4, Progress: 50 },
          ]
        },
        {
          TaskID: 5,
          TaskName: 'Project Estimation',
          StartDate: new Date('04/02/2019'),
          EndDate: new Date('04/21/2019'),
          subtasks: [
            { TaskID: 6, TaskName: 'Develop floor plan for
              estimation', StartDate: new Date('04/04/2019'), Duration: 3, Progress: 50 },
            { TaskID: 7, TaskName: 'List materials', StartDate: new
              Date('04/04/2019'), Duration: 3, Progress: 50 },
            { TaskID: 8, TaskName: 'Estimation approval', StartDate:
              new Date('04/04/2019'), Duration: 3, Progress: 50 }
          ]
        }
      ],
      height: '450px',
      taskFields: {
        id: 'TaskID',
        name: 'TaskName',
        startDate: 'StartDate',
        duration: 'Duration',
        progress: 'Progress',
        child: 'subtasks'
      },
      splitterSettings: {
        columnIndex: 3
      },
      columns: [
        { field: 'TaskID', headerText: 'Task ID', textAlign: 'Left',
          width: '100' },
        { field: 'TaskName', headerText: 'Task Name', width: '250' },
        { field: 'StartDate', headerText: 'Start Date', width: '150' },
        { field: 'Duration', headerText: 'Duration', width: '150' },
        { field: 'Progress', headerText: 'Progress', width: '150' },
      ]
    }
  }
}

```

```

    ],
    };
  },
  provide: {
    gantt: [ Sort ]
  }
};
</script>
<style>
@import '../node_modules/@syncfusion/ej2-base/styles/material.css';
@import '../node_modules/@syncfusion/ej2-buttons/styles/material.css';
@import '../node_modules/@syncfusion/ej2-calendars/styles/material.css';
@import '../node_modules/@syncfusion/ej2-dropdowns/styles/material.css';
@import '../node_modules/@syncfusion/ej2-inputs/styles/material.css';
@import '../node_modules/@syncfusion/ej2-navigations/styles/material.css';
@import '../node_modules/@syncfusion/ej2-lists/styles/material.css';
@import '../node_modules/@syncfusion/ej2-layouts/styles/material.css';
@import '../node_modules/@syncfusion/ej2-popups/styles/material.css';
@import '../node_modules/@syncfusion/ej2-splitbuttons/styles/material.css';
@import '../node_modules/@syncfusion/ej2-grids/styles/material.css';
@import '../node_modules/@syncfusion/ej2-richtexteditor/styles/material.css';
@import '../node_modules/@syncfusion/ej2-treegrid/styles/material.css';
@import "../node_modules/@syncfusion/ej2-vue-gantt/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/gantt/getting-started-cs11" %}

### Defining eventmarkers

The [eventMarkers](#) property in Gantt component is used to highlight the important event in Gantt chart part. By using this feature, you can add the lines and label to highlight important days in your project.

To highlight the days, inject the **DayMarkers** module in the **provide** section.

### ~/SRC/APP.VUE

```

<template>
  <div>
    <ejs-gantt ref='gantt' id="GanttContainer" :dataSource="data"
    :taskFields = "taskFields" :height = "height"
    :eventMarkers="eventMarkers"></ejs-gantt>
  </div>
</template>
<script>
import { GanttComponent, DayMarkers} from "@syncfusion/ej2-vue-gantt";
import { projectNewData } from './data-source.js';
export default {
  components: {
    'ejs-gantt': GanttComponent
  },
  data: function() {
    return{
      data: projectNewData,
      height: '450px',
      taskFields: {
        id: 'TaskID',

```

```

        name: 'TaskName',
        startDate: 'StartDate',
        endDate: 'EndDate',
        duration: 'Duration',
        progress: 'Progress',
        dependency: 'Predecessor',
        child: 'subtasks'
    },
    eventMarkers: [
        {
            day: new Date('04/09/2019'),
            label: 'Research phase'
        }, {
            day: new Date('04/30/2019'),
            label: 'Design phase'
        }, {
            day: new Date('05/23/2019'),
            label: 'Production phase'
        }, {
            day: new Date('06/20/2019'),
            label: 'Sales and marketing phase'
        }
    ]
    };
},
provide: {
    gantt: [DayMarkers]
}
};
</script>
<style>
@import '../node_modules/@syncfusion/ej2-base/styles/material.css';
@import '../node_modules/@syncfusion/ej2-buttons/styles/material.css';
@import '../node_modules/@syncfusion/ej2-calendars/styles/material.css';
@import '../node_modules/@syncfusion/ej2-dropdowns/styles/material.css';
@import '../node_modules/@syncfusion/ej2-inputs/styles/material.css';
@import '../node_modules/@syncfusion/ej2-navigations/styles/material.css';
@import '../node_modules/@syncfusion/ej2-lists/styles/material.css';
@import '../node_modules/@syncfusion/ej2-layouts/styles/material.css';
@import '../node_modules/@syncfusion/ej2-popups/styles/material.css';
@import '../node_modules/@syncfusion/ej2-splitbuttons/styles/material.css';
@import '../node_modules/@syncfusion/ej2-grids/styles/material.css';
@import '../node_modules/@syncfusion/ej2-richtexteditor/styles/material.css';
@import '../node_modules/@syncfusion/ej2-treegrid/styles/material.css';
@import "../node_modules/@syncfusion/ej2-vue-gantt/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/gantt/getting-started-cs12" %}

### Run the application

Now run the `npm run serve` command in the console, it will build your application and open in the web browser.

The following example shows a basic Gantt:

~/SRC/APP.VUE

```

<template>
<div>
<ejs-gantt id="gantt" :dataSource="data" :taskFields = "taskFields"
:height="height"></ejs-gantt>
</div>
</template>
<script>
import { GanttComponent } from '@syncfusion/ej2-vue-gantt';
export default {
  components: {
    'ejs-gantt': GanttComponent
  },
  data: function() {
    return{
      data: [
        {
          TaskID: 1,
          TaskName: 'Project Initiation',
          StartDate: new Date('04/02/2019'),
          EndDate: new Date('04/21/2019'),
          subtasks: [
            { TaskID: 2, TaskName: 'Identify Site location',
StartDate: new Date('04/02/2019'), Duration: 4, Progress: 50 },
            { TaskID: 3, TaskName: 'Perform Soil test', StartDate:
new Date('04/02/2019'), Duration: 4, Progress: 50 },
            { TaskID: 4, TaskName: 'Soil test approval', StartDate:
new Date('04/02/2019'), Duration: 4, Progress: 50 },
          ]
        },
        {
          TaskID: 5,
          TaskName: 'Project Estimation',
          StartDate: new Date('04/02/2019'),
          EndDate: new Date('04/21/2019'),
          subtasks: [
            { TaskID: 6, TaskName: 'Develop floor plan for
estimation', StartDate: new Date('04/04/2019'), Duration: 3, Progress: 50 },
            { TaskID: 7, TaskName: 'List materials', StartDate: new
Date('04/04/2019'), Duration: 3, Progress: 50 },
            { TaskID: 8, TaskName: 'Estimation approval', StartDate:
new Date('04/04/2019'), Duration: 3, Progress: 50 }
          ]
        }
      ],
      height: '450px',
      taskFields: {
        id: 'TaskID',
        name: 'TaskName',
        startDate: 'StartDate',
        endDate: 'EndDate',
        duration: 'Duration',
        progress: 'Progress',
        dependency: 'Predecessor',
        child: 'subtasks'
      }
    }
  }
}

```

```

    },
  },
};
</script>
<style>
@import '../node_modules/@syncfusion/ej2-base/styles/material.css';
@import '../node_modules/@syncfusion/ej2-buttons/styles/material.css';
@import '../node_modules/@syncfusion/ej2-calendars/styles/material.css';
@import '../node_modules/@syncfusion/ej2-dropdowns/styles/material.css';
@import '../node_modules/@syncfusion/ej2-inputs/styles/material.css';
@import '../node_modules/@syncfusion/ej2-navigations/styles/material.css';
@import '../node_modules/@syncfusion/ej2-lists/styles/material.css';
@import '../node_modules/@syncfusion/ej2-layouts/styles/material.css';
@import '../node_modules/@syncfusion/ej2-popups/styles/material.css';
@import '../node_modules/@syncfusion/ej2-splitbuttons/styles/material.css';
@import '../node_modules/@syncfusion/ej2-grids/styles/material.css';
@import '../node_modules/@syncfusion/ej2-richtexteditor/styles/material.css';
@import '../node_modules/@syncfusion/ej2-treegrid/styles/material.css';
@import "../node_modules/@syncfusion/ej2-vue-gantt/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/gantt/getting-started-cs13" %}

## Getting Started with the Vue Gantt Component in Vue 3

This article provides a step-by-step guide for setting up a [Vite](#) project with a JavaScript environment and integrating the Syncfusion Vue Gantt component using the [Composition API](#) / [Options API](#).

The **Composition API** is a new feature introduced in Vue.js 3 that provides an alternative way to organize and reuse component logic. It allows developers to write components as functions that use smaller, reusable functions called composition functions to manage their properties and behavior.

The **Options API** is the traditional way of writing Vue.js components, where the component logic is organized into a series of options that define the component's properties and behavior. These options include data, methods, computed properties, watchers, lifecycle hooks, and more.

### Prerequisites

#### [System requirements for Syncfusion Vue UI components](#)

#### Set up the Vite project

A recommended approach for beginning with Vue is to scaffold a project using [Vite](#). To create a new Vite project, use one of the commands that are specific to either NPM or Yarn.

```
`bash
```

```
npm create vite@latest
```

```
,
```

```
or
```

```
`bash
```

```
yarn create vite
```

```
,
```

Using one of the above commands will lead you to set up additional configurations for the project as below:

1. Define the project name: We can specify the name of the project directly. Let's specify the name of the project as `my-project` for this article.

```
`bash
```

```
? Project name: » my-project
```

```
,
```

2. Select `Vue` as the framework. It will create a Vue 3 project.

```
`bash
```

```
? Select a framework: » - Use arrow-keys. Return to submit.
```

```
Vanilla
```

```
Vue
```

```
React
```

```
Preact
```

```
Lit
```

```
Svelte
```

```
Others
```

```
,
```

3. Choose `JavaScript` as the framework variant to build this Vite project using JavaScript and Vue.

```
`bash
```

```
? Select a variant: » - Use arrow-keys. Return to submit.
```

```
JavaScript
```

```
TypeScript
```

```
Customize with create-vue ↗
```

```
Nuxt ↗
```

```
,
```

4. Upon completing the aforementioned steps to create the `my-project`, run the following command to install its dependencies:

```
`bash
```

```
cd my-project
```

```
npm install
```

```
,
```

```
or
```

```
`bash
```

```
cd my-project
```

```
yarn install
```

```
,
```

Now that **my-project** is ready to run with default settings, let's add Syncfusion components to the project.

### Add Syncfusion Vue packages

Syncfusion Vue component packages are available at [npmjs.com](https://www.npmjs.com). To use Syncfusion Vue components in the project, install the corresponding npm package.

This article uses the [Vue Gantt component](#) as an example. To use the Vue Gantt component in the project, the `@syncfusion/ej2-vue-gantt` package needs to be installed using the following command:

```
`bash
```

```
npm install @syncfusion/ej2-vue-gantt --save
```

```
,
```

or

```
`bash
```

```
yarn add @syncfusion/ej2-vue-gantt
```

```
,
```

### Import Syncfusion CSS styles

You can import themes for the Syncfusion Vue component in various ways, such as using CSS or SASS styles from npm packages, CDN, [CRG](#) and [Theme Studio](#). Refer to [themes topic](#) to know more about built-in themes and different ways to refer to themes in a Vue project.

In this article, **Material** theme is applied using CSS styles, which are available in installed packages. The necessary **Material** CSS styles for the Gantt component and its dependents were imported into the `<style>` section of **src/App.vue** file.

### ~/SRC/APP.VUE

```
<style>
<!-- Material theme used for this sample -->
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-buttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-calendars/styles/material.css";
@import "../node_modules/@syncfusion/ej2-dropdowns/styles/material.css";
@import "../node_modules/@syncfusion/ej2-inputs/styles/material.css";
@import "../node_modules/@syncfusion/ej2-navigations/styles/material.css";
@import "../node_modules/@syncfusion/ej2-popups/styles/material.css";
@import "../node_modules/@syncfusion/ej2-splitbuttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-layouts/styles/material.css";
@import "../node_modules/@syncfusion/ej2-grids/styles/material.css";
@import "../node_modules/@syncfusion/ej2-treegrid/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-gantt/styles/material.css";
</style>
```

The order of importing CSS styles should be in line with its dependency graph.

### Add Syncfusion Vue component

Follow the below steps to add the Vue Gantt component using **Composition API** or **Options API**:

1.First, import and register the Gantt component and its child directives in the **script** section of the **src/App.vue** file. If you are using the **Composition API**, you should add the **setup** attribute to the **script** tag to indicate that Vue will be using the **Composition API**.

#### **COMPOSITION API (~SRC/APP.VUE)**

```
<script setup>
import { GanttComponent as EjsGantt, ColumnsDirective as EColumns,
ColumnDirective as EColumn } from '@syncfusion/ej2-vue-gantt';
</script>
```

#### **OPTIONS API (~SRC/APP.VUE)**

```
<script>
import { GanttComponent, ColumnsDirective, ColumnDirective } from
 '@syncfusion/ej2-vue-gantt';
//Component registration
export default {
  name: "App",
  components: {
    'ejs-gantt': GanttComponent,
    'e-columns': ColumnsDirective,
    'e-column': ColumnDirective
  }
}
</script>
```

2.In the **template** section, define the Gantt component with the [dataSource](#) property and column definitions.

#### **~/SRC/APP.VUE**

```
<template>
<ejs-gantt :dataSource='data' :treeColumnIndex='1' child='subtasks'
:taskFields= 'taskFields'>
<e-columns>
<e-column field='TaskID' headerText='Task ID' textAlign='Right'
width=70></e-column>
<e-column field='TaskName' headerText='Task Name' textAlign='Left'
width=200></e-column>
<e-column field='StartDate' headerText='Start Date' textAlign='Right'
format='yMd' width=90></e-column>
<e-column field='Duration' headerText='Duration' textAlign='Right'
width=80></e-column>
</e-columns>
</ejs-gantt>
</template>
```

3.Declare the values for the **dataSource** property in the **script** section.

#### **COMPOSITION API (~SRC/APP.VUE)**



```

<script setup>
const data = [{
  TaskID: 1,
  TaskName: 'Planning',
  StartDate: new Date('02/03/2017'),
  EndDate: new Date('02/07/2017'),
  Progress: 100,
  Duration: 5,
  subtasks: [
    { TaskID: 2, TaskName: 'Plan timeline', StartDate: new Date('02/03/2017'),
      EndDate: new Date('02/07/2017'), Duration: 5, Progress: 100, },
    { TaskID: 3, TaskName: 'Plan budget', StartDate: new Date('02/03/2017'),
      EndDate: new Date('02/07/2017'), Duration: 5, Progress: 100, },
    { TaskID: 4, TaskName: 'Allocate resources', StartDate: new
      Date('02/03/2017'), EndDate: new Date('02/07/2017'), Duration: 5, Progress:
      100, },
    { TaskID: 5, TaskName: 'Planning complete', StartDate: new
      Date('02/07/2017'), EndDate: new Date('02/07/2017'), Duration: 0, Progress:
      0, }
  ],
},
{
  TaskID: 6,
  TaskName: 'Design',
  StartDate: new Date('02/10/2017'),
  EndDate: new Date('02/14/2017'),
  Duration: 3,
  Progress: 86,
  subtasks: [
    { TaskID: 7, TaskName: 'Software Specification', StartDate: new
      Date('02/10/2017'), EndDate: new Date('02/12/2017'), Duration: 3, Progress:
      60, },
    { TaskID: 8, TaskName: 'Develop prototype', StartDate: new
      Date('02/10/2017'), EndDate: new Date('02/12/2017'), duration: 3, Progress:
      100, },
    { TaskID: 9, TaskName: 'Get approval from customer', startDate: new
      Date('02/13/2017'), EndDate: new Date('02/14/2017'), Duration: 2, Progress:
      100, },
    { TaskID: 10, TaskName: 'Design Documentation', startDate: new
      Date('02/13/2017'), endDate: new Date('02/14/2017'), duration: 2, Progress:
      100, },
    { TaskID: 11, TaskName: 'Design complete', StartDate: new
      Date('02/14/2017'), EndDate: new Date('02/14/2017'), Duration: 0, Progress:
      0, }
  ],
},
],
taskFields: {
  id: 'TaskID',
  name: 'TaskName',
  startDate: 'StartDate',
  endDate: 'EndDate',
  duration: 'Duration',
  progress: 'Progress',
  child: 'subtasks',
}
}
</script>

```

**OPTIONS API (~SRC/APP.VUE)**

```

<script>
data() {
  return {
    data: [{
      TaskID: 1,
      TaskName: 'Planning',
      StartDate: new Date('02/03/2017'),
      EndDate: new Date('02/07/2017'),
      Progress: 100,
      Duration: 5,
      subtasks: [
        { TaskID: 2, TaskName: 'Plan timeline', StartDate: new Date('02/03/2017'),
          EndDate: new Date('02/07/2017'), Duration: 5, Progress: 100, },
        { TaskID: 3, TaskName: 'Plan budget', StartDate: new Date('02/03/2017'),
          EndDate: new Date('02/07/2017'), Duration: 5, Progress: 100, },
        { TaskID: 4, TaskName: 'Allocate resources', StartDate: new
          Date('02/03/2017'), EndDate: new Date('02/07/2017'), Duration: 5, Progress:
          100, },
        { TaskID: 5, TaskName: 'Planning complete', StartDate: new
          Date('02/07/2017'), EndDate: new Date('02/07/2017'), Duration: 0, Progress:
          0, }
      ],
    },
    {
      TaskID: 6,
      TaskName: 'Design',
      StartDate: new Date('02/10/2017'),
      EndDate: new Date('02/14/2017'),
      Duration: 3,
      Progress: 86,
      subtasks: [
        { TaskID: 7, TaskName: 'Software Specification', StartDate: new
          Date('02/10/2017'), EndDate: new Date('02/12/2017'), Duration: 3, Progress:
          60, },
        { TaskID: 8, TaskName: 'Develop prototype', StartDate: new
          Date('02/10/2017'), EndDate: new Date('02/12/2017'), duration: 3, Progress:
          100, },
        { TaskID: 9, TaskName: 'Get approval from customer', startDate: new
          Date('02/13/2017'), EndDate: new Date('02/14/2017'), Duration: 2, Progress:
          100, },
        { TaskID: 10, TaskName: 'Design Documentation', startDate: new
          Date('02/13/2017'), endDate: new Date('02/14/2017'), duration: 2, Progress:
          100, },
        { TaskID: 11, TaskName: 'Design complete', StartDate: new
          Date('02/14/2017'), EndDate: new Date('02/14/2017'), Duration: 0, Progress:
          0, }
      ],
    },
  ],
  taskFields: {
    id: 'TaskID',
    name: 'TaskName',
    startDate: 'StartDate',
    endDate: 'EndDate',
  }
}

```

```

duration: 'Duration',
progress: 'Progress',
child: 'subtasks',
}
};
}
</script>

```

Here is the summarized code for the above steps in the **src/App.vue** file:

### COMPOSITION API (~SRC/APP.VUE)

```

<template>
<ejs-gantt :dataSource='data' :treeColumnIndex='1' child='subtasks'
:taskFields= 'taskFields'>
<e-columns>
<e-column field='TaskID' headerText='Task ID' textAlign='Right'
width=70></e-column>
<e-column field='TaskName' headerText='Task Name' textAlign='Left'
width=200></e-column>
<e-column field='StartDate' headerText='Start Date' textAlign='Right'
format='yMd' width=90></e-column>
<e-column field='Duration' headerText='Duration' textAlign='Right'
width=80></e-column>
</e-columns>
</ejs-gantt>
</template>
<script setup>
const data = [{
TaskID: 1,
TaskName: 'Planning',
StartDate: new Date('02/03/2017'),
EndDate: new Date('02/07/2017'),
Progress: 100,
Duration: 5,
subtasks: [
{ TaskID: 2, TaskName: 'Plan timeline', StartDate: new Date('02/03/2017'),
EndDate: new Date('02/07/2017'), Duration: 5, Progress: 100, },
{ TaskID: 3, TaskName: 'Plan budget', StartDate: new Date('02/03/2017'),
EndDate: new Date('02/07/2017'), Duration: 5, Progress: 100, },
{ TaskID: 4, TaskName: 'Allocate resources', StartDate: new
Date('02/03/2017'), EndDate: new Date('02/07/2017'), Duration: 5, Progress:
100, },
{ TaskID: 5, TaskName: 'Planning complete', StartDate: new
Date('02/07/2017'), EndDate: new Date('02/07/2017'), Duration: 0, Progress:
0, }
]
},
{
TaskID: 6,
TaskName: 'Design',
StartDate: new Date('02/10/2017'),
EndDate: new Date('02/14/2017'),
Duration: 3,
Progress: 86,
subtasks: [

```

```

{ TaskID: 7, TaskName: 'Software Specification', StartDate: new
Date('02/10/2017'), EndDate: new Date('02/12/2017'), Duration: 3, Progress:
60, },
{ TaskID: 8, TaskName: 'Develop prototype', StartDate: new
Date('02/10/2017'), EndDate: new Date('02/12/2017'), duration: 3, Progress:
100, },
{ TaskID: 9, TaskName: 'Get approval from customer', startDate: new
Date('02/13/2017'), EndDate: new Date('02/14/2017'), Duration: 2, Progress:
100, },
{ TaskID: 10, TaskName: 'Design Documentation', startDate: new
Date('02/13/2017'), endDate: new Date('02/14/2017'), duration: 2, Progress:
100, },
{ TaskID: 11, TaskName: 'Design complete', StartDate: new
Date('02/14/2017'), EndDate: new Date('02/14/2017'), Duration: 0, Progress:
0, }
],
taskFields: {
id: 'TaskID',
name: 'TaskName',
startDate: 'StartDate',
endDate: 'EndDate',
duration: 'Duration',
progress: 'Progress',
child: 'subtasks',
}
}
</script>
<style>
<!-- Material theme used for this sample -->
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-buttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-calendars/styles/material.css";
@import "../node_modules/@syncfusion/ej2-dropdowns/styles/material.css";
@import "../node_modules/@syncfusion/ej2-inputs/styles/material.css";
@import "../node_modules/@syncfusion/ej2-navigations/styles/material.css";
@import "../node_modules/@syncfusion/ej2-popups/styles/material.css";
@import "../node_modules/@syncfusion/ej2-splitbuttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-layouts/styles/material.css";
@import "../node_modules/@syncfusion/ej2-grids/styles/material.css";
@import "../node_modules/@syncfusion/ej2-treegrid/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-gantt/styles/material.css";
</style>

```

### OPTIONS API (~SRC/APP.VUE)

```

<template>
<div>
<ejs-gantt ref='gantt' id="GanttContainer" :dataSource="data" :taskFields =
"taskFields" :height = "height" :columns="columns" :toolbar="toolbar"
:allowFiltering= "true" :timelineSettings="timelineSettings"
:splitterSettings= "splitterSettings" :labelSettings= "labelSettings"
:projectStartDate="projectStartDate" :projectEndDate=
"projectEndDate"></ejs-gantt>
</div>
</template>
<script>

```

```

import { GanttComponent, Filter, Toolbar } from '@syncfusion/ej2-vue-gantt';
export default {
  name: "App",
  components: {
    'ejs-gantt' : GanttComponent
  },
  data: function() {
    return{
      data: [
        {
          TaskID: 1,
          TaskName: 'Project Initiation',
          StartDate: new Date('04/02/2019'),
          EndDate: new Date('04/21/2019'),
          subtasks: [
            { TaskID: 2, TaskName: 'Identify Site location', StartDate: new
              Date('04/02/2019'), Duration: 4, Progress: 50 },
            { TaskID: 3, TaskName: 'Perform Soil test', StartDate: new
              Date('04/02/2019'), Duration: 4, Progress: 50 },
            { TaskID: 4, TaskName: 'Soil test approval', StartDate: new
              Date('04/02/2019'), Duration: 4, Progress: 50 },
          ]
        },
        {
          TaskID: 5,
          TaskName: 'Project Estimation',
          StartDate: new Date('04/02/2019'),
          EndDate: new Date('04/21/2019'),
          subtasks: [
            { TaskID: 6, TaskName: 'Develop floor plan for estimation', StartDate: new
              Date('04/04/2019'), Duration: 3, Progress: 50 },
            { TaskID: 7, TaskName: 'List materials', StartDate: new Date('04/04/2019'),
              Duration: 3, Progress: 50 },
            { TaskID: 8, TaskName: 'Estimation approval', StartDate: new
              Date('04/04/2019'), Duration: 3, Progress: 50 }
          ]
        },
      ],
      height: '450px',
      taskFields: {
        id: 'TaskID',
        name: 'TaskName',
        startDate: 'StartDate',
        endDate: 'EndDate',
        duration: 'Duration',
        dependency: 'Predecessor',
        child: 'subtasks',
      },
      columns: [
        { field: 'TaskName', headerText: 'Task Name', width: '250' , clipMode:
          'EllipsisWithTooltip' },
        { field: 'StartDate', headerText: 'Start Date' },
        { field: 'Duration', headerText: 'Duration' },
        { field: 'EndDate', headerText: 'End Date' },
        { field: 'Predecessor', headerText: 'Predecessor' }
      ],
      toolbar: ['Search'],
    }
  }
}

```

```

timelineSettings: {
  timelineUnitSize: 60,
  topTier: {
    format: 'MMM dd, yyyy',
    unit: 'Week',
  },
  bottomTier: {
    unit: 'Day',
  },
},
splitterSettings: {
  columnIndex: 3
},
labelSettings: {
  rightLabel: 'TaskName',
},
projectStartDate: new Date('04/01/2019 01:00:00 AM'),
projectEndDate: new Date('05/10/2019')
};
},
provide: {
  gantt: [ Filter, Toolbar ]
}
};
</script>
<style>
<!-- Material theme used for this sample -->
@import
"https://ej2.syncfusion.com/vue/documentation/node\_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-buttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-calendars/styles/material.css";
@import "../node_modules/@syncfusion/ej2-dropdowns/styles/material.css";
@import "../node_modules/@syncfusion/ej2-inputs/styles/material.css";
@import "../node_modules/@syncfusion/ej2-navigations/styles/material.css";
@import "../node_modules/@syncfusion/ej2-popups/styles/material.css";
@import "../node_modules/@syncfusion/ej2-splitbuttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-layouts/styles/material.css";
@import "../node_modules/@syncfusion/ej2-grids/styles/material.css";
@import "../node_modules/@syncfusion/ej2-treegrid/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-gantt/styles/material.css";
</style>

```

### Run the project

To run the project, use the following command:

```
`bash
```

```
npm run serve
```

```
`
```

or

```
`bash
```

```
yarn run dev
```

The output will appear as follows:

```
![vueimage](./images/vue-gantt-image.png)
```

#### *Using local style*

Import the needed css styles for the Gantt component along with dependency styles in the `<style>` section of the `src/App.vue` file as follows.

```
<style>
<!-- Material theme used for this sample -->
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-buttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-calendars/styles/material.css";
@import "../node_modules/@syncfusion/ej2-dropdowns/styles/material.css";
@import "../node_modules/@syncfusion/ej2-inputs/styles/material.css";
@import "../node_modules/@syncfusion/ej2-navigations/styles/material.css";
@import "../node_modules/@syncfusion/ej2-popups/styles/material.css";
@import "../node_modules/@syncfusion/ej2-splitbuttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-layouts/styles/material.css";
@import "../node_modules/@syncfusion/ej2-grids/styles/material.css";
@import "../node_modules/@syncfusion/ej2-treegrid/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-gantt/styles/material.css";
</style>
```

Note: Gantt component use other Syncfusion components too, the dependent component's CSS references need to be added for using all the Gantt functionalities.

#### *Using CDN link for style*

Using CDN link, you can directly refer to the Gantt control's style in the `index.html`.

Refer to the Gantt's CDN links as follows.

#### **Syntax:**

Styles: `https://cdn.syncfusion.com/ej2/{PACKAGE_NAME}/styles/material.css`

#### **Example:**

Styles: <https://cdn.syncfusion.com/ej2/ej2-gantt/styles/material.css>

```
`js
```

```
<!-- Material theme used for this sample -->
```

```
<link href="http://cdn.syncfusion.com/ej2/ej2-base/styles/material.css" rel="stylesheet"
type="text/css"/>
<link href="http://cdn.syncfusion.com/ej2/ej2-buttons/styles/material.css" rel="stylesheet"
type="text/css"/>
<link href="http://cdn.syncfusion.com/ej2/ej2-popups/styles/material.css" rel="stylesheet"
type="text/css"/>
<link href="http://cdn.syncfusion.com/ej2/ej2-navigations/styles/material.css" rel="stylesheet"
type="text/css"/>
<link href="http://cdn.syncfusion.com/ej2/ej2-lists/styles/material.css" rel="stylesheet"
type="text/css"/>
<link href="http://cdn.syncfusion.com/ej2/ej2-dropdowns/styles/material.css" rel="stylesheet"
type="text/css"/>
<link href="http://cdn.syncfusion.com/ej2/ej2-inputs/styles/material.css" rel="stylesheet"
type="text/css"/>
<link href="http://cdn.syncfusion.com/ej2/ej2-calendars/styles/material.css" rel="stylesheet"
type="text/css"/>
<link href="http://cdn.syncfusion.com/ej2/ej2-layouts/styles/material.css" rel="stylesheet"
type="text/css"/>
<link href="http://cdn.syncfusion.com/ej2/ej2-richtexteditor/styles/material.css" rel="stylesheet"
type="text/css"/>
<link href="http://cdn.syncfusion.com/ej2/ej2-grids/styles/material.css" rel="stylesheet"
type="text/css"/>
<link href="http://cdn.syncfusion.com/ej2/ej2-treegrid/styles/material.css" rel="stylesheet"
type="text/css"/>
<!-- Essential JS 2 material theme -->
<link href="http://cdn.syncfusion.com/ej2/ej2-gantt/styles/material.css" rel="stylesheet"
type="text/css"/>
```

### Binding Gantt with data

Bind data with the Gantt component by using the [dataSource](#) property. It accepts an array of JavaScript object or the DataManager instance.

```
<template>
<div>
<ejs-gantt ref='gantt' :dataSource="data" :taskFields= "taskFields"></ejs-gantt>
</div>
</template>
<script>
```



```
import { GanttComponent } from '@syncfusion/ej2-vue-gantt';
export default {
  name: "App",
  components: {
    'ejs-gantt': GanttComponent
  },
  data: function() {
    return{
      data: [
        {
          TaskID: 1,
          TaskName: 'Project Initiation',
          StartDate: new Date('04/02/2019'),
          EndDate: new Date('04/21/2019'),
          subtasks: [
            { TaskID: 2, TaskName: 'Identify Site location', StartDate: new Date('04/02/2019'), Duration: 4, Progress: 50 },
            { TaskID: 3, TaskName: 'Perform Soil test', StartDate: new Date('04/02/2019'), Duration: 4, Progress: 50 },
            { TaskID: 4, TaskName: 'Soil test approval', StartDate: new Date('04/02/2019'), Duration: 4, Progress: 50 },
          ],
        },
        {
          TaskID: 5,
          TaskName: 'Project Estimation',
          StartDate: new Date('04/02/2019'),
          EndDate: new Date('04/21/2019'),
          subtasks: [
            { TaskID: 6, TaskName: 'Develop floor plan for estimation', StartDate: new Date('04/04/2019'), Duration: 3, Progress: 50 },
            { TaskID: 7, TaskName: 'List materials', StartDate: new Date('04/04/2019'), Duration: 3, Progress: 50 },
            { TaskID: 8, TaskName: 'Estimation approval', StartDate: new Date('04/04/2019'), Duration: 3, Progress: 50 },
          ],
        }
      ]
    }
  }
}
```

```
},  
],  
taskFields: {  
  id: 'TaskID',  
  name: 'TaskName',  
  startDate: 'StartDate',  
  endDate: 'EndDate',  
  duration: 'Duration',  
  progress: 'Progress',  
  child: 'subtasks',  
}  
};  
},  
};  
</script>  
<style>  
<!-- Material theme used for this sample -->  
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";  
@import "../node_modules/@syncfusion/ej2-buttons/styles/material.css";  
@import "../node_modules/@syncfusion/ej2-calendars/styles/material.css";  
@import "../node_modules/@syncfusion/ej2-dropdowns/styles/material.css";  
@import "../node_modules/@syncfusion/ej2-inputs/styles/material.css";  
@import "../node_modules/@syncfusion/ej2-navigations/styles/material.css";  
@import "../node_modules/@syncfusion/ej2-popups/styles/material.css";  
@import "../node_modules/@syncfusion/ej2-splitbuttons/styles/material.css";  
@import "../node_modules/@syncfusion/ej2-layouts/styles/material.css";  
@import "../node_modules/@syncfusion/ej2-grids/styles/material.css";  
@import "../node_modules/@syncfusion/ej2-treegrid/styles/material.css";  
@import "../node_modules/@syncfusion/ej2-vue-gantt/styles/material.css";  
</style>  
,
```

### Mapping task fields

The data source fields that are required to render the tasks are mapped to the Gantt control using the [taskFields](#) property.

```
,

<template>
<div>
<ejs-gantt ref='gantt' :dataSource="data" id="GanttContainer" :taskFields = "taskFields" :height =
"height"></ejs-gantt>
</div>
</template>
<script>
import { GanttComponent } from '@syncfusion/ej2-vue-gantt';
export default {
name: "App",
components: {
'ejs-gantt' : GanttComponent
},
data: function() {
return{
data: [
{
TaskID: 1,
TaskName: 'Project Initiation',
StartDate: new Date('04/02/2019'),
EndDate: new Date('04/21/2019'),
subtasks: [
{ TaskID: 2, TaskName: 'Identify Site location', StartDate: new Date('04/02/2019'), Duration: 4, Progress:
50 },
{ TaskID: 3, TaskName: 'Perform Soil test', StartDate: new Date('04/02/2019'), Duration: 4, Progress: 50
},
{ TaskID: 4, TaskName: 'Soil test approval', StartDate: new Date('04/02/2019'), Duration: 4, Progress: 50
},
]
},
{
TaskID: 5,
TaskName: 'Project Estimation',
```

```
StartDate: new Date('04/02/2019'),
EndDate: new Date('04/21/2019'),
subtasks: [
  { TaskID: 6, TaskName: 'Develop floor plan for estimation', StartDate: new Date('04/04/2019'), Duration:
  3, Progress: 50 },
  { TaskID: 7, TaskName: 'List materials', StartDate: new Date('04/04/2019'), Duration: 3, Progress: 50 },
  { TaskID: 8, TaskName: 'Estimation approval', StartDate: new Date('04/04/2019'), Duration: 3, Progress:
  50 }
]
},
],
height: '450px',
taskFields: {
  id: 'TaskID',
  name: 'TaskName',
  startDate: 'StartDate',
  endDate: 'EndDate',
  duration: 'Duration',
  progress: 'Progress',
  child: 'subtasks'
},
};
},
};
</script>
<style>
<!-- Material theme used for this sample -->
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-buttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-calendars/styles/material.css";
@import "../node_modules/@syncfusion/ej2-dropdowns/styles/material.css";
@import "../node_modules/@syncfusion/ej2-inputs/styles/material.css";
@import "../node_modules/@syncfusion/ej2-navigations/styles/material.css";
@import "../node_modules/@syncfusion/ej2-popups/styles/material.css";
```

```

@import "../node_modules/@syncfusion/ej2-splitbuttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-layouts/styles/material.css";
@import "../node_modules/@syncfusion/ej2-grids/styles/material.css";
@import "../node_modules/@syncfusion/ej2-treegrid/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-gantt/styles/material.css";
</style>
`

```

### Defining timeline

The Gantt has an option to define timeline using [timelineSettings](#) property with various options. Using this property we can customize the Gantt timeline.

```

<template>
<div>
<ejs-gantt id="GanttContainer" :dataSource="data" :taskFields = "taskFields" :height = "height"
:timelineSettings="timelineSettings"></ejs-gantt>
</div>
</template>
<script>
import { GanttComponent } from '@syncfusion/ej2-vue-gantt';
export default {
name: "App",
components: {
'ejs-gantt' : GanttComponent
},
data: function() {
return{
data: [
{
TaskID: 1,
TaskName: 'Project Initiation',
StartDate: new Date('04/02/2019'),
EndDate: new Date('04/21/2019'),
subtasks: [

```

```
{ TaskID: 2, TaskName: 'Identify Site location', StartDate: new Date('04/02/2019'), Duration: 4, Progress: 50 },
{ TaskID: 3, TaskName: 'Perform Soil test', StartDate: new Date('04/02/2019'), Duration: 4, Progress: 50 },
{ TaskID: 4, TaskName: 'Soil test approval', StartDate: new Date('04/02/2019'), Duration: 4, Progress: 50 },
],
},
{
  TaskID: 5,
  TaskName: 'Project Estimation',
  StartDate: new Date('04/02/2019'),
  EndDate: new Date('04/21/2019'),
  subtasks: [
    { TaskID: 6, TaskName: 'Develop floor plan for estimation', StartDate: new Date('04/04/2019'), Duration: 3, Progress: 50 },
    { TaskID: 7, TaskName: 'List materials', StartDate: new Date('04/04/2019'), Duration: 3, Progress: 50 },
    { TaskID: 8, TaskName: 'Estimation approval', StartDate: new Date('04/04/2019'), Duration: 3, Progress: 50 }
  ]
},
],
height: '450px',
taskFields: {
  id: 'TaskID',
  name: 'TaskName',
  startDate: 'StartDate',
  endDate: 'EndDate',
  duration: 'Duration',
  progress: 'Progress',
  dependency: 'Predecessor',
  child: 'subtasks'
},
timelineSettings: {
  topTier: {
```

```

format: 'MMM dd, yyyy',
unit: 'Week',
},
bottomTier: {
unit: 'Day',
},
},
};
},
};
</script>
<style>
<!-- Material theme used for this sample -->
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-buttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-calendars/styles/material.css";
@import "../node_modules/@syncfusion/ej2-dropdowns/styles/material.css";
@import "../node_modules/@syncfusion/ej2-inputs/styles/material.css";
@import "../node_modules/@syncfusion/ej2-navigations/styles/material.css";
@import "../node_modules/@syncfusion/ej2-popups/styles/material.css";
@import "../node_modules/@syncfusion/ej2-splitbuttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-layouts/styles/material.css";
@import "../node_modules/@syncfusion/ej2-grids/styles/material.css";
@import "../node_modules/@syncfusion/ej2-treegrid/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-gantt/styles/material.css";
</style>
`

```

### Enable Toolbar

The [toolbar](#) property is used to add the toolbar items like Add, Remove, Edit, Update, Delete, Expand All, Collapse All in Gantt.

To use toolbar, inject the **Toolbar** module in the **provide** section.

```

<template>
<div>

```

```
<ejs-gantt ref='gantt' id="GanttContainer" :dataSource="data" :taskFields = "taskFields" :height =  
"height" :toolbar="toolbar" :editSettings= "editSettings"></ejs-gantt>  
</div>  
</template>  
<script>  
import { GanttComponent, Edit, Selection, Toolbar } from '@syncfusion/ej2-vue-gantt';  
export default {  
  name: "App",  
  components: {  
    'ejs-gantt' : GanttComponent  
  },  
  data: function() {  
    return{  
      data: [  
        {  
          TaskID: 1,  
          TaskName: 'Project Initiation',  
          StartDate: new Date('04/02/2019'),  
          EndDate: new Date('04/21/2019'),  
          subtasks: [  
            { TaskID: 2, TaskName: 'Identify Site location', StartDate: new Date('04/02/2019'), Duration: 4, Progress: 50 },  
            { TaskID: 3, TaskName: 'Perform Soil test', StartDate: new Date('04/02/2019'), Duration: 4, Progress: 50 },  
            { TaskID: 4, TaskName: 'Soil test approval', StartDate: new Date('04/02/2019'), Duration: 4, Progress: 50 },  
          ],  
        },  
        {  
          TaskID: 5,  
          TaskName: 'Project Estimation',  
          StartDate: new Date('04/02/2019'),  
          EndDate: new Date('04/21/2019'),  
          subtasks: [  

```



```
{ TaskID: 6, TaskName: 'Develop floor plan for estimation', StartDate: new Date('04/04/2019'), Duration: 3, Progress: 50 },
{ TaskID: 7, TaskName: 'List materials', StartDate: new Date('04/04/2019'), Duration: 3, Progress: 50 },
{ TaskID: 8, TaskName: 'Estimation approval', StartDate: new Date('04/04/2019'), Duration: 3, Progress: 50 }
]
},
],
height: '450px',
taskFields: {
  id: 'TaskID',
  name: 'TaskName',
  startDate: 'StartDate',
  endDate: 'EndDate',
  duration: 'Duration',
  progress: 'Progress',
  dependency: 'Predecessor',
  child: 'subtasks'
},
toolbar: ['Add', 'Edit', 'Update', 'Delete', 'Cancel', 'ExpandAll', 'CollapseAll'],
editSettings: {
  allowAdding: true,
  allowEditing: true,
  allowDeleting: true,
  allowTaskbarEditing: true,
  showDeleteConfirmDialog: true
},
};
},
provide: {
  gantt: [ Edit, Selection, Toolbar ]
}
};
</script>
```

```

<style>
<!-- Material theme used for this sample -->

@import "https://ej2.syncfusion.com/vue/documentation/node_modules/@syncfusion/ej2-
base/styles/material.css";

@import "../node_modules/@syncfusion/ej2-buttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-calendars/styles/material.css";
@import "../node_modules/@syncfusion/ej2-dropdowns/styles/material.css";
@import "../node_modules/@syncfusion/ej2-inputs/styles/material.css";
@import "../node_modules/@syncfusion/ej2-navigations/styles/material.css";
@import "../node_modules/@syncfusion/ej2-popups/styles/material.css";
@import "../node_modules/@syncfusion/ej2-splitbuttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-layouts/styles/material.css";
@import "../node_modules/@syncfusion/ej2-grids/styles/material.css";
@import "../node_modules/@syncfusion/ej2-treegrid/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-gantt/styles/material.css";

</style>
`

```

### Enabling editing

The editing feature enables you to edit the tasks in Gantt component. It can be enabled by using the [editSettings.allowEditing](#) and [editSettings.allowTaskbarEditing](#) properties.

To use Editing, inject the [Edit](#) module in the `provide` section.

The following editing options are available to update the tasks in Gantt:

- Cell
- Dialog
- Taskbar
- Connector line

### Cell editing

Modify the task details through cell editing by setting the `edit mode` property as `Auto`. To enable edit support [Edit](#) module should be injected in Gantt. If [Edit](#) module is not injected, you cannot do any editing action in Gantt.

```

<template>
<div>

<ejs-gantt ref='gantt' id="GanttContainer" :dataSource="data" :taskFields = "taskFields" :height =
"height" :toolbar="toolbar" :columns = "columns" :editSettings= "editSettings"></ejs-gantt>

```

```
</div>
</template>
<script>
import { GanttComponent, Edit } from '@syncfusion/ej2-vue-gantt';
export default {
  name: "App",
  components: {
    'ejs-gantt': GanttComponent
  },
  data: function() {
    return{
      data: [
        {
          TaskID: 1,
          TaskName: 'Project Initiation',
          StartDate: new Date('04/02/2019'),
          EndDate: new Date('04/21/2019'),
          subtasks: [
            { TaskID: 2, TaskName: 'Identify Site location', StartDate: new Date('04/02/2019'), Duration: 4, Progress: 50 },
            { TaskID: 3, TaskName: 'Perform Soil test', StartDate: new Date('04/02/2019'), Duration: 4, Progress: 50 },
            { TaskID: 4, TaskName: 'Soil test approval', StartDate: new Date('04/02/2019'), Duration: 4, Progress: 50 },
          ]
        },
        {
          TaskID: 5,
          TaskName: 'Project Estimation',
          StartDate: new Date('04/02/2019'),
          EndDate: new Date('04/21/2019'),
          subtasks: [
            { TaskID: 6, TaskName: 'Develop floor plan for estimation', StartDate: new Date('04/04/2019'), Duration: 3, Progress: 50 },
          ]
        }
      ]
    }
  }
}
```

```
{ TaskID: 7, TaskName: 'List materials', StartDate: new Date('04/04/2019'), Duration: 3, Progress: 50 },
{ TaskID: 8, TaskName: 'Estimation approval', StartDate: new Date('04/04/2019'), Duration: 3, Progress:
50 }
]
},
],
height: '450px',
taskFields: {
id: 'TaskID',
name: 'TaskName',
startDate: 'StartDate',
endDate: 'EndDate',
duration: 'Duration',
progress: 'Progress',
child: 'subtasks'
},
columns: [
{ field: 'TaskID', headerText: 'Task ID', textAlign: 'Left', width: '100' },
{ field: 'TaskName', headerText: 'Task Name', width: '250' },
{ field: 'StartDate', headerText: 'Start Date', width: '150' },
{ field: 'Duration', headerText: 'Duration', width: '150' },
{ field: 'Progress', headerText: 'Progress', width: '150' },
],
toolbar: ['Edit'],
editSettings: {
allowEditing: true,
mode: "Auto"
},
};
},
provide: {
ganttt: [ Edit ]
}
```

```

};
</script>
<style>
<!-- Material theme used for this sample -->
@import "https://ej2.syncfusion.com/vue/documentation/node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-buttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-calendars/styles/material.css";
@import "../node_modules/@syncfusion/ej2-dropdowns/styles/material.css";
@import "../node_modules/@syncfusion/ej2-inputs/styles/material.css";
@import "../node_modules/@syncfusion/ej2-navigations/styles/material.css";
@import "../node_modules/@syncfusion/ej2-popups/styles/material.css";
@import "../node_modules/@syncfusion/ej2-splitbuttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-layouts/styles/material.css";
@import "../node_modules/@syncfusion/ej2-grids/styles/material.css";
@import "../node_modules/@syncfusion/ej2-treegrid/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-gantt/styles/material.css";
</style>
`

```

**Note:** When the edit mode is set to **Auto**, you can change the cells to editable mode by double-clicking anywhere at the TreeGrid and edit the task details in the edit dialog by double-clicking anywhere at the chart.

#### *Dialog editing*

Modify the task details through dialog by setting edit [mode](#) property as **Dialog**.

```

<template>
<div>
<ejs-gantt ref='gantt' id="GanttContainer" :dataSource="data" :taskFields = "taskFields" :height =
"height" :toolbar="toolbar" :columns = "columns" :editSettings= "editSettings"></ejs-gantt>
</div>
</template>
<script>
import { GanttComponent, Edit } from '@syncfusion/ej2-vue-gantt';
export default {
name: "App",

```

```
components: {
  'ejs-gantt': GanttComponent
},
data: function() {
  return{
    data: [
      {
        TaskID: 1,
        TaskName: 'Project Initiation',
        StartDate: new Date('04/02/2019'),
        EndDate: new Date('04/21/2019'),
        subtasks: [
          { TaskID: 2, TaskName: 'Identify Site location', StartDate: new Date('04/02/2019'), Duration: 4, Progress: 50 },
          { TaskID: 3, TaskName: 'Perform Soil test', StartDate: new Date('04/02/2019'), Duration: 4, Progress: 50 },
          { TaskID: 4, TaskName: 'Soil test approval', StartDate: new Date('04/02/2019'), Duration: 4, Progress: 50 },
        ],
      },
      {
        TaskID: 5,
        TaskName: 'Project Estimation',
        StartDate: new Date('04/02/2019'),
        EndDate: new Date('04/21/2019'),
        subtasks: [
          { TaskID: 6, TaskName: 'Develop floor plan for estimation', StartDate: new Date('04/04/2019'), Duration: 3, Progress: 50 },
          { TaskID: 7, TaskName: 'List materials', StartDate: new Date('04/04/2019'), Duration: 3, Progress: 50 },
          { TaskID: 8, TaskName: 'Estimation approval', StartDate: new Date('04/04/2019'), Duration: 3, Progress: 50 },
        ],
      },
    ],
    height: '450px',
  }
}
```

```
taskFields: {
  id: 'TaskID',
  name: 'TaskName',
  startDate: 'StartDate',
  endDate: 'EndDate',
  duration: 'Duration',
  progress: 'Progress',
  child: 'subtasks'
},
columns: [
  { field: 'TaskID', headerText: 'Task ID', textAlign: 'Left', width: '100' },
  { field: 'TaskName', headerText: 'Task Name', width: '250' },
  { field: 'StartDate', headerText: 'Start Date', width: '150' },
  { field: 'Duration', headerText: 'Duration', width: '150' },
  { field: 'Progress', headerText: 'Progress', width: '150' },
],
toolbar: ['Edit'],
editSettings: {
  allowEditing: true,
  mode: "Dialog"
},
};
},
provide: {
  gantt: [ Edit ]
}
};
</script>
`
```

**Note:** In dialog editing mode, the edit dialog will appear while performing double click action in both TreeGrid and chart sides.

#### *Taskbar editing*

Modify the task details through user interaction such as resizing and dragging the taskbar by enabling the [allowTaskbarEditing](#) property.

```
`  
  
<template>  
<div>  
<ejs-gantt ref='gantt' id="GanttContainer" :dataSource="data" :taskFields = "taskFields" :height =  
"height" :toolbar="toolbar" :columns = "columns" :editSettings= "editSettings"></ejs-gantt>  
</div>  
</template>  
<script>  
import { GanttComponent, Edit } from '@syncfusion/ej2-vue-gantt';  
export default {  
  name: "App",  
  components: {  
    'ejs-gantt': GanttComponent  
  },  
  data: function() {  
    return{  
      data: [  
        {  
          TaskID: 1,  
          TaskName: 'Project Initiation',  
          StartDate: new Date('04/02/2019'),  
          EndDate: new Date('04/21/2019'),  
          subtasks: [  
            { TaskID: 2, TaskName: 'Identify Site location', StartDate: new Date('04/02/2019'), Duration: 4, Progress:  
              50 },  
            { TaskID: 3, TaskName: 'Perform Soil test', StartDate: new Date('04/02/2019'), Duration: 4, Progress: 50  
              },  
            { TaskID: 4, TaskName: 'Soil test approval', StartDate: new Date('04/02/2019'), Duration: 4, Progress: 50  
              },  
          ]  
        },  
        {  
          TaskID: 5,  
          TaskName: 'Project Estimation',
```



```
StartDate: new Date('04/02/2019'),
EndDate: new Date('04/21/2019'),
subtasks: [
  { TaskID: 6, TaskName: 'Develop floor plan for estimation', StartDate: new Date('04/04/2019'), Duration:
    3, Progress: 50 },
  { TaskID: 7, TaskName: 'List materials', StartDate: new Date('04/04/2019'), Duration: 3, Progress: 50 },
  { TaskID: 8, TaskName: 'Estimation approval', StartDate: new Date('04/04/2019'), Duration: 3, Progress:
    50 }
],
height: '450px',
taskFields: {
  id: 'TaskID',
  name: 'TaskName',
  startDate: 'StartDate',
  endDate: 'EndDate',
  duration: 'Duration',
  progress: 'Progress',
  child: 'subtasks'
},
columns: [
  { field: 'TaskID', headerText: 'Task ID', textAlign: 'Left', width: '100' },
  { field: 'TaskName', headerText: 'Task Name', width: '250' },
  { field: 'StartDate', headerText: 'Start Date', width: '150' },
  { field: 'Duration', headerText: 'Duration', width: '150' },
  { field: 'Progress', headerText: 'Progress', width: '150' },
],
toolbar: ['Edit'],
editSettings: {
  allowTaskbarEditing:true
},
};
```

```

provide: {
  gantt: [ Edit ]
}
};
</script>
<style>
<!-- Material theme used for this sample -->
@import "https://ej2.syncfusion.com/vue/documentation/node_modules/@syncfusion/ej2-
base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-buttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-calendars/styles/material.css";
@import "../node_modules/@syncfusion/ej2-dropdowns/styles/material.css";
@import "../node_modules/@syncfusion/ej2-inputs/styles/material.css";
@import "../node_modules/@syncfusion/ej2-navigations/styles/material.css";
@import "../node_modules/@syncfusion/ej2-popups/styles/material.css";
@import "../node_modules/@syncfusion/ej2-splitbuttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-layouts/styles/material.css";
@import "../node_modules/@syncfusion/ej2-grids/styles/material.css";
@import "../node_modules/@syncfusion/ej2-treegrid/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-gantt/styles/material.css";
</style>
`

```

### Dependency editing

Modify the task dependencies using mouse interactions by enabling the [allowTaskbarEditing](#) property along with mapping the task dependency data source field to the [dependency](#) property.

```

<template>
<div>
<ejs-gantt ref='gantt' id="GanttContainer" :dataSource="data" :taskFields = "taskFields" :height =
"height" :toolbar="toolbar" :columns = "columns" :editSettings= "editSettings"></ejs-gantt>
</div>
</template>
<script>
import { GanttComponent, Edit } from '@syncfusion/ej2-vue-gantt';

```

```
export default {
  name: "App",
  components: {
    'ejs-gantt': GanttComponent
  },
  data: function() {
    return{
      data: [
        {
          TaskID: 1,
          TaskName: 'Project Initiation',
          StartDate: new Date('04/02/2019'),
          EndDate: new Date('04/21/2019'),
          subtasks: [
            { TaskID: 2, TaskName: 'Identify Site location', StartDate: new Date('04/02/2019'), Duration: 0, Progress: 50 },
            { TaskID: 3, TaskName: 'Perform Soil test', StartDate: new Date('04/02/2019'), Duration: 4, Progress: 50 },
            { TaskID: 4, TaskName: 'Soil test approval', StartDate: new Date('04/02/2019'), Duration: 4, Predecessor: "2FS", Progress: 50 },
          ]
        },
        {
          TaskID: 5,
          TaskName: 'Project Estimation',
          StartDate: new Date('04/02/2019'),
          EndDate: new Date('04/21/2019'),
          subtasks: [
            { TaskID: 6, TaskName: 'Develop floor plan for estimation', StartDate: new Date('04/04/2019'), Duration: 3, Progress: 50 },
            { TaskID: 7, TaskName: 'List materials', StartDate: new Date('04/04/2019'), Duration: 3, Progress: 50 },
            { TaskID: 8, TaskName: 'Estimation approval', StartDate: new Date('04/04/2019'), Duration: 0, Predecessor: "6SS", Progress: 50 }
          ]
        },
      ]
    }
  }
}
```

```
],
height: '450px',
taskFields: {
  id: 'TaskID',
  name: 'TaskName',
  startDate: 'StartDate',
  endDate: 'EndDate',
  duration: 'Duration',
  progress: 'Progress',
  dependency: 'Predecessor',
  child: 'subtasks'
},
columns: [
  { field: 'TaskID', headerText: 'Task ID', textAlign: 'Left', width: '100' },
  { field: 'TaskName', headerText: 'Task Name', width: '250' },
  { field: 'StartDate', headerText: 'Start Date', width: '150' },
  { field: 'Duration', headerText: 'Duration', width: '150' },
  { field: 'Progress', headerText: 'Progress', width: '150' },
],
toolbar: ['Edit'],
editSettings: {
  allowTaskbarEditing: true
},
};
},
provide: {
  gantt: [ Edit ]
}
};
</script>
<style>
<!-- Material theme used for this sample -->
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
```

```

@import "../node_modules/@syncfusion/ej2-buttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-calendars/styles/material.css";
@import "../node_modules/@syncfusion/ej2-dropdowns/styles/material.css";
@import "../node_modules/@syncfusion/ej2-inputs/styles/material.css";
@import "../node_modules/@syncfusion/ej2-navigations/styles/material.css";
@import "../node_modules/@syncfusion/ej2-popups/styles/material.css";
@import "../node_modules/@syncfusion/ej2-splitbuttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-layouts/styles/material.css";
@import "../node_modules/@syncfusion/ej2-grids/styles/material.css";
@import "../node_modules/@syncfusion/ej2-treegrid/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-gantt/styles/material.css";
</style>

```

### Enabling predecessors or task relationships

Predecessor or task dependency in the Gantt component is used to depict the relationship between the tasks.

Start to Start (SS) : You cannot start a task until the dependent task starts.

Start to Finish (SF) : You cannot finish a task until the dependent task finishes.

Finish to Start (FS) : You cannot start a task until the dependent task completes.

Finish to Finish (FF) : You cannot finish a task until the dependent task completes.

You can show the relationship in tasks, by using the [dependency](#) property as shown in the following code example:

```

<template>
<div>
<ejs-gantt ref='gantt' id="GanttContainer" :dataSource="data" :taskFields = "taskFields" :height =
"height" highlightWeekends='true'></ejs-gantt>
</div>
</template>
<script>
import { GanttComponent } from '@syncfusion/ej2-vue-gantt';
export default {
name: "App",
components: {

```

```
'ejs-gantt' : GanttComponent
},
data: function() {
  return{
    data: [
      {
        TaskID: 1,
        TaskName: 'Project Initiation',
        StartDate: new Date('04/02/2019'),
        EndDate: new Date('04/21/2019'),
        subtasks: [
          { TaskID: 2, TaskName: 'Identify Site location', StartDate: new Date('04/02/2019'), Duration: 0, Progress: 50 },
          { TaskID: 3, TaskName: 'Perform Soil test', StartDate: new Date('04/02/2019'), Duration: 4, Progress: 50 },
          { TaskID: 4, TaskName: 'Soil test approval', StartDate: new Date('04/02/2019'), Duration: 4, Predecessor: "2FS", Progress: 50 },
        ]
      },
      {
        TaskID: 5,
        TaskName: 'Project Estimation',
        StartDate: new Date('04/02/2019'),
        EndDate: new Date('04/21/2019'),
        subtasks: [
          { TaskID: 6, TaskName: 'Develop floor plan for estimation', StartDate: new Date('04/04/2019'), Duration: 3, Progress: 50 },
          { TaskID: 7, TaskName: 'List materials', StartDate: new Date('04/04/2019'), Duration: 3, Progress: 50 },
          { TaskID: 8, TaskName: 'Estimation approval', StartDate: new Date('04/04/2019'), Duration: 0, Predecessor: "6SS", Progress: 50 },
        ]
      },
    ],
    height: '450px',
    taskFields: {
```

```

id: 'TaskID',
name: 'TaskName',
startDate: 'StartDate',
endDate: 'EndDate',
duration: 'Duration',
progress: 'Progress',
dependency: 'Predecessor',
child: 'subtasks'
},
};
},
};
</script>
<style>
<!-- Material theme used for this sample -->
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-buttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-calendars/styles/material.css";
@import "../node_modules/@syncfusion/ej2-dropdowns/styles/material.css";
@import "../node_modules/@syncfusion/ej2-inputs/styles/material.css";
@import "../node_modules/@syncfusion/ej2-navigations/styles/material.css";
@import "../node_modules/@syncfusion/ej2-popups/styles/material.css";
@import "../node_modules/@syncfusion/ej2-splitbuttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-layouts/styles/material.css";
@import "../node_modules/@syncfusion/ej2-grids/styles/material.css";
@import "../node_modules/@syncfusion/ej2-treegrid/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-gantt/styles/material.css";
</style>
`

```

### Assigning resources

You can display and assign the resource for each task in the Gantt control.

Create a collection of JSON object, which contains id, name, unit and group of the resources and assign it to the [resources](#) property.

Map these fields to the Gantt control using the [resourceFields](#) property.

```
,  
  
<template>  
<div>  
<ejs-gantt ref='gantt' id="GanttContainer" :dataSource="data" :taskFields= "taskFields" :height= "height"  
:treeColumnIndex= "1" :resourceFields= "resourceFields" :resources= "resources" :highlightWeekends=  
"true" :labelSettings= "labelSettings"></ejs-gantt>  
</div>  
</template>  
<script>  
import { GanttComponent } from '@syncfusion/ej2-vue-gantt';  
export default {  
  name: "App",  
  components: {  
    'ejs-gantt' : GanttComponent  
  },  
  data: function() {  
    return{  
      data: [  
        {  
          TaskID: 1,  
          TaskName: 'Project initiation',  
          StartDate: new Date('04/02/2019'),  
          EndDate: new Date('04/21/2019'),  
          subtasks: [  
            {TaskID: 2, TaskName: 'Identify site location', StartDate: new Date('04/02/2019'), Duration: 0,Progress:  
            50, resources: [1]},  
            {TaskID: 3, TaskName: 'Perform soil test', StartDate: new Date('04/02/2019'), Duration: 4, Predecessor:  
            '2',Progress: 50, resources: [2, 3, 5]},  
            {TaskID: 4, TaskName: 'Soil test approval', StartDate: new Date('04/02/2019'), Duration: 0, Predecessor:  
            '3', Progress: 50 },  
          ]  
        },  
        {  
          TaskID: 5,  
          TaskName: 'Project estimation',
```



```
StartDate: new Date('04/02/2019'),
EndDate: new Date('04/21/2019'),
subtasks: [
  {TaskID: 6, TaskName: 'Develop floor plan for estimation', StartDate: new Date('04/04/2019'),Duration:
  3, Predecessor: '4', Progress: 50, resources: [4]},
  {TaskID: 7, TaskName: 'List materials', StartDate: new Date('04/04/2019'),Duration: 3, Predecessor: '6',
  resources: [4, 8],Progress: 50},
  {TaskID: 8, TaskName: 'Estimation approval', StartDate: new Date('04/04/2019'),Duration: 0,
  Predecessor: '7', resources: [12, 5]}
]
},
taskFields: {
  id: 'TaskID',
  name: 'TaskName',
  startDate: 'StartDate',
  endDate: 'EndDate',
  duration: 'Duration',
  progress: 'Progress',
  dependency: 'Predecessor',
  child: 'subtasks',
  resourceInfo: 'resources'
},
height: '450px',
resourceFields: {
  id: 'resourceId',
  name: 'resourceName',
},
resources: [
  { resourceId: 1, resourceName: 'Martin Tamer' },
  { resourceId: 2, resourceName: 'Rose Fuller' },
  { resourceId: 3, resourceName: 'Margaret Buchanan' },
  { resourceId: 4, resourceName: 'Fuller King' },
  { resourceId: 5, resourceName: 'Davolio Fuller' },
```

```
{ resourceId: 6, resourceName: 'Van Jack' },
{ resourceId: 7, resourceName: 'Fuller Buchanan' },
{ resourceId: 8, resourceName: 'Jack Davolio' },
{ resourceId: 9, resourceName: 'Tamer Vinet' },
{ resourceId: 10, resourceName: 'Vinet Fuller' },
{ resourceId: 11, resourceName: 'Bergs Anton' },
{ resourceId: 12, resourceName: 'Construction Supervisor' }
],
labelSettings: {
  leftLabel: 'TaskName',
  rightLabel: 'resources'
},
};
}
};
</script>
<style>
<!-- Material theme used for this sample -->
@import "https://ej2.syncfusion.com/vue/documentation/node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-buttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-calendars/styles/material.css";
@import "../node_modules/@syncfusion/ej2-dropdowns/styles/material.css";
@import "../node_modules/@syncfusion/ej2-inputs/styles/material.css";
@import "../node_modules/@syncfusion/ej2-navigations/styles/material.css";
@import "../node_modules/@syncfusion/ej2-popups/styles/material.css";
@import "../node_modules/@syncfusion/ej2-splitbuttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-layouts/styles/material.css";
@import "../node_modules/@syncfusion/ej2-grids/styles/material.css";
@import "../node_modules/@syncfusion/ej2-treegrid/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-gantt/styles/material.css";
</style>
`
```

### Enable filtering

The filtering feature enables you to view reduced amount of records based on filter criteria. Gantt provides support for menu filtering support for each columns. It can be enabled by setting the [allowFiltering](#) property to `true` along with injecting the [Filter](#) module as shown in the following code example. Filtering feature can also be customized using the [filterSettings](#) property.

To use Filtering, inject the [Filter](#) module in the `provide` section.

```
,

<template>
<div>
<ejs-gantt ref='gantt' id="GanttContainer" :dataSource="data" :taskFields = "taskFields" :height =
"height" :columns="columns" :toolbar="toolbar" :allowFiltering= "true"
:timelineSettings="timelineSettings" :splitterSettings= "splitterSettings" :labelSettings= "labelSettings"
:projectStartDate="projectStartDate" :projectEndDate= "projectEndDate"></ejs-gantt>
</div>
</template>
<script>
import { GanttComponent, Filter, Toolbar } from '@syncfusion/ej2-vue-gantt';
export default {
name: "App",
components: {
'ejs-gantt' : GanttComponent
},
data: function() {
return{
data: [
{
TaskID: 1,
TaskName: 'Project Initiation',
StartDate: new Date('04/02/2019'),
EndDate: new Date('04/21/2019'),
subtasks: [
{ TaskID: 2, TaskName: 'Identify Site location', StartDate: new Date('04/02/2019'), Duration: 4, Progress:
50 },
{ TaskID: 3, TaskName: 'Perform Soil test', StartDate: new Date('04/02/2019'), Duration: 4, Progress: 50
},

```

```
{ TaskID: 4, TaskName: 'Soil test approval', StartDate: new Date('04/02/2019'), Duration: 4, Progress: 50
},
]
},
{
  TaskID: 5,
  TaskName: 'Project Estimation',
  StartDate: new Date('04/02/2019'),
  EndDate: new Date('04/21/2019'),
  subtasks: [
    { TaskID: 6, TaskName: 'Develop floor plan for estimation', StartDate: new Date('04/04/2019'), Duration:
    3, Progress: 50 },
    { TaskID: 7, TaskName: 'List materials', StartDate: new Date('04/04/2019'), Duration: 3, Progress: 50 },
    { TaskID: 8, TaskName: 'Estimation approval', StartDate: new Date('04/04/2019'), Duration: 3, Progress:
    50 }
  ]
},
],
height: '450px',
taskFields: {
  id: 'TaskID',
  name: 'TaskName',
  startDate: 'StartDate',
  endDate: 'EndDate',
  duration: 'Duration',
  dependency: 'Predecessor',
  child: 'subtasks',
},
columns: [
  { field: 'TaskName', headerText: 'Task Name', width: '250' , clipMode: 'EllipsisWithTooltip' },
  { field: 'StartDate', headerText: 'Start Date' },
  { field: 'Duration', headerText: 'Duration' },
  { field: 'EndDate', headerText: 'End Date' },
  { field: 'Predecessor', headerText: 'Predecessor' }
```

```
],
toolbar: ['Search'],
timelineSettings: {
  timelineUnitSize: 60,
  topTier: {
    format: 'MMM dd, yyyy',
    unit: 'Week',
  },
  bottomTier: {
    unit: 'Day',
  },
},
splitterSettings: {
  columnIndex: 3
},
labelSettings: {
  rightLabel: 'TaskName',
},
projectStartDate: new Date('04/01/2019 01:00:00 AM'),
projectEndDate: new Date('05/10/2019')
};
},
provide: {
  gantt: [ Filter, Toolbar ]
}
};
</script>
<style>
<!-- Material theme used for this sample -->
@import "https://ej2.syncfusion.com/vue/documentation/node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-buttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-calendars/styles/material.css";
```

```

@import "../node_modules/@syncfusion/ej2-dropdowns/styles/material.css";
@import "../node_modules/@syncfusion/ej2-inputs/styles/material.css";
@import "../node_modules/@syncfusion/ej2-navigations/styles/material.css";
@import "../node_modules/@syncfusion/ej2-popups/styles/material.css";
@import "../node_modules/@syncfusion/ej2-splitbuttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-layouts/styles/material.css";
@import "../node_modules/@syncfusion/ej2-grids/styles/material.css";
@import "../node_modules/@syncfusion/ej2-treegrid/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-gantt/styles/material.css";
</style>

```

,

### Enable Sorting

The sorting feature enables the user to order the records. It can be enabled by setting [allowSorting](#) property to true. Also, need to inject the `Sort` module in the `provide` section as follow. If we didn't inject the `Sort` module, then user not able to sort when click on headers. Sorting feature can be customized using [sortSettings](#) property.

,

```

<template>
<div>
<ejs-gantt ref='gantt' id="GanttContainer" :dataSource="data" :taskFields = "taskFields" :height =
"height" :columns="columns" :splitterSettings= "splitterSettings" :allowSorting= 'true'></ejs-gantt>
</div>
</template>
<script>
import { GanttComponent, Sort } from '@syncfusion/ej2-vue-gantt';
export default {
name: "App",
components: {
'ej2-gantt' : GanttComponent
},
data: function() {
return{
data: [
{
TaskID: 1,

```

```
TaskName: 'Project Initiation',
StartDate: new Date('04/02/2019'),
EndDate: new Date('04/21/2019'),
subtasks: [
  { TaskID: 2, TaskName: 'Identify Site location', StartDate: new Date('04/02/2019'), Duration: 4, Progress: 50 },
  { TaskID: 3, TaskName: 'Perform Soil test', StartDate: new Date('04/02/2019'), Duration: 4, Progress: 50 },
  { TaskID: 4, TaskName: 'Soil test approval', StartDate: new Date('04/02/2019'), Duration: 4, Progress: 50 },
],
{
  TaskID: 5,
  TaskName: 'Project Estimation',
  StartDate: new Date('04/02/2019'),
  EndDate: new Date('04/21/2019'),
  subtasks: [
    { TaskID: 6, TaskName: 'Develop floor plan for estimation', StartDate: new Date('04/04/2019'), Duration: 3, Progress: 50 },
    { TaskID: 7, TaskName: 'List materials', StartDate: new Date('04/04/2019'), Duration: 3, Progress: 50 },
    { TaskID: 8, TaskName: 'Estimation approval', StartDate: new Date('04/04/2019'), Duration: 3, Progress: 50 },
  ],
},
height: '450px',
taskFields: {
  id: 'TaskID',
  name: 'TaskName',
  startDate: 'StartDate',
  duration: 'Duration',
  progress: 'Progress',
  child: 'subtasks'
},
```

```
allowSorting: true,
splitterSettings: {
  columnIndex: 3
},
columns: [
  { field: 'TaskID', headerText: 'Task ID', textAlign: 'Left', width: '100' },
  { field: 'TaskName', headerText: 'Task Name', width: '250' },
  { field: 'StartDate', headerText: 'Start Date', width: '150' },
  { field: 'Duration', headerText: 'Duration', width: '150' },
  { field: 'Progress', headerText: 'Progress', width: '150' },
],
};
},
provide: {
  gantt: [ Sort ]
}
};
</script>
<style>
<!-- Material theme used for this sample -->
@import "https://ej2.syncfusion.com/vue/documentation/node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-buttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-calendars/styles/material.css";
@import "../node_modules/@syncfusion/ej2-dropdowns/styles/material.css";
@import "../node_modules/@syncfusion/ej2-inputs/styles/material.css";
@import "../node_modules/@syncfusion/ej2-navigations/styles/material.css";
@import "../node_modules/@syncfusion/ej2-popups/styles/material.css";
@import "../node_modules/@syncfusion/ej2-splitbuttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-layouts/styles/material.css";
@import "../node_modules/@syncfusion/ej2-grids/styles/material.css";
@import "../node_modules/@syncfusion/ej2-treegrid/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-gantt/styles/material.css";
```



```
</style>
```

```
,
```

### Defining eventmarkers

The [eventMarkers](#) property in Gantt component is used to highlight the important event in Gantt chart part. By using this feature, you can add the lines and label to highlight important days in your project.

To highlight the days, inject the `DayMarkers` module in the `provide` section.

```
,
```

```
<template>
```

```
<div>
```

```
<ejs-gantt ref='gantt' id="GanttContainer" :dataSource="data" :taskFields = "taskFields" :height =  
"height" :eventMarkers="eventMarkers"></ejs-gantt>
```

```
</div>
```

```
</template>
```

```
<script>
```

```
import { GanttComponent, DayMarkers } from '@syncfusion/ej2-vue-gantt';
```

```
export default {
```

```
  name: "App",
```

```
  components: {
```

```
    'ejs-gantt' : GanttComponent
```

```
  },
```

```
  data: function() {
```

```
    return{
```

```
      data: projectNewData,
```

```
      height: '450px',
```

```
      taskFields: {
```

```
        id: 'TaskID',
```

```
        name: 'TaskName',
```

```
        startDate: 'StartDate',
```

```
        endDate: 'EndDate',
```

```
        duration: 'Duration',
```

```
        progress: 'Progress',
```

```
        dependency: 'Predecessor',
```

```
        child: 'subtasks'
```

```
      },
```

```
eventMarkers: [  
  {  
    day: new Date('04/09/2019'),  
    label: 'Research phase'  
  }, {  
    day: new Date('04/30/2019'),  
    label: 'Design phase'  
  }, {  
    day: new Date('05/23/2019'),  
    label: 'Production phase'  
  }, {  
    day: new Date('06/20/2019'),  
    label: 'Sales and marketing phase'  
  }  
]  
};  
  
provide: {  
  gantt: [DayMarkers]  
}  
};  
</script>  
  
<style>  
  
<!-- Material theme used for this sample -->  
  
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";  
@import "../node_modules/@syncfusion/ej2-buttons/styles/material.css";  
@import "../node_modules/@syncfusion/ej2-calendars/styles/material.css";  
@import "../node_modules/@syncfusion/ej2-dropdowns/styles/material.css";  
@import "../node_modules/@syncfusion/ej2-inputs/styles/material.css";  
@import "../node_modules/@syncfusion/ej2-navigations/styles/material.css";  
@import "../node_modules/@syncfusion/ej2-popups/styles/material.css";  
@import "../node_modules/@syncfusion/ej2-splitbuttons/styles/material.css";  
@import "../node_modules/@syncfusion/ej2-layouts/styles/material.css";
```

```
@import "../node_modules/@syncfusion/ej2-grids/styles/material.css";
@import "../node_modules/@syncfusion/ej2-treegrid/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-gantt/styles/material.css";
</style>
```

### Running the application

Run the application using the following command.

```
`bash
npm run serve
```

Web server will be initiated, Open the quick start app in the browser at port `localhost:8080`.

### Module in Vue Gantt component

The modules that are available in Gantt are as follows.

Module	Description
<a href="#">Sort</a>	Inject this module to use sorting feature.
<a href="#">Filter</a>	Inject this module to use filtering feature.
<a href="#">Reorder</a>	Inject this module to use reorder feature.
<a href="#">ExcelExport</a>	Inject this module to use excel export feature.
<a href="#">PdfExport</a>	Inject this module to use PDF export feature.
<a href="#">RowDD</a>	Inject this module to use row drag and drop feature.
<a href="#">Resize</a>	Inject this module to use resize feature.
<a href="#">Toolbar</a>	Inject this module to use toolbar feature.
<a href="#">Edit</a>	Inject this module is use editing feature.
<a href="#">Selection</a>	Inject this module to use selection feature.
<a href="#">DayMarkers</a>	Inject this module to use event markers.
<a href="#">ContextMenu</a>	Inject this module to use context menu feature.
<a href="#">ColumnMenu</a>	Inject this module to use column menu feature.
<a href="#">VirtualScroll</a>	Inject this module to use virtual scroll feature.
<a href="#">CriticalPath</a>	Inject this module to use critical path feature.

These modules should be injected into the Gantt using the **Gantt.Inject** method.

## Data Binding

### Data binding in Vue Gantt component

The Gantt uses **DataManager**, which supports both RESTful JSON data services binding and local JavaScript object array binding. The [Link to the Video](#) property can be assigned either with the instance of DataManager or JavaScript object array collection. Gantt provides support to bind two kinds of data,

- Local data
- Remote data

To learn about Gantt Chart data binding quickly, you can check on this video:

#### Local data

To bind local data to Gantt, you can assign a JavaScript object array to the [dataSource](#) property. The local data source can also be provided as an instance of the **DataManager**.

In local data binding, the data source for rendering the Gantt component is retrieved from the same application locally.

The following are the two types of data binding possible with the Gantt component:

- Hierarchical data binding.
- Self-referential data binding (Flat data).

#### Hierarchical data binding

The [child](#) property is used to map the child records in hierarchical data.

The following code example shows how to bind the hierarchical local data into the Gantt component.

#### APP.VUE

```
<template>
  <div>
    <ejs-gantt ref='gantt' :dataSource="data" id="GanttContainer"
    :taskFields = "taskFields" :height = "height"></ejs-gantt>
  </div>
</template>
<script>
import Vue from "vue";
import { GanttPlugin } from "@syncfusion/ej2-vue-gantt";
Vue.use(GanttPlugin);
export default {
  data: function() {
    return {
      data: [
        {
          TaskID: 1,
          TaskName: 'Project Initiation',
          StartDate: new Date('04/02/2019'),
          EndDate: new Date('04/21/2019'),
          isParent:true,
          subtasks: [
            { TaskID: 2, TaskName: 'Identify Site location', StartDate:
new Date('04/02/2019'), Duration: 4, Progress: 50,isParent:false },
```

```

        { TaskID: 3, TaskName: 'Perform Soil test', StartDate: new
Date('04/02/2019'), Duration: 4, Progress: 50, resources: [2, 3,
5],isParent:false },
        { TaskID: 4, TaskName: 'Soil test approval', StartDate: new
Date('04/02/2019'), Duration: 4,Predecessor:"2FS", Progress:
50,isParent:false },
    ]
},
{
    TaskID: 5,
    TaskName: 'Project Estimation',
    StartDate: new Date('04/02/2019'),
    EndDate: new Date('04/21/2019'),
    isParent:true,
    subtasks: [
        { TaskID: 6, TaskName: 'Develop floor plan for estimation',
StartDate: new Date('04/04/2019'), Duration: 3, Progress: 50, resources:
[4],isParent:false },
        { TaskID: 7, TaskName: 'List materials', StartDate: new
Date('04/04/2019'), Duration: 3, Progress: 50, resources: [4,
8],isParent:false },
        { TaskID: 8, TaskName: 'Estimation approval', StartDate: new
Date('04/04/2019'), Duration: 3,Predecessor:"6SS", Progress: 50, resources:
[12, 5],isParent:false }
    ]
},
],
    height: '450px',
    taskFields: {
        id: 'TaskID',
        name: 'TaskName',
        startDate: 'StartDate',
        duration: 'Duration',
        progress: 'Progress',
        dependency: 'Predecessor',
        child: 'subtasks'
    }
};
},
};
</script>

```

{% previewsample "page.domainurl/code-snippet/gantt/data-binding-cs1" %}

#### Self-referential data binding (Flat data)

The Gantt component can be bound with self-referential data by mapping the data source field values to the [id](#) and [parentID](#) properties.

- ID field: This field contains unique values used to identify each individual task and it is mapped to the [id](#) property.
- Parent ID field: This field contains values that indicate parent tasks and it is mapped to the [parentID](#) property.

#### APP.VUE

```

<template>
  <div>
    <ejs-gantt ref='gantt' :dataSource="data" id="GanttContainer"
:taskFields = "taskFields" :height = "height" :treeColumnIndex='1'></ejs-
gantt>
  </div>
</template>
<script>
import Vue from "vue";
import { GanttPlugin } from "@syncfusion/ej2-vue-gantt";
Vue.use(GanttPlugin);
export default {
  data: function() {
    return{
      data: [
        { TaskID: 1,TaskName: 'Project Initiation',StartDate: new
Date('04/02/2019'),EndDate: new Date('04/21/2019')},
        { TaskID: 2, TaskName: 'Identify Site location', StartDate: new
Date('04/02/2019'), Duration: 4, Progress: 50,ParentId:1 },
        { TaskID: 3, TaskName: 'Perform Soil test', StartDate: new
Date('04/02/2019'), Duration: 4, Progress: 50, ParentId:1 },
        { TaskID: 4, TaskName: 'Soil test approval', StartDate: new
Date('04/02/2019'), Duration: 4, Progress: 50,ParentId:1 },
        { TaskID: 5, TaskName: 'Project Estimation',StartDate: new
Date('04/02/2019'),EndDate: new Date('04/21/2019')},
        { TaskID: 6, TaskName: 'Develop floor plan for estimation',
StartDate: new Date('04/04/2019'), Duration: 3, Progress: 50, ParentId:2 },
        { TaskID: 7, TaskName: 'List materials', StartDate: new
Date('04/04/2019'), Duration: 3, Progress: 50,ParentId:2 },
        { TaskID: 8, TaskName: 'Estimation approval', StartDate: new
Date('04/04/2019'), Duration: 3, Progress: 50, ParentId:2 }
      ],
      height: '450px',
      taskFields: {
        id: 'TaskID',
        name: 'TaskName',
        startDate: 'StartDate',
        duration: 'Duration',
        progress: 'Progress',
        parentID: 'ParentId'
      }
    };
  },
};
</script>

```

{% previewsample "page.domainurl/code-snippet/gantt/data-binding-cs2" %}

#### Remote data

To bind remote data to the Gantt component, assign service data as an instance of **DataManager** to the [dataSource](#) property.

#### APP.VUE

```

<template>
  <div>

```

```

        <ejs-gantt ref='gantt' :dataSource="data" id="GanttContainer"
:taskFields = "taskFields" :height = "height"></ejs-gantt>
    </div>
</template>
<script>
import Vue from "vue";
import { GanttPlugin } from "@syncfusion/ej2-vue-gantt";
import { DataManager, WebApiAdaptor } from '@syncfusion/ej2-data';
Vue.use(GanttPlugin);
export default {
    data: function() {
        return{
            data: new DataManager({
                url: 'https://ej2services.syncfusion.com/production/web-
services/api/GanttData',
                adaptor: new WebApiAdaptor,
                crossDomain: true
            }),
            height: '450px',
            taskFields: {
                id: 'TaskId',
                name: 'TaskName',
                startDate: 'StartDate',
                progress: 'Progress',
                duration: 'Duration',
                dependency: 'Predecessor',
                child: 'SubTasks'
            }
        };
    },
};
</script>

```

{% previewsample "page.domainurl/code-snippet/gantt/data-binding-cs3" %}

### URL Adaptor

In Gantt, we can fetch data from SQL database using **ADO.NET** Entity Data Model and update the changes on CRUD action to the server by using **DataManager** support. To communicate with the remote data we are using **UrlAdaptor** of **DataManager** property to call the server method and get back resultant data in JSON format. We can know more about **UrlAdaptor** from [here](#).

Please refer the [link](#) to create the **ADO.NET** Entity Data Model in Visual studio,

We can define data source for Gantt as instance of **DataManager** using **url** property of **DataManager**. Please Check the below code snippet to assign data source to Gantt.

,

```

<template>
<div>

<ejs-gantt ref='gantt' :dataSource="data" id="GanttContainer" :taskFields = "taskFields" :height =
"height"></ejs-gantt>

</div>

```

```

</template>
<script>
import Vue from "vue";
import { GanttPlugin } from "@syncfusion/ej2-vue-gantt";
import { DataManager, UrlAdaptor } from '@syncfusion/ej2-data';
Vue.use(GanttPlugin);
export default {
  data: function() {
    return{
      data: new DataManager({
        url: '/Home/UrlDatasource',
        adaptor: new UrlAdaptor
      }),
      height: '450px',
      taskFields: {
        id: 'TaskId',
        name: 'TaskName',
        startDate: 'StartDate',
        progress: 'Progress',
        duration: 'Duration',
        dependency: 'Predecessor',
        child: 'SubTasks'
      }
    };
  },
};
</script>
`ts
GanttDataSourceEntities db = new GanttDataSourceEntities();
public ActionResult UrlDatasource(DataManagerRequest dm)
{
  List<GanttData>DataList = db.GanttDatas.ToList();

```



```

var count = DataList.Count();
return Json(new { result = DataList, count = count });
}
`

```

#### Remote Save Adaptor

You may need to perform all Gantt Actions on the client-side except the CRUD operations, which should be interacted with the server-side to persist data. It can be achieved in Gantt by using **RemoteSaveAdaptor**.

Datasource must be set to the **json** property and set **RemoteSaveAdaptor** to the **adaptor** property. CRUD operations can be mapped to the server-side using the **batchUrl** properties.

You can use the following code example to use **RemoteSaveAdaptor** in Gantt.

```

`
<template>
<div>
<ejs-gantt ref='gantt' :dataSource="data" id="GanttContainer" :taskFields = "taskFields" :height =
"height"></ejs-gantt>
</div>
</template>
<script>
import Vue from "vue";
import { GanttPlugin, Edit } from "@syncfusion/ej2-vue-gantt";
import { DataManager, RemoteSaveAdaptor } from '@syncfusion/ej2-data';
Vue.use(GanttPlugin);
export default {
data: function() {
return{
data: new DataManager({
batchUrl: 'Home/BatchUpdate',
adaptor: new RemoteSaveAdaptor
}),
height: '450px',
taskFields: {
id: 'TaskId',
name: 'TaskName',
startDate: 'StartDate',

```

```

progress: 'Progress',
duration: 'Duration',
dependency: 'Predecessor',
child: 'SubTasks'
},
editSettings: {
allowAdding: true,
allowEditing: true,
allowDeleting: true,
allowTaskbarEditing: true,
showDeleteConfirmDialog: true
},
};
},
};
</script>
`

```

The following code example describes the CRUD operations handled at server-side.

```

public IActionResult BatchUpdate([FromBody] CRUDModel batchmodel)
{
    public class CRUDModel
    {
        public List<GanttDataSource> added { get; set; }
        public List<GanttDataSource> changed { get; set; }
        public List<GanttDataSource> deleted { get; set; }
        public object key { get; set; }
        public string action { get; set; }
        public string table { get; set; }
    }

    public IActionResult BatchUpdate([FromBody] CRUDModel batchmodel)
    {
        if (batchmodel.changed != null)

```

```
{
for (var i = 0; i < batchmodel.changed.Count(); i++)
{
var value = batchmodel.changed[i];
GanttDataSource result = DataList.Where(or => or.taskId == value.taskId).FirstOrDefault();
result.taskId = value.taskId;
result.taskName = value.taskName;
result.startDate = value.startDate;
result.endDate = value.endDate;
result.duration = value.duration;
result.progress = value.progress;
result.parentID = value.parentID;
}
}
if (batchmodel.deleted != null)
{
for (var i = 0; i < batchmodel.deleted.Count(); i++)
{
DataList.Remove(DataList.Where(or => or.taskId.Equals(batchmodel.deleted[i].taskId)).FirstOrDefault());
RemoveChildRecords(batchmodel.deleted[i].taskId);
}
}
if (batchmodel.added != null)
{
for (var i = 0; i < batchmodel.added.Count(); i++)
{
DataList.Add(batchmodel.added[i]);
}
}
return Json(new { addedRecords = batchmodel.added, changedRecords = batchmodel.changed,
deletedRecords = batchmodel.deleted });
}

public void RemoveChildRecords(int key)
```

```

{
var childList = DataList.Where(x => x.parentID == key).ToList();
foreach (var item in childList)
{
DataList.Remove(item);
RemoveChildRecords(item.taskId);
}
}

return Json(new { addedRecords = batchmodel.added, changedRecords = batchmodel.changed,
deletedRecords = batchmodel.deleted });
}
,

```

#### Load child on demand

To render child records on demand, assign a remote service URL in the instance of DataManager to the `Url` property. To interact with the remote data source, provide the endpoint URL and also define the [hasChildMapping](#) property in taskFields of Gantt Chart.

The `hasChildMapping` property maps the field name in the data source, which denotes whether the current record holds any child records. This is useful internally to show expand icon while binding child data on demand.

When [loadChildOnDemand](#) is disabled, all the root nodes are rendered in a collapsed state at initial load. On expanding the root node, the child nodes will be loaded from the remote server.

When `enableVirtualization` is enabled and `loadChildOnDemand` is disabled, only the current viewport root nodes are rendered in a collapsed state.

When a root node is expanded, its child nodes are rendered and maintained in a collection locally, such that on consecutive expand/collapse actions on the root node, the child nodes are loaded locally instead of from the remote server.

When the `loadChildOnDemand` is enabled, parent records are rendered in an expanded state.

```

<template>
<div id="app">
<ejs-gantt ref='gantt' id="GanttContainer"
:dataSource= "dataSource"
:taskFields= "taskFields"
:allowSelection= "true"
:allowResizing= "true"
:height= "height"

```

```
:treeColumnIndex= "1"
:highlightWeekends= "true"
:columns= "columns"
:enableVirtualization="true"
:loadChildOnDemand = "false"
:projectStartDate= "projectStartDate"
:projectEndDate= "projectEndDate"
:splitterSettings= "splitterSettings"

</ejs-gantt>
</div>
</template>
<script>
import Vue from "vue";
import { GanttPlugin, GanttComponent, Selection, VirtualScroll } from "@syncfusion/ej2-vue-gantt";
import { DataManager, WebApiAdaptor } from '@syncfusion/ej2-data';
Vue.use(GanttPlugin);
export default {
  data() {
    return {
      data: new DataManager({
        url: 'https://services.syncfusion.com/vue/production/api/GanttLoadOnDemand',
        adaptor: new WebApiAdaptor,
        crossDomain: true
      }),
      projectStartDate: new Date('01/02/2000'),
      projectEndDate: new Date('12/01/2002'),
      splitterSettings: {
        columnIndex: 3
      },
      taskFields: {
        id: 'taskId',
        name: 'taskName',
```

```

startDate: 'startDate',
endDate: 'endDate',
duration: 'duration',
progress: 'progress',
hasChildMapping: 'isParent',
parentID: 'parentID'
},
height: '460px',
columns: [
{ field: 'taskId' },
{ field: 'taskName', headerText: 'Task Name', width: '250', clipMode: 'EllipsisWithTooltip' },
{ field: 'startDate' },
{ field: 'duration' },
{ field: 'progress' }
]
};
},
provide: {
ganttt: [ Selection, VirtualScroll]
}
}
</script>
、

```

The following code example describes handling of Load on demand at server end.

```

`ts
public object Get()
{
    DataOperations operation = new DataOperations();
    var queryString = Request.Query;
    if (tree.Count == 0)
    tree = TreeData.GetTree();
    if (queryString.Keys.Contains("$filter") && !queryString.Keys.Contains("$stop"))
    {

```

```
StringValues filter;
queryString.TryGetValue("$filter", out filter);
int? fltr;
if (filter[0].ToString().Split("eq")[1] == " null")
{
    fltr = null;
}
else
{
    fltr = Int32.Parse(filter[0].ToString().Split("eq")[1]);
}
IQueryable<TreeData> data1 = tree.Where(f => f.parentID == fltr).AsQueryable();
return new { result = data1.ToList(), count = data1.Count() };
}

StringValues expand;
queryString.TryGetValue("$expand", out expand);
if (queryString.Keys.Contains("$expand")) // setting the ExpandStateMapping property whether is true
or false
{
    if (expand[0].ToString().Split(",")[0] == "ExpandingAction")
    {
        var val = TreeData.GetTree().Where(ds => ds.taskId ==
int.Parse(expand[0].ToString().Split(",")[1])).FirstOrDefault();
        val.IsExpanded = true;
    }
    else if (expand[0].ToString().Split(",")[0] == "CollapsingAction")
    {
        var val = TreeData.GetTree().Where(ds => ds.taskId ==
int.Parse(expand[0].ToString().Split(",")[1])).FirstOrDefault();
        val.IsExpanded = false;
    }
}

List<TreeData> data = tree.ToList();
if (queryString.Keys.Contains("$select"))
```

```

{
data = (from ord in tree
select new TreeData
{
parentID = ord.parentID
}
).ToList();
return data;
}

data = data.Where(p => p.parentID == null).ToList();
int count = data.Count;
if (queryString.Keys.Contains("$inlinecount"))
{
StringValues Skip;
StringValues Take;
StringValues loadchild;
int skip = (queryString.TryGetValue("$skip", out Skip)) ? Convert.ToInt32(Skip[0]) : 0;
int top = (queryString.TryGetValue("$top", out Take)) ? Convert.ToInt32(Take[0]) : data.Count();
var GroupData = TreeData.GetTree().ToList().GroupBy(rec => rec.parentID)
.Where(g => g.Key != null).ToDictionary(g => g.Key?.ToString(), g => g.ToList());
foreach (var Record in data.ToList())
{
if (GroupData.ContainsKey(Record.taskId.ToString()))
{
var ChildGroup = GroupData[Record.taskId.ToString()];
if (ChildGroup?.Count > 0)
AppendChildren(ChildGroup, Record, GroupData, data);
}
}
if (expand.Count > 0 && expand[0].ToString().Split(",")[0] == "CollapsingAction")
{
string IdMapping = "taskId";
List<WhereFilter> CollapseFilter = new List<WhereFilter>();

```



```

CollapseFilter.Add(new WhereFilter() { Field = IdMapping, value = expand[0].ToString().Split(",")[1],
Operator = "equal" });
var CollapsedParentRecord = operation.PerformFiltering(data, CollapseFilter, "and");
var index = data.Cast<object>().ToList().IndexOf(CollapsedParentRecord.Cast<object>().ToList()[0]);
skip = index;
}
else if (expand.Count > 0 && expand[0].ToString().Split(",")[0] == "ExpandingAction")
{
string IdMapping = "taskId";
List<WhereFilter> ExpandFilter = new List<WhereFilter>();
ExpandFilter.Add(new WhereFilter() { Field = IdMapping, value = expand[0].ToString().Split(",")[1],
Operator = "equal" });
var ExpandedParentRecord = operation.PerformFiltering(data, ExpandFilter, "and");
var index = data.Cast<object>().ToList().IndexOf(ExpandedParentRecord.Cast<object>().ToList()[0]);
skip = index;
}
return new { result = data.Skip(skip).Take(top), count = data.Count };
}
else
{
return TreeData.GetTree();
}
}

private void AppendChildren(List<TreeData> ChildRecords, TreeData ParentItem, Dictionary<string,
List<TreeData>> GroupData, List<TreeData> data)
{
var queryString = Request.Query;
string TaskId = ParentItem.taskId.ToString();
var index = data.IndexOf(ParentItem);
foreach (var Child in ChildRecords)
{
string ParentId = Child.parentID.ToString();
if (TaskId == ParentId && (bool)ParentItem.IsExpanded)
{

```

```

if (data.IndexOf(Child) == -1)
    ((IList)data).Insert(++index, Child);
if (GroupData.ContainsKey(Child.taskId.ToString()))
{
    var DeepChildRecords = GroupData[Child.taskId.ToString()];
    if (DeepChildRecords?.Count > 0)
        AppendChildren(DeepChildRecords, Child, GroupData, data);
}
}
}
}

// GET: api/Orders/
[HttpGet("{id}", Name = "Get")]
public string Get(int id)
{
    return "value";
}

[HttpPost]
public object Post([FromBody] TreeData[] value)
{
    //handle insert action
    for (var i = 0; i < value.Length; i++)
    {
        tree.Insert(0, value[i]);
    }
    return value;
}

//// PUT: api/Orders
[HttpPut]
public object Put([FromBody] TreeData[] value)
{
    //handle edit action
    if (value.Length == 1 && value[0].isParent == true)

```

```
{
UpdateDependentRecords(value[0]);
}
for (var i = 0; i < value.Length; i++) {
var ord = value[i];
TreeData val = tree.Where(or => or.taskId == ord.taskId).FirstOrDefault();
val.taskId = ord.taskId;
val.taskName = ord.taskName;
val.endDate = ord.endDate;
val.startDate = ord.startDate;
val.duration = ord.duration;
val.predecessor = ord.predecessor;
}
return value;
}

private void UpdateDependentRecords(TreeData ParentItem)
{
var data = tree.Where(p => p.parentID == ParentItem.taskId).ToList();
var previousData = tree.Where(p => p.taskId == ParentItem.taskId).ToList();
var previousStartDate = previousData[0].startDate;
var previousEndDate = previousData[0].endDate;
double sdiff = (double)GetTimeDifference((DateTime)previousStartDate,
(DateTime)ParentItem.startDate);
double ediff = (double)GetTimeDifference((DateTime)previousEndDate,
(DateTime)ParentItem.endDate);
GetRootChildRecords(ParentItem);
for(var i=0; i<ChildRecords.Count;i++)
{
ChildRecords[i].startDate = ((DateTime)ChildRecords[i].startDate).AddSeconds(sdiff);
ChildRecords[i].endDate = ((DateTime)ChildRecords[i].endDate).AddSeconds(ediff);
}
}

private void GetRootChildRecords(TreeData ParentItem)
{

```

```

var currentchildRecords = tree.Where(p => p.parentID == ParentItem.taskId).ToList();
for (var i = 0; i < currentchildRecords.Count; i++) {
    var currentRecord = currentchildRecords[i];
    ChildRecords.Add(currentRecord);
    if (currentRecord.isParent == true)
    {
        GetRootChildRecords(currentRecord);
    }
}

public object GetTimeDifference(DateTime sdate, DateTime edate)
{
    return new DateTime(edate.Year, edate.Month, edate.Day, edate.Hour, edate.Minute, edate.Second,
        DateTimeKind.Utc).Subtract(new DateTime(sdate.Year, sdate.Month, sdate.Day, sdate.Hour,
        sdate.Minute, sdate.Second, DateTimeKind.Utc)).TotalSeconds;
}

// DELETE: api/ApiWithActions
[HttpDelete("{id:int}")]
[Route("Orders/{id:int}")]
public object Delete(int id)
{
    //handle delete action
    tree.Remove(tree.Where(or => or.taskId == id).FirstOrDefault());
    return Json(id);
}

public class CRUDModel<T> where T : class
{
    public TreeData Value;
    public int Key { get; set; }
    public int RelationalKey { get; set; }
    public List<TreeData> added { get; set; }
    public List<TreeData> changed { get; set; }
    public List<TreeData> deleted { get; set; }
}

```

```
public class TreeData
{
    public static List<TreeData> tree = new List<TreeData>();
    [System.ComponentModel.DataAnnotations.Key]
    public int taskId { get; set; }
    public string taskName { get; set; }
    public DateTime startDate { get; set; }
    public DateTime endDate { get; set; }
    public string duration { get; set; }
    public int progress { get; set; }
    public int? parentId { get; set; }
    public string predecessor { get; set; }
    public bool? isParent { get; set; }
    public bool? IsExpanded { get; set; }
    public static List<TreeData> GetTree()
    {
        if (tree.Count == 0)
        {
            Random rand = new Random();
            var x = 0;
            int duration = 0;
            DateTime startDate = new DateTime(2000, 1, 3, 08, 00, 00);
            for (var i = 1; i <= 50; i++)
            {
                startDate = startDate.AddDays(i == 1 ? 0 : 7);
                DateTime childStartDate = startDate;
                TreeData Parent = new TreeData()
                {
                    taskId = ++x,
                    taskName = "Task " + x,
                    startDate = startDate,
                    endDate = childStartDate.AddDays(26),
                    duration = "20",
```

```

progress = rand.Next(100),
predecessor = null,
isParent = true,
parentID = null,
IsExpanded = false
};
tree.Add(Parent);
for (var j = 1; j <= 4; j++)
{
    childStartDate = childStartDate.AddDays(j == 1 ? 0 : duration + 2);
    duration = 5;
    tree.Add(new TreeData()
    {
        taskId = ++x,
        taskName = "Task " + x,
        startDate = childStartDate,
        endDate = childStartDate.AddDays(5),
        duration = duration.ToString(),
        progress = rand.Next(100),
        parentID = Parent.taskId,
        predecessor = (j > 1 ? (x - 1) + "FS" : ""),
        isParent = false,
        IsExpanded = false
    });
}
}
}
return tree;
}
}
`

```

### Limitations

- Filtering, sorting and searching are not supported in load on demand.

- Only Self-Referential type data is supported with remote data binding in Gantt Chart.
- Load-on-demand supports only the validated data source

#### [Sending additional parameters to the server](#)

We can pass additional parameters using `addParams` method of `Query` class. In server side we have inherited and shown the additional parameter value in Syncfusion DataManager class itself. We pass an additional parameter in load time using [load](#) event. We can also pass additional parameter to the CRUD model. Please Check the below code snippet to send additional parameter to Gantt.

`

```
<template>
<div>

<ejs-gantt ref='gantt' :dataSource="data" id="GanttContainer" :taskFields = "taskFields" :height =
"height" :toolbar="toolbar" :editSettings= "editSettings" :load= "load"></ejs-gantt>

</div>
</template>
<script>
import Vue from "vue";
import { GanttPlugin, Edit, Toolbar } from "@syncfusion/ej2-vue-gantt";
import { DataManager, UrlAdaptor, Query } from '@syncfusion/ej2-data';
Vue.use(GanttPlugin);
export default {
  data: function() {
    return{
      data: new DataManager({
        url: 'http://localhost:50039/Home/UrlDatasource',
        adaptor: new UrlAdaptor,
        batchUrl: 'http://localhost:50039/Home/BatchSave'
      }),
      height: '450px',
      taskFields: {
        id: 'TaskId',
        name: 'TaskName',
        startDate: 'StartDate',
        progress: 'Progress',
        duration: 'Duration',
        dependency: 'Predecessor',
```

```

parentID: 'parentID'
},
toolbar: ['Add', 'Edit', 'Delete', 'Cancel', 'ExpandAll', 'CollapseAll', 'Update'],
editSettings: {
allowAdding: true,
allowEditing: true,
allowDeleting: true
},
};
load: function(args) {
this.query = new Query().addParams('ej2Gantt', "test");
}
},
provide: {
gant: [ Edit, Toolbar ]
}
};
</script>
`ts
namespace URLAdaptor.Controllers
{
public class HomeController : Controller
{
...///
//inherit the class to show age as property of DataManager
public class Test : DataManagerRequest
{
public string ej2Gantt { get; set; }
}
public ActionResult UrlDatasource([FromBody]Test dm)
{
if (DataList == null)

```



```

{
ProjectData datasource = new ProjectData();
DataList = datasource.GetUrlDataSource();
}

var count = DataList.Count();

return Json(new { result = DataList, count = count }, JsonRequestBehavior.AllowGet);
}

...///

public class ICRUDModel<T> where T : class
{
public object key { get; set; }
public T value { get; set; }
public List<T> added { get; set; }
public List<T> changed { get; set; }
public List<T> deleted { get; set; }
public IDictionary<string, object> @params { get; set; }
}

...///
}
}
`

```

You can find the full sample from [here](#).

#### Handling HTTP error

During server interaction from the Gantt, some server-side exceptions may occur, and you can acquire those error messages or exception details in client-side using the [actionFailure](#) event.

The argument passed to the `actionFailure` event contains the error details returned from the server.

#### APP.VUE

```

<template>
  <div>
    <ejs-gantt ref='gantt' :dataSource="data" id="GanttContainer"
:taskFields = "taskFields" :height = "height" :actionFailure=
"actionFailure"></ejs-gantt>
  </div>
</template>
<script>
import Vue from "vue";
import { GanttPlugin } from "@syncfusion/ej2-vue-gantt";
import { DataManager } from '@syncfusion/ej2-data';

```

```

Vue.use(GanttPlugin);
export default {
  data: function() {
    return{
      data: new DataManager({
        url: 'http://some.com/invalidUrl',
      }),
      height: '450px',
      taskFields: {
        id: 'TaskId',
        name: 'TaskName',
        startDate: 'StartDate',
        progress: 'Progress',
        duration: 'Duration',
        dependency: 'Predecessor',
        parentID: 'parentID'
      },

      actionFailure: function(args){
        let gantt = document.getElementsByClassName("e-gantt")[0].ej2_instances[0]; // Gantt instance
        let span = document.createElement('span');
        this.element.parentNode.insertBefore(span, gantt.element);
        span.style.color = '#FF0000'
        span.innerHTML = 'Server exception: 404 Not found';
      },
    };
  }
};
</script>

```

{% previewsample "page.domainurl/code-snippet/gantt/data-binding-cs4" %}

### Binding with Ajax

You can use Gantt [dataSource](#) property to bind the data source to Gantt from external Ajax request. In the below code we have fetched the data source from the server with the help of Ajax request and provided that to `dataSource` property by using `onSuccess` event of the Ajax.

### APP.VUE

```

<template>
  <div>
    <ejs-button id="databind" cssClass="e-info" v-on:click.native="databind">Bind Data</ejs-button>
    <br>
    <br>
    <ejs-gantt ref='gantt' :dataSource="data" id="GanttContainer"
      :taskFields = "taskFields" :height = "height"
      :projectStartDate="projectStartDate" :projectEndDate=
      "projectEndDate"></ejs-gantt>
    </div>
  </template>
<script>
import Vue from "vue";
import { GanttPlugin } from "@syncfusion/ej2-vue-gantt";
import { Ajax } from '@syncfusion/ej2-base';
Vue.use(GanttPlugin);

```

```

export default {
  data: function() {
    return{
      height: '450px',
      taskFields: {
        id: 'TaskId',
        name: 'TaskName',
        startDate: 'StartDate',
        progress: 'Progress',
        duration: 'Duration',
        dependency: 'Predecessor',
        child: 'SubTasks'
      },
      projectStartDate: new Date('02/24/2019'),
      projectEndDate: new Date('07/20/2019')
    };
  },
  methods: {
    databind: function(e){
      let ajax = new
      Ajax("https://ej2services.syncfusion.com/production/web-
      services/api/GanttData", "GET");
      this.showSpinner();
      ajax.send();
      ajax.onSuccess = function (data) {
        this.hideSpinner();
        this.dataSource = (JSON.parse(data)).Items;
        this.refresh();
      };
    },
  }
};
</script>

```

{% previewsample "page.domainurl/code-snippet/gantt/data-binding-cs5" %}

Note: If you bind the dataSource from this way, then it acts like a local dataSource. So you cannot perform any server side crud actions.

### Split task

The **Split-task** feature allows you to split a task or interrupt the work during planned or unforeseen circumstances. We can split the task either in load time or dynamically, by defining the segments either in hierarchical or self-referential way.

### Hierarchical

To split a task at load time in hierarchical way, we need to define the segment details in datasource and this field should be mapped by using the [taskFields.segments](#) property.

```
`ts
```

```
[
```

```
{
```

```
TaskID: 1, TaskName: 'Identify Site location', StartDate: new Date('04/02/2019'), Duration: 4, Progress: 50,
```

Segments: [

{ StartDate: new Date("04/02/2019"), Duration: 2 },

{ StartDate: new Date("04/04/2019"), Duration: 2 }

]

}

]

,

### APP.VUE

```
<template>
  <div>
    <ejs-gantt ref='gantt' id="GanttContainer" :dataSource="data"
:taskFields = "taskFields" :height = "height"></ejs-gantt>
  </div>
</template>
<script>
import Vue from "vue";
import { GanttPlugin } from "@syncfusion/ej2-vue-gantt";
Vue.use(GanttPlugin);
export default {
  data: function() {
    return{
      data: [
        {
          TaskID: 1,
          TaskName: 'Project Initiation',
          StartDate: new Date('04/02/2019'),
          EndDate: new Date('04/21/2019'),
          subtasks: [
            { TaskID: 2, TaskName: 'Identify Site location', StartDate:
new Date('04/02/2019'), Duration: 4, Progress: 50,
              Segments: [
                { StartDate: new Date("04/02/2019"), Duration: 2 },
                { StartDate: new Date("04/04/2019"), Duration: 2 }
              ] },
            { TaskID: 3, TaskName: 'Perform Soil test', StartDate: new
Date('04/02/2019'), Duration: 4, Progress: 50 },
            { TaskID: 4, TaskName: 'Soil test approval', StartDate: new
Date('04/02/2019'), Duration: 4 , Progress: 50 },
          ]
        },
        {
          TaskID: 5,
          TaskName: 'Project Estimation',
          StartDate: new Date('04/02/2019'),
          EndDate: new Date('04/21/2019'),
          subtasks: [
            { TaskID: 6, TaskName: 'Develop floor plan for estimation',
StartDate: new Date('04/04/2019'), Duration: 3, Progress: 50 },
            { TaskID: 7, TaskName: 'List materials', StartDate: new
Date('04/04/2019'), Duration: 3, Progress: 50 },
          ]
        }
      ]
    }
  }
}
```

```

        { TaskID: 8, TaskName: 'Estimation approval', StartDate: new
Date('04/04/2019'), Duration: 3, Progress: 50 }
    ]
  },
  ],
  height: '450px',
  taskFields: {
    id: 'TaskID',
    name: 'TaskName',
    startDate: 'StartDate',
    endDate: 'EndDate',
    duration: 'Duration',
    progress: 'Progress',
    child: 'subtasks',
    segments: 'Segments'
  }
};
}
};
</script>

```

{% previewsample "page.domainurl/code-snippet/gantt/data-binding-cs6" %}

#### Self-referential

We can also define segment details as a flat data and this collection can be mapped by using [segmentData](#) property. The segment id field of this collection is mapped by using the [taskFields.segmentId](#) property.

`ts

```

taskFields: {
  segmentId: "segmentId"
},
segmentData: [
  { segmentId: 1, StartDate: new Date("02/04/2019"), Duration: 2 },
  { segmentId: 1, StartDate: new Date("02/05/2019"), Duration: 5 },
  { segmentId: 4, StartDate: new Date("04/02/2019"), Duration: 2 },
  { segmentId: 4, StartDate: new Date("04/04/2019"), Duration: 2 }
],
`

```

#### APP.VUE

```

<template>
  <div>
    <ejs-gantt ref='gantt' id="GanttContainer" :dataSource="data"
:taskFields = "taskFields" :height = "height" :segmentData =
"segmentData"></ejs-gantt>
  </div>
</template>

```

```

<script>
import Vue from "vue";
import { GanttPlugin } from "@syncfusion/ej2-vue-gantt";
Vue.use(GanttPlugin);
export default {
  data: function() {
    return{
      data: [
        {
          TaskID: 1,
          TaskName: 'Project Initiation',
          StartDate: new Date('04/02/2019'),
          EndDate: new Date('04/21/2019'),
          subtasks: [
            { TaskID: 2, TaskName: 'Identify Site location', StartDate:
new Date('04/02/2019'), Duration: 4, Progress: 50,
              Segments: [
                { StartDate: new Date("04/02/2019"), Duration: 2 },
                { StartDate: new Date("04/04/2019"), Duration: 2 }
              ],
            { TaskID: 3, TaskName: 'Perform Soil test', StartDate: new
Date('04/02/2019'), Duration: 4, Progress: 50 },
            { TaskID: 4, TaskName: 'Soil test approval', StartDate: new
Date('04/02/2019'), Duration: 4 , Progress: 50 },
          ]
        },
        {
          TaskID: 5,
          TaskName: 'Project Estimation',
          StartDate: new Date('04/02/2019'),
          EndDate: new Date('04/21/2019'),
          subtasks: [
            { TaskID: 6, TaskName: 'Develop floor plan for estimation',
StartDate: new Date('04/04/2019'), Duration: 3, Progress: 50 },
            { TaskID: 7, TaskName: 'List materials', StartDate: new
Date('04/04/2019'), Duration: 3, Progress: 50 },
            { TaskID: 8, TaskName: 'Estimation approval', StartDate: new
Date('04/04/2019'), Duration: 3, Progress: 50 }
          ]
        }
      ],
      height: '450px',
      taskFields: {
        id: 'TaskID',
        name: 'TaskName',
        startDate: 'StartDate',
        endDate: 'EndDate',
        duration: 'Duration',
        progress: 'Progress',
        child: 'subtasks',
        segments: 'Segments'
      },
      segmentData: [
        { segmentId: 2, StartDate: new Date("04/02/2019"), Duration:
2 },
        { segmentId: 2, StartDate: new Date("04/04/2019"), Duration:
2 },

```

```

                { segmentId: 4, StartDate: new Date("04/02/2019"), Duration:
2 },
                { segmentId: 4, StartDate: new Date("04/04/2019"), Duration:
2 }
            ]
        };
    }
};
</script>

```

{% previewsample "page.domainurl/code-snippet/gantt/data-binding-cs7" %}

Note: Segment id field contains id of a task which should be split at load time.

#### *Improve performance by disabling validations*

The [autoCalculateDateScheduling](#) property can help you reduce the time taken for the Gantt chart to render on the initial load. When this API is enabled, parent-child validation, data validation, and predecessor validation are restricted, allowing the Gantt chart to load more quickly. Since we are disabling the validations, data source provided to gantt should have all data such as start date, end date, duration, as proper data.

#### **APP.VUE**

```

<template>
</template>
<script>
import Vue from "vue";
import { GanttPlugin, Selection, VirtualScroll, Edit } from "@syncfusion/ej2-vue-gantt";
import {tempData} from "../datasource.js";
Vue.use(GanttPlugin);
var virtualData = [];
var projId = 1;
for (var i = 0; i < 50; i++) {
    var x = virtualData.length + 1;
    var parent_1 = {};
    parent_1['TaskID'] = x;
    parent_1['TaskName'] = 'Project' + (projId++);
    virtualData.push(parent_1);
    for (var j = 0; j < tempData.length; j++) {
        var subtasks = {};
        subtasks['TaskID'] = tempData[j].TaskID + x;
        subtasks['TaskName'] = tempData[j].TaskName;
        subtasks['StartDate'] = tempData[j].StartDate;
        subtasks['Duration'] = tempData[j].Duration;
        subtasks['Progress'] = tempData[j].Progress;
        subtasks['parentID'] = tempData[j].parentID + x;
        virtualData.push(subtasks);
    }
}
new Vue({
  el: '#app',
  template: `
    <div>
      <ejs-gantt ref='gantt'
        :dataSource= "data"

```

```

        :taskFields= "taskFields"
        :allowSelection= "true"
        :enableVirtualization= "true"
        :labelSettings= "labelSettings"
        :height= "height"
        :treeColumnIndex= "1"
        :highlightWeekends= "true"
        :columns= "columns"
        :autoCalculateDateScheduling='false',
        :editSettings= "editSettings"
        :splitterSettings= "splitterSettings">
    </ejs-gantt>
</div>
`,
data: function() {
    return{
data : virtualData,
        taskFields: {
            id: 'TaskID',
            name: 'TaskName',
            startDate: 'StartDate',
            endDate: 'EndDate',
            duration: 'Duration',
            progress: 'Progress',
            parentID: 'parentID'
        },
        columns: [
            { field: 'TaskID' },
            { field: 'TaskName' },
            { field: 'StartDate' },
            { field: 'Duration' },
            { field: 'Progress' }
        ],
        height: '450px',
        labelSettings: {
            taskLabel: 'Progress'
        },
        splitterSettings: {
            columnIndex: 2
        },
        editSettings:{
            allowAdding:true
        },
    }
}
,
provide: {
    gantt: [ Selection, VirtualScroll,Edit]
}
});
</script>

```

### Limitations

Gantt has the support for both Hierarchical and Self-Referential data binding. When rendering the Gantt control with SQL database, we suggest you to use the Self-Referential data binding to maintain the



parent-child relation. Because the complex json structure is very difficult to manage it in SQL tables, we need to write a complex queries and we have to write a complex algorithm to find out the proper record details while updating/deleting the inner level task in Gantt data source. We cannot implement both data binding for Gantt control and this is not a recommended way. If both child and parentID are mapped, the records will not render properly because, when task id of a record defined in the hierarchy structure is assigned to parent id of another record, in such case the records will not properly render. As the self-referential will search the record with particular id in flat data only, not in the inner level of records. If we map the parentID field, it will be prioritized and Gantt will be rendered based on the parentID values.

### Immutable in Vue Gantt component

The immutable mode optimizes the Gantt re-rendering performance by using the object reference and [deep compare](#) concept. When performing the Gantt actions, it will only re-render the modified or newly added rows and prevent the re-rendering of the unchanged rows.

To enable this feature, you have to set the [enableImmutableMode](#) property as **true**.

This feature uses the primary key value for data comparison. So, you need to provide the [isPrimaryKey](#) column.

### APP.VUE

```
<template>
  <div>
    <ejs-gantt ref='gantt' id="GanttContainer" :dataSource="data"
    :taskFields = "taskFields" :height = "height" :toolbar="toolbar"
    :editSettings= "editSettings" :enableImmutableMode='true'></ejs-
gantt>
  </div>
</template>
<script>
import Vue from "vue";
import { GanttPlugin, Edit, Selection, Toolbar } from "@syncfusion/ej2-vue-gantt";
import { projectNewData } from './data-source.js';
Vue.use(GanttPlugin);
export default {
  data: function() {
    return{
      data: projectNewData,
      height: '450px',
      taskFields: {
        id: 'TaskID',
        name: 'TaskName',
        startDate: 'StartDate',
        duration: 'Duration',
        progress: 'Progress',
        child: 'subtasks'
      },
      toolbar: ['Add', 'Edit', 'Delete', 'Cancel', 'Update', 'Indent',
'Outdent'],
      editSettings: {
        allowAdding: true,
        allowEditing: true,
        allowDeleting: true,
        allowTaskbarEditing: true,

```

```

        showDeleteConfirmDialog: true
    },
    },
    provide: {
        gantt: [ Edit, Selection, Toolbar]
    }
};
</script>

```

{% previewsample "page.domainurl/code-snippet/gantt/immutable-cs1" %}

### Limitations

The following features are not supported in the immutable mode:

- Column reorder
- Virtualization

### Loading animation in Vue Gantt component

The loading indicator is used to display a visual indicator while the Gantt is fetching data or performing certain actions, such as sorting or filtering. The gantt support two indicator types, which is achieved by setting the [loadingIndicator.indicatorType](#) property to Shimmer or Spinner. The default value of the indicator type is "Spinner."

In the following sample, the Shimmer indicator is displayed while the gantt is scrolled when using the virtual data.

### APP.VUE

```

<template>
</template>
<script>
import Vue from "vue";
import { GanttPlugin, Selection, VirtualScroll, Sort, Filter } from
"@syncfusion/ej2-vue-gantt";
import{tempData} from "./datasource.js";
Vue.use(GanttPlugin);
var virtualData = [];
var projId = 1;
for (var i = 0; i < 50; i++) {
    var x = virtualData.length + 1;
    var parent_1 = {};
    parent_1['TaskID'] = x;
    parent_1['TaskName'] = 'Project' + (projId++);
    virtualData.push(parent_1);
    for (var j = 0; j < tempData.length; j++) {
        var subtasks = {};
        subtasks['TaskID'] = tempData[j].TaskID + x;
        subtasks['TaskName'] = tempData[j].TaskName;
        subtasks['StartDate'] = tempData[j].StartDate;
        subtasks['Duration'] = tempData[j].Duration;
        subtasks['Progress'] = tempData[j].Progress;
        subtasks['parentID'] = tempData[j].parentID + x;
        virtualData.push(subtasks);
    }
}

```

```

new Vue({
  el: '#app',
  template: `
    <div>
      <ejs-gantt ref='gantt'
        :dataSource= "data"
        :taskFields= "taskFields"
        :allowSelection= "true"
        :allowSorting= "true"
        :allowFiltering= "true"
        :enableVirtualization= "true"
        :labelSettings= "labelSettings"
        :height= "height"
        :treeColumnIndex= "1"
        :highlightWeekends= "true"
        :columns= "columns"
        :splitterSettings= "splitterSettings">
      </ejs-gantt>
    </div>
  `,
  data: function() {
    return{
      data : virtualData,
      taskFields: {
        id: 'TaskID',
        name: 'TaskName',
        startDate: 'StartDate',
        endDate: 'EndDate',
        duration: 'Duration',
        progress: 'Progress',
        parentID: 'parentID'
      },
      allowSorting:true,
      allowFiltering:true,
      columns: [
        { field: 'TaskID' },
        { field: 'TaskName' },
        { field: 'StartDate' },
        { field: 'Duration' },
        { field: 'Progress' }
      ],
      height: '450px',
      labelSettings: {
        taskLabel: 'Progress'
      },
      splitterSettings: {
        columnIndex: 2
      }
    }
  },
  provide: {
    gantt: [ Selection, VirtualScroll, Sort, Filter]
  }
});
</script>

```

```
{% previewsample "page.domainurl/code-snippet/gantt/loading-animation-cs1" %}
```

## Columns

### Columns in Vue Gantt component

The column displays information from a bound data source, and you can edit the values of column to update the task details through TreeGrid. The operations such as sorting, filtering, and searching can be performed based on column definitions. To display a Gantt column, the [field](#) property should be mapped from the data source to the column.

If the column [field](#) is not specified in the dataSource, the column values will be empty.

The [Link to the Video](#) property is used to define the expander column in the Gantt component to expand and collapse the child rows.

To learn about Gantt Chart Customize Column Data, you can check on this video:

### Defining columns

Using the [columns](#) property, you can define the columns in Gantt. If the columns are not defined, then the default columns will be rendered based on the mapped data source fields in the [taskFields](#) property. Refer to the following code example for defining the columns in Gantt along with their widths.

### APP.VUE

```
<template>
  <div>
    <ejs-gantt ref='gantt' id="GanttContainer" :dataSource="data"
    :taskFields = "taskFields" :height = "height" :columns = "columns"
    :splitterSettings = "splitterSettings"></ejs-gantt>
  </div>
</template>
<script>
import Vue from "vue";
import { GanttPlugin } from "@syncfusion/ej2-vue-gantt";
import { editingData } from './data-source.js';
Vue.use(GanttPlugin);
export default {
  data: function() {
    return{
      data: [
        {
          TaskID: 1,
          TaskName: 'Project Initiation',
          StartDate: new Date('04/02/2019'),
          EndDate: new Date('04/21/2019'),
          subtasks: [
            { TaskID: 2, TaskName: 'Identify Site location', StartDate:
            new Date('04/02/2019'), Duration: 4, Progress: 50 },
            { TaskID: 3, TaskName: 'Perform Soil test', StartDate: new
            Date('04/02/2019'), Duration: 4, Progress: 50 },
            { TaskID: 4, TaskName: 'Soil test approval', StartDate: new
            Date('04/02/2019'), Duration: 4, Progress: 50 },
          ]
        },
        {
          TaskID: 5,
```

```

TaskName: 'Project Estimation',
StartDate: new Date('04/02/2019'),
EndDate: new Date('04/21/2019'),
subtasks: [
    { TaskID: 6, TaskName: 'Develop floor plan for estimation',
StartDate: new Date('04/04/2019'), Duration: 3, Progress: 50 },
    { TaskID: 7, TaskName: 'List materials', StartDate: new
Date('04/04/2019'), Duration: 3, Progress: 50 },
    { TaskID: 8, TaskName: 'Estimation approval', StartDate: new
Date('04/04/2019'), Duration: 3, Progress: 50 }
],
},
],
taskFields: {
    id: 'TaskID',
    name: 'TaskName',
    startDate: 'StartDate',
    duration: 'Duration',
    progress: 'Progress',
    child: 'subtasks',
},
splitterSettings:{
    columnIndex:2
},
height:'450px',
columns: [
    { field: 'TaskID', width: '150' },
    { field: 'TaskName', width: '250' }
]
},
};
</script>

```

```
{% previewsample "page.domainurl/code-snippet/gantt/columns-cs14" %}
```

### Custom column header

The column header text can be defined using the `headerText` property, and you can customize the column headers using the `headerTemplate` property.

## APP.VUE

```
<template>
  <div>
    <ejs-gantt ref='gantt' id="GanttContainer" :dataSource="data"
:taskFields="taskFields" :height="height"
    :splitterSettings="splitterSettings">
      <e-columns>
        <e-column field='TaskName' :headerTemplate='projectName'
width=150></e-column>
        <e-column field='StartDate' :headerTemplate='dateTemplate'
width=150></e-column>
        <e-column field='Duration' :headerTemplate='durationTemplate'
width=150></e-column>
        <e-column field='Progress' :headerTemplate='progressTemplate'
width=150></e-column>
      </e-columns>
    </ejs-gantt>
  </div>
</template>
```

```

    </e-columns>
  </ejs-gantt>
</div>
</template>
<script>
import Vue from "vue";
import { GanttPlugin } from "@syncfusion/ej2-vue-gantt";
import { editingData } from './data-source.js';
Vue.use(GanttPlugin);
export default {
  data: function() {
    return{
      data: editingData,
      projectName: function () {
        return { template : Vue.component('columnTemplate',{
          template: `<div class="image">
            
Task Name
          </div>`,
          data: function() {
            return {
              data: {}
            }
          },
          computed: {
            image: function() {
              return "Taskname.png";
            }
          }
        })}
      },
      dateTemplate: function () {
        return { template : Vue.component('columnTemplate',{
          template: `<div class="image">
            
Start Date
          </div>`,
          data: function() {
            return {
              data: {}
            }
          },
          computed: {
            image: function() {
              return "Startdate.png";
            }
          }
        })}
      },
      durationTemplate: function () {
        return { template : Vue.component('columnTemplate',{
          template: `<div class="image">
            
Duration
          </div>`,
          data: function() {
            return {

```

```

        data: {}
    },
    computed: {
        image: function() {
            return "Duration.png";
        }
    }
    }
    }
    },
    progressTemplate: function () {
        return { template : Vue.component('columnTemplate',{
            template: `<div class="image">
                
Progress
                </div>`,
            data: function() {
                return {
                    data: {}
                }
            },
            computed: {
                image: function() {
                    return "progress.png" ;
                }
            }
        })
    },
    taskFields: {
        id: 'TaskID',
        name: 'TaskName',
        startDate: 'StartDate',
        duration: 'Duration',
        progress: 'Progress',
        child: 'subtasks',
    },
    height: '450px',
    splitterSettings: {
        columnIndex: 4
    },
    columns: [
        { field: 'TaskName', headerTemplate: '#projectName', width:
'150' },
        { field: 'StartDate', headerTemplate: '#dateTemplate', width:
'150' },
        { field: 'Duration', headerTemplate: '#durationTemplate',
headerText: 'Duration', width: '150' },
        { field: 'Progress', headerTemplate: '#progressTemplate',
headerText: 'Progress', width: '150' },
    ],
    };
    },
};
</script>

```

{% previewsample "page.domainurl/code-snippet/gantt/columns-cs15" %}

### Format

To format the cell values based on a specific culture, use the [columns.format](#) property. The Gantt component uses the [Internationalization](#) library to format **number** and **date** values.

### APP.VUE

```
<template>
  <div>
    <ejs-gantt ref='gantt' id="GanttContainer" :dataSource="data"
    :taskFields = "taskFields" :height = "height" :columns = "columns"
    :rowHeight = "rowHeight" :splitterSettings = "splitterSettings"></ejs-gantt>
  </div>
</template>
<script>
import Vue from "vue";
import { GanttPlugin } from "@syncfusion/ej2-vue-gantt";
import { editingData } from './data-source.js';
Vue.use(GanttPlugin);
export default {
  data: function() {
    return{
      data: editingData,
      taskFields: {
        id: 'TaskID',
        name: 'TaskName',
        startDate: 'StartDate',
        duration: 'Duration',
        progress: 'Progress',
        child: 'subtasks',
      },
      rowHeight:50,
      splitterSettings:{
        columnIndex:3
      },
      height:'450px',
      columns: [
        { field: 'TaskID', headerText: 'Task ID', textAlign: 'Left',
width: '100' },
        { field: 'Progress', headerText: 'Progress', width:
'150',format: 'C' },
        { field: 'TaskName', headerText: 'Task Name', width: '150' },
        { field: 'StartDate', headerText: 'Start Date', width: '150' },
        { field: 'Duration', headerText: 'Duration', width: '150' }
      ]
    };
  },
};
</script>
```

{% previewsample "page.domainurl/code-snippet/gantt/columns-cs16" %}

By default, the number and date values are formatted in **en-US** culture.

### Number formatting

The number or integer values can be formatted using the following format strings.



Format | Description | Remarks

{ type:'date', format:'dd/MM/yyyy' } | 04/07/1996

{ type:'date', format:'dd.MM.yyyy' } | 04.07.1996

{ type:'date', skeleton:'short' } | 7/4/96

{ type: 'dateTime', format: 'dd/MM/yyyy hh:mm a' } | 04/07/1996 12:00 AM

{ type: 'dateTime', format: 'MM/dd/yyyy hh:mm:ss a' } | 07/04/1996 12:00:00 AM

### APP.VUE

```
<template>
  <div>
    <ejs-gantt ref='gantt' id="GanttContainer" :dataSource="data"
:taskFields = "taskFields" :height = "height" :columns = "columns"
:rowHeight = "rowHeight" :splitterSettings = "splitterSettings"></ejs-gantt>
  </div>
</template>
<script>
import Vue from "vue";
import { GanttPlugin } from "@syncfusion/ej2-vue-gantt";
import { editingData } from './data-source.js';
Vue.use(GanttPlugin);
export default {
  data: function() {
    return{
      data: editingData,
      taskFields: {
        id: 'TaskID',
        name: 'TaskName',
        startDate: 'StartDate',
        duration: 'Duration',
        progress: 'Progress',
        child: 'subtasks',
      },
      rowHeight:50,
      splitterSettings:{
        columnIndex:3
      },
      height:'450px',
      columns: [
        { field: 'TaskID', headerText: 'Task ID', textAlign: 'Left',
width: '100' },
        { field: 'TaskName', headerText: 'Task Name', width: '150' },
        { field: 'StartDate', headerText: 'Start Date', width: '150',
format: { format: 'dd/MM/yyyy', type: 'date' } },
        { field: 'Duration', headerText: 'Duration', width: '150' },
        { field: 'Progress', headerText: 'Progress', width:
'150',format: 'C' },
      ]
    };
  },
};
</script>
```

```
{% previewsample "page.domainurl/code-snippet/gantt/columns-cs17" %}
```

### [Change tree/expander column](#)

The tree/expander column is a column in the Gantt component, that has icons to expand or collapse the parent records. You can define the tree column index in the Gantt component by using the [treeColumnIndex](#) property and the default value of this property is 0. The following code example shows how to use this property.

#### APP.VUE

```
<template>
  <div>
    <ejs-gantt ref='gantt' id="GanttContainer" :dataSource="data"
      :taskFields = "taskFields" :height = "height" :treeColumnIndex='2'
      :splitterSettings = "splitterSettings" :allowResizing = 'true'></ejs-gantt>
  </div>
</template>
<script>
import Vue from "vue";
import { GanttPlugin } from "@syncfusion/ej2-vue-gantt";
import { editingData } from './data-source.js';
Vue.use(GanttPlugin);
export default {
  data: function() {
    return{
      data: editingData,
      taskFields: {
        id: 'TaskID',
        name: 'TaskName',
        startDate: 'StartDate',
        duration: 'Duration',
        progress: 'Progress',
        child: 'subtasks',
      },
      splitterSettings:{
        columnIndex:3
      },
      height:'450px',
    };
  },
};
</script>
```

```
{% previewsample "page.domainurl/code-snippet/gantt/columns-cs19" %}
```

### [Show or Hide columns dynamically](#)

You can show or hide gantt columns dynamically using external buttons by invoking the [showColumn](#) or [hideColumn](#) method.

#### APP.VUE

```
<template>
  <div>
    <ejs-button id="show" cssClass="e-info" v-
      on:click.native="show">Show</ejs-button>
```

```

        <ejs-button id="hide" cssClass="e-info" v-
on:click.native="hide">Hide</ejs-button>
        <br>
        <ejs-gantt ref='gantt' id="GanttContainer" :dataSource="data"
:taskFields = "taskFields" :height = "height" :columns = "columns"
:splitterSettings = "splitterSettings"></ejs-gantt>
    </div>
</template>
<script>
import Vue from "vue";
import { GanttPlugin } from "@syncfusion/ej2-vue-gantt";
import { editingData } from './data-source.js';
import { ButtonPlugin } from "@syncfusion/ej2-vue-buttons";
Vue.use(ButtonPlugin);
Vue.use(GanttPlugin);
export default {
    data: function() {
        return{
            data: editingData,
            taskFields: {
                id: 'TaskID',
                name: 'TaskName',
                startDate: 'StartDate',
                duration: 'Duration',
                progress: 'Progress',
                child: 'subtasks',
            },
            height: '450px',
            splitterSettings:{
                position: '80%'
            },
            columns: [
                { field: 'TaskID', headerText: 'Task ID', textAlign: 'Left',
width: '100' },
                { field: 'TaskName', headerText: 'Task Name', width: '150' },
                { field: 'StartDate', headerText: 'Start Date', width: '150' },
                { field: 'Duration', headerText: 'Duration', width: '150' },
                { field: 'Progress', headerText: 'Progress', width: '150' },
            ]
        };
    }
    methods: {
        show: function(e){
            var ganttChart =
document.getElementById('GanttContainer').ej2_instances[0];
            ganttChart.showColumn(["Duration"]);
        },
        hide: function(e){
            var ganttChart =
document.getElementById('GanttContainer').ej2_instances[0];
            ganttChart.hideColumn(["Duration"]);
        },
    },
};
</script>

```

```
{% previewsample "page.domainurl/code-snippet/gantt/columns-cs20" %}
```

### Controlling Gantt column actions

You can enable or disable gantt action for a particular column by setting the [allowFiltering](#), [allowSorting](#), [allowReordering](#), and [allowEditing](#) properties.

### APP.VUE

```
<template>
  <div>
    <ejs-gantt ref='gantt' id="GanttContainer" :dataSource="data"
      :taskFields = "taskFields" :height = "height" :columns = "columns"
      :allowFiltering = 'true' :splitterSettings = "splitterSettings"
      :allowSorting = 'true' :allowReordering = 'true' :editSettings =
        "editSettings"></ejs-gantt>
    </div>
  </template>
<script>
import Vue from "vue";
import { GanttPlugin, Sort, Filter, Reorder, Edit } from "@syncfusion/ej2-
vue-gantt";
import { editingData } from './data-source.js';
Vue.use(GanttPlugin);
export default {
  data: function() {
    return{
      data: editingData,
      taskFields: {
        id: 'TaskID',
        name: 'TaskName',
        startDate: 'StartDate',
        duration: 'Duration',
        progress: 'Progress',
        child: 'subtasks',
      },
      height: '450px',
      splitterSettings: {
        position: '80%'
      },
      editSettings: {
        allowEditing: true
      },
      columns: [
        { field: 'TaskID', headerText: 'Task ID', textAlign: 'Left',
width: '100' },
        { field: 'TaskName', headerText: 'Task Name', width: '150',
allowSorting: false },
        { field: 'StartDate', headerText: 'Start Date', width: '150',
allowEditing: false },
        { field: 'Duration', headerText: 'Duration', width: '150',
allowFiltering: false },
        { field: 'Progress', headerText: 'Progress', width: '150',
allowReordering: false },
      ],
    };
  },
  provide: {
```

```

      gantt: [ Sort, Filter, Reorder, Edit ]
    }
  };
</script>

```

{% previewsample "page.domainurl/code-snippet/gantt/columns-cs2" %}

### Column type

Column type can be specified using the `columns.type` property. It specifies the type of data the column binds.

If the `format` is defined for a column, the column uses `type` to select the appropriate format option [number](#) or [date](#).

Gantt column supports the following types:

- string
- number
- boolean
- date
- datetime

If the `type` is not defined, it will be determined from the first record of the [dataSource](#).

In case if the first record of the [dataSource](#) is null/blank value for a column then it is necessary to define the `type` for that column.

### Column reordering in Vue Gantt component

The column reordering can be done by dragging a column header from one index to another index within the TreeGrid. To enable reordering, set the [allowReordering](#) property to `true`.

To reorder the columns, inject the `Reorder` module in the `provide` section.

### APP.VUE

```

<template>
  <div>
    <ejs-gantt ref='gantt' id="GanttContainer" :dataSource="data"
    :taskFields = "taskFields" :height = "height" :columns = "columns"
    :allowReordering = 'true' :splitterSettings = "splitterSettings"></ejs-
    gantt>
  </div>
</template>
<script>
import Vue from "vue";
import { GanttPlugin, Reorder } from "@syncfusion/ej2-vue-gantt";
import { editingData } from './data-source.js';
Vue.use(GanttPlugin);
export default {
  data: function() {
    return {
      data: editingData,
      taskFields: {
        id: 'TaskID',
        name: 'TaskName',

```

```

        startDate: 'StartDate',
        duration: 'Duration',
        progress: 'Progress',
        child: 'subtasks',
    },
    height: '450px',
    splitterSettings: {
        columnIndex: 3
    },
    columns: [
        { field: 'TaskID', headerText: 'Task ID', textAlign: 'Left',
width: '100' },
        { field: 'TaskName', headerText: 'Task Name', width: '150' },
        { field: 'StartDate', headerText: 'Start Date', width: '150' },
        { field: 'Duration', headerText: 'Duration', width: '150' },
        { field: 'Progress', headerText: 'Progress', width: '150' },
    ]
    };
    },
    provide: {
        gantt: [Reorder]
    }
};
</script>

```

{% previewsample "page.domainurl/code-snippet/gantt/columns-cs7" %}

You can disable the reordering of a particular column by setting the [columns.allowReordering](#) property to **false**.

#### Reorder Events

During the reorder action, the gantt component triggers the below three events.

1. The [columnDragStart](#) event triggers when column header element drag (move) starts.
2. The [columnDrag](#) event triggers when column header element is dragged (moved) continuously.
3. The [columnDrop](#) event triggers when a column header element is dropped on the target column.

#### APP.VUE

```

<template>
  <div>
    <ejs-gantt ref='gantt' id="GanttContainer" :dataSource="data"
:taskFields = "taskFields" :height = "height" :columns = "columns"
:allowReordering = 'true' :splitterSettings = "splitterSettings"
:columnDrag='columnDrag' :columnDrop='columnDrop'
:columnDragStart='columnDragStart'></ejs-gantt>
  </div>
</template>
<script>
import Vue from "vue";
import { GanttPlugin, Reorder } from "@syncfusion/ej2-vue-gantt";
import { editingData } from './data-source.js';
Vue.use(GanttPlugin);
export default {

```

```

data: function() {
  return{
    data: editingData,
    taskFields: {
      id: 'TaskID',
      name: 'TaskName',
      startDate: 'StartDate',
      duration: 'Duration',
      progress: 'Progress',
      child: 'subtasks',
    },
    height: '450px',
    splitterSettings: {
      columnIndex: 3
    },
    columns: [
      { field: 'TaskID', headerText: 'Task ID', textAlign: 'Left',
width: '100' },
      { field: 'TaskName', headerText: 'Task Name', width: '150' },
      { field: 'StartDate', headerText: 'Start Date', width: '150' },
      { field: 'Duration', headerText: 'Duration', width: '150' },
      { field: 'Progress', headerText: 'Progress', width: '150' },
    ]
  };
},
provide: {
  gantt: [Reorder]
},
methods: {
  columnDragStart: function() {
    alert('columnDragStart event is Triggered');
  },
  columnDrag: function() {
    alert('columnDrag event is Triggered');
  },
  columnDrop: function() {
    alert('columnDrop event is Triggered');
  }
}
};
</script>

```

{% previewsample "page.domainurl/code-snippet/gantt/columns-cs8" %}

### Reorder multiple columns

Multiple columns can be reordered at a time by using the [reorderColumns](#) method.

### APP.VUE

```

<template>
  <div>
    <ejs-button id="reorder" cssClass="e-info" v-
on:click.native="change">Reorder</ejs-button>
    <br><br><br>
    <ejs-gantt ref='gantt' id="GanttContainer" :dataSource="data"
:taskFields = "taskFields" :height="height" :columns = "columns"

```

```

:allowReordering = 'true' :splitterSettings = "splitterSettings"></ejs-
gantt>
</div>
</template>
<script>
import Vue from "vue";
import { GanttPlugin, Reorder } from "@syncfusion/ej2-vue-gantt";
import { ButtonPlugin } from "@syncfusion/ej2-vue-buttons";
import { editingData } from './data-source.js';
Vue.use(GanttPlugin);
Vue.use(ButtonPlugin);
export default {
  data: function() {
    return{
      data: editingData,
      taskFields: {
        id: 'TaskID',
        name: 'TaskName',
        startDate: 'StartDate',
        duration: 'Duration',
        progress: 'Progress',
        child: 'subtasks',
      },
      height: '450px',
      splitterSettings: {
        columnIndex: 3
      },
      columns: [
        { field: 'TaskID', headerText: 'Task ID', textAlign: 'Left',
width: '100' },
        { field: 'TaskName', headerText: 'Task Name', width: '150' },
        { field: 'StartDate', headerText: 'Start Date', width: '150' },
        { field: 'Duration', headerText: 'Duration', width: '150' },
        { field: 'Progress', headerText: 'Progress', width: '150' },
      ]
    };
  },
  provide: {
    gantt: [Reorder]
  },
  methods: {
    change: function(e) {
      var ganttChart =
document.getElementById('GanttContainer').ej2_instances[0];
      ganttChart.reorderColumns(['TaskID', 'TaskName'], 'Progress');
    }
  },
};
</script>

```

{% previewsample "page.domainurl/code-snippet/gantt/columns-cs9" %}

### Column resizing in Vue Gantt component

The column width can be resized by clicking and dragging the right edge of the column header. While dragging, the width of the column will be resized immediately. Each column can be auto resized by



double-clicking the right edge of the column header to fit the width of that column based on the widest cell content. To resize the column, set the [columns.allowResizing](#) property to `true`. The following code example shows how to enable the column resize feature in the Gantt component.

To resize the column, inject the `Resize` module in the `provide` section.

#### APP.VUE

```
<template>
  <div>
    <ejs-gantt ref='gantt' id="GanttContainer" :dataSource="data"
    :taskFields = "taskFields" :height = "height" :columns = "columns"
    :splitterSettings = "splitterSettings" :allowResizing = 'true'></ejs-gantt>
  </div>
</template>
<script>
import Vue from "vue";
import { GanttPlugin, Resize } from "@syncfusion/ej2-vue-gantt";
import { editingData } from './data-source.js';
Vue.use(GanttPlugin);
export default {
  data: function() {
    return{
      data: editingData,
      taskFields: {
        id: 'TaskID',
        name: 'TaskName',
        startDate: 'StartDate',
        duration: 'Duration',
        progress: 'Progress',
        child: 'subtasks',
      },
      splitterSettings:{
        columnIndex:4
      },
      height:'450px',
      columns: [
        { field: 'TaskID', headerText: 'Task ID', textAlign: 'Left',
width: '100' },
        { field: 'TaskName', headerText: 'Task Name', width: '150' },
        { field: 'StartDate', headerText: 'Start Date', width: '150' },
        { field: 'Duration', headerText: 'Duration', width: '150' },
        { field: 'Progress', headerText: 'Progress', width: '150' },
      ]
    };
  },
  provide: {
    gantt: [Resize]
  }
};
</script>
```

{% previewsample "page.domainurl/code-snippet/gantt/columns-cs10" %}

You can disable resizing for a particular column by setting the [columns.allowResizing](#) to `false`.

*Defining minimum and maximum column width*

The column resizing can be restricted between minimum and maximum widths by defining the [columns->minWidth](#) and [columns->maxWidth](#) properties.

In the following example, the minimum and maximum widths are defined for the **Duration**, and **Task Name** columns.

**APP.VUE**

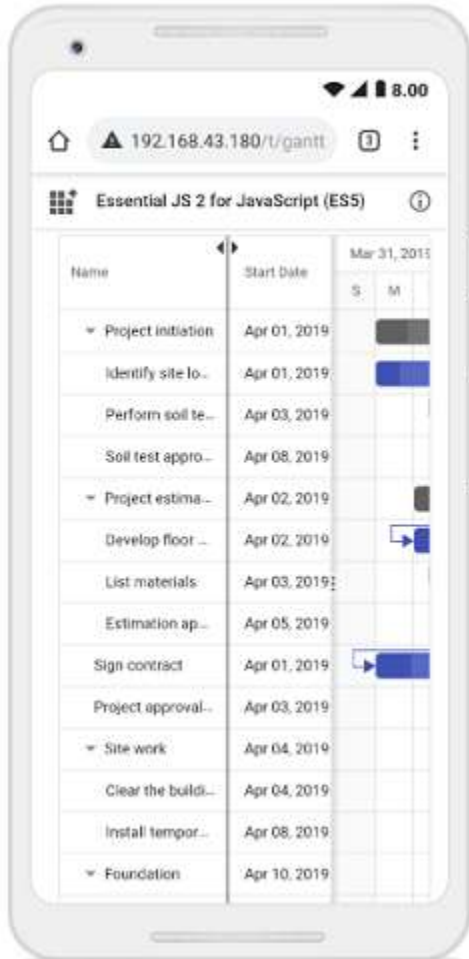
```
<template>
  <div>
    <ejs-gantt ref='gantt' id="GanttContainer" :dataSource="data"
    :taskFields = "taskFields" :height = "height" :columns = "columns"
    :splitterSettings="splitterSettings" :allowResizing = 'true'></ejs-gantt>
  </div>
</template>
<script>
import Vue from "vue";
import { GanttPlugin, Resize } from "@syncfusion/ej2-vue-gantt";
import { editingData } from './data-source.js';
Vue.use(GanttPlugin);
export default {
  data: function() {
    return{
      data: editingData,
      taskFields: {
        id: 'TaskID',
        name: 'TaskName',
        startDate: 'StartDate',
        duration: 'Duration',
        progress: 'Progress',
        child: 'subtasks',
      },
      height: '450px',
      columns: [
        { field: 'TaskID', headerText: 'Task ID', textAlign: 'Left',
width: '100' },
        { field: 'TaskName', headerText: 'Task Name', width:
'200',minWidth: '150' ,maxWidth: '250' },
        { field: 'Duration', headerText: 'Duration', width:
'100',minWidth: '50' ,maxWidth: '200' },
        { field: 'StartDate', headerText: 'Start Date', width: '150' },
        { field: 'Progress', headerText: 'Progress', width: '150' },
      ],
      splitterSettings:{
        columnIndex:4
      },
    };
  },
  provide: {
    gantt: [Resize]
  }
};
</script>
```

{% previewsample "page.domainurl/code-snippet/gantt/columns-cs11" %}

*Touch interaction*

When the right edge of the column header cell is **tapped**, a floating handler will be visible over the right border of the column. To [resize](#) the column, drag the floating handler as needed.

The following screenshot represents the Gantt column resizing in touch device.

*Column template in Vue Gantt component*

A column template is used to customize the column's look. The following code example explains how to define the custom template in Gantt using the [template](#) property.

**APP.VUE**

```
<template>
  <div>
    <ejs-gantt ref='gantt' id="GanttContainer" :dataSource="data"
    :taskFields="taskFields" :height="height"
      :rowHeight="rowHeight" :splitterSettings="splitterSettings"
    :allowResizing='true' :resourceFields="resourceFields"
      :resources="resources">
      <e-columns>
        <e-column field='TaskID' headerText='Task ID' textAlign='Left'
width='100'></e-column>
```

```

    <e-column field='TaskName' headerText='Task Name' width='150'></e-
column>
    <e-column field='resources' headerText='Resources' width='250'
:template="cTemplate"></e-column>
    <e-column field='StartDate' headerText='Start Date' width='150'></e-
column>
    <e-column field='Duration' headerText='Duration' width='150'></e-
column>
    <e-column field='Progress' headerText='Progress' width='150'></e-
column>
  </e-columns>
  <template v-slot:cTemplate="{data}">
    <div class="columnTemplate" v-
if="data.ganttProperties.resourceNames">
      
      <div style="display:inline-
block;width:100%;position:relative;left:30px">
        {{data.ganttProperties.resourceNames}}</div>
      </div>
    </template>
  </ejs-gantt>
</div>
</template>
<script>
import Vue from "vue";
import { GanttPlugin } from "@syncfusion/ej2-vue-gantt";
import { editingData , editingResources } from './data-source.js';
Vue.use(GanttPlugin);
export default {
  data: function() {
    return{
      data: [
        {
          TaskID: 1,
          TaskName: 'Project Initiation',
          StartDate: new Date('04/02/2019'),
          EndDate: new Date('04/21/2019'),
          subtasks: [
            {
              TaskID: 2, TaskName: 'Identify Site location', StartDate:
new Date('04/02/2019'), Duration: 0,
              Progress: 30, resources: [1], info: 'Measure the total
property area allotted for construction'
            },
            {
              TaskID: 3, TaskName: 'Perform Soil test', StartDate: new
Date('04/02/2019'), Duration: 4, Predecessor: '2',
              resources: [2], info: 'Obtain an engineered soil test of lot
where construction is planned.' +
              'From an engineer or company specializing in soil
testing'
            }
          ],
        }
      ],
    }
  },

```

```

        { TaskID: 4, TaskName: 'Soil test approval', StartDate: new
Date('04/02/2019'), Duration: 0, Predecessor: '3', Progress: 30, resources:
[3], },
    ],
    },
    {
        TaskID: 5,
        TaskName: 'Project Estimation',
        StartDate: new Date('04/02/2019'),
        EndDate: new Date('04/21/2019'),
        subtasks: [
            {
                TaskID: 6, TaskName: 'Develop floor plan for estimation',
                StartDate: new Date('04/04/2019'),
                Duration: 3, Predecessor: '4', Progress: 30, resources: [4],
                info: 'Develop floor plans and obtain a materials list for
estimations'
            },
            {
                TaskID: 7, TaskName: 'List materials', StartDate: new
Date('04/04/2019'),
                Duration: 3, Predecessor: '6', resources: [8], info: ''
            },
            {
                TaskID: 8, TaskName: 'Estimation approval', StartDate: new
Date('04/04/2019'),
                Duration: 0, Predecessor: '7', info: ''
            }
        ]
    },
    ],
    taskFields: {
        id: 'TaskID',
        name: 'TaskName',
        resourceInfo: 'resources',
        startDate: 'StartDate',
        duration: 'Duration',
        progress: 'Progress',
        child: 'subtasks',
    },
    rowHeight: 50,
    splitterSettings: {
        columnIndex: 3
    },
    height: '450px',
    resourceFields: {
        id: 'resourceId',
        name: 'resourceName',
    },
    resources: editingResources
}
},
};
</script>

```

{% previewsample "page.domainurl/code-snippet/gantt/columns-cs13" %}

### Column menu in Vue Gantt component

The column menu has options to integrate features like sorting, filtering, and autofit. It will show a menu with the integrated feature when users click the Multiple icon of the column. To enable the column menu, you should set the [showColumnMenu](#) property to `true`.

The default items are displayed in the following table:

Item	Description
SortAscending	Sort the current column in ascending order.
SortDescending	Sort the current column in descending order.
AutoFit	Auto fit the current column.
AutoFitAll	Auto fit all columns.
Filter	Show the filter option as given in <code>filterSettings.type</code>

### APP.VUE

```
<template>
  <div>
    <ejs-gantt ref='gantt' id="GanttContainer" :dataSource="data"
    :taskFields = "taskFields" :height = "height" :columns = "columns"
    :allowFiltering = 'true' :splitterSettings = "splitterSettings"
    :showColumnMenu = 'true' :allowSorting = 'true'></ejs-gantt>
  </div>
</template>
<script>
import Vue from "vue";
import { GanttPlugin, Sort, Filter, ColumnMenu } from "@syncfusion/ej2-vue-gantt";
import { editingData } from './data-source.js';
Vue.use(GanttPlugin);
export default {
  data: function() {
    return{
      data: editingData,
      taskFields: {
        id: 'TaskID',
        name: 'TaskName',
        startDate: 'StartDate',
        duration: 'Duration',
        progress: 'Progress',
        child: 'subtasks',
      },
      height: '450px',
      splitterSettings: {
        position: '100%'
      },
      columns: [
        { field: 'TaskID', headerText: 'Task ID', textAlign: 'Left',
width: '100' },
        { field: 'TaskName', headerText: 'Task Name', width: '150' },
        { field: 'StartDate', headerText: 'Start Date', width: '150' },

```

```

        { field: 'Duration', headerText: 'Duration', width: '150' },
        { field: 'Progress', headerText: 'Progress', width: '150' },
    ],
    };
},
    provide: {
        gantt: [ Sort, Filter, ColumnMenu ]
    }
};
</script>

```

{% previewsample "page.domainurl/code-snippet/gantt/columns-cs3" %}

You can disable the column menu for a particular column by setting the `columns.showColumnMenu` to `false`.

### Column Menu Events

During the resizing action, the gantt component triggers the below two events.

1. The [columnMenuOpen](#) event triggers before the column menu opens.
2. The [columnMenuClick](#) event triggers when the user clicks the column menu of the gantt.

### APP.VUE

```

<template>
  <div>
    <ejs-gantt ref='gantt' id="GanttContainer" :dataSource="data"
    :taskFields = "taskFields" :height = "height" :columns = "columns"
    :allowFiltering = 'true' :splitterSettings = "splitterSettings"
    :showColumnMenu = 'true' :allowSorting = 'true'
    :columnMenuClick='columnMenuClick'
    :columnMenuOpen='columnMenuOpen'></ejs-gantt>
  </div>
</template>
<script>
import Vue from "vue";
import { GanttPlugin, Sort, Filter, ColumnMenu } from "@syncfusion/ej2-vue-gantt";
import { editingData } from './data-source.js';
Vue.use(GanttPlugin);
export default {
  data: function() {
    return{
      data: editingData,
      taskFields: {
        id: 'TaskID',
        name: 'TaskName',
        startDate: 'StartDate',
        duration: 'Duration',
        progress: 'Progress',
        child: 'subtasks',
      },
      height: '450px',
      splitterSettings: {
        position: '100%'
      }
    }
  }
}

```

```

    },
    columns: [
      { field: 'TaskID', headerText: 'Task ID', textAlign: 'Left',
width: '100' },
      { field: 'TaskName', headerText: 'Task Name', width: '150' },
      { field: 'StartDate', headerText: 'Start Date', width: '150' },
      { field: 'Duration', headerText: 'Duration', width: '150' },
      { field: 'Progress', headerText: 'Progress', width: '150' },
    ],
  },
  provide: {
    gantt: [ Sort, Filter, ColumnMenu ]
  },
  methods: {
    columnMenuOpen: function(){
      alert('columnMenuOpen event is Triggered');
    },
    columnMenuClick: function(){
      alert('columnMenuClick event is Triggered');
    }
  }
};
</script>

```

{% previewsample "page.domainurl/code-snippet/gantt/columns-cs4" %}

#### Custom Column Menu Item

Custom column menu items can be added by defining the [columnMenuItems](#).

Actions for this customized items can be defined in the [columnMenuClick](#) event.

#### APP.VUE

```

<template>
  <div>
    <ejs-gantt ref='gantt' id="GanttContainer" :dataSource="data"
:taskFields = "taskFields" :height = "height" :columns = "columns"
:splitterSettings = "splitterSettings" :showColumnMenu = 'true'
:allowSorting = 'true' :columnMenuClick='columnMenuClick'
:columnMenuItems='columnMenuItems'
:sortSettings='sortSettings'></ejs-gantt>
  </div>
</template>
<script>
import Vue from "vue";
import { GanttPlugin, Sort, Filter, ColumnMenu } from "@syncfusion/ej2-vue-gantt";
import { editingData } from './data-source.js';
Vue.use(GanttPlugin);
export default {
  data: function() {
    return{
      data: editingData,
      columnMenuItems: [{text:'Clear Sorting', id:'ganttclearsorting'}],
      taskFields: {
        id: 'TaskID',

```



```

        name: 'TaskName',
        startDate: 'StartDate',
        duration: 'Duration',
        progress: 'Progress',
        child: 'subtasks',
    },
    height: '450px',
    splitterSettings: {
        position: '100%'
    },
    columns: [
        { field: 'TaskID', headerText: 'Task ID', textAlign: 'Left',
width: '100' },
        { field: 'TaskName', headerText: 'Task Name', width: '150' },
        { field: 'StartDate', headerText: 'Start Date', width: '150' },
        { field: 'Duration', headerText: 'Duration', width: '150' },
        { field: 'Progress', headerText: 'Progress', width: '150' },
    ],
    sortSettings: { columns: [{ direction: "Ascending", field:
"TaskName" }] }
    };
},
    provide: {
        gantt: [ Sort, Filter, ColumnMenu ]
    },
    methods: {
        columnMenuClick: function(args) {
            if(args.item.id === 'ganttclearsorting'){
                var ganttChart =
document.getElementById('GanttContainer').ej2_instances[0];
                ganttChart.clearSorting();
            }
        }
    }
};
</script>

```

{% previewsample "page.domainurl/code-snippet/gantt/columns-cs5" %}

#### *Customize menu items for particular columns*

Sometimes, you have a scenario that to hide an item from column menu for particular columns. In that case, you need to define the [columnMenuOpenEventArgs.hide](#) as true in the [columnMenuOpen](#) event.

The following sample, **Filter** item was hidden in column menu when opens for the **Task Name** column.

#### **APP.VUE**

```

<template>
  <div>
    <ejs-gantt ref='gantt' id="GanttContainer" :dataSource="data"
:taskFields = "taskFields" :height = "height" :columns = "columns"
:allowFiltering = 'true' :splitterSettings = "splitterSettings"
:showColumnMenu = 'true' :allowSorting = 'true'
:columnMenuOpen='columnMenuOpen'></ejs-gantt>
  </div>
</template>
<script>

```

```

import Vue from "vue";
import { GanttPlugin, Sort, Filter, ColumnMenu } from "@syncfusion/ej2-vue-gantt";
import { editingData } from './data-source.js';
Vue.use(GanttPlugin);
export default {
  data: function() {
    return {
      data: editingData,
      taskFields: {
        id: 'TaskID',
        name: 'TaskName',
        startDate: 'StartDate',
        duration: 'Duration',
        progress: 'Progress',
        child: 'subtasks',
      },
      height: '450px',
      splitterSettings: {
        position: '100%'
      },
      columns: [
        { field: 'TaskID', headerText: 'Task ID', textAlign: 'Left',
width: '100' },
        { field: 'TaskName', headerText: 'Task Name', width: '150' },
        { field: 'StartDate', headerText: 'Start Date', width: '150' },
        { field: 'Duration', headerText: 'Duration', width: '150' },
        { field: 'Progress', headerText: 'Progress', width: '150' },
      ],
    };
  },
  provide: {
    gantt: [ Sort, Filter, ColumnMenu ]
  },
  methods: {
    columnMenuOpen: function (args) {
      for (let item of args.items) {
        if (item.text === 'Filter' && args.column.field === 'TaskName')
        {
          item.hide = true;
        } else {
          item.hide = false;
        }
      }
    }
  },
};
</script>

```

{% previewsample "page.domainurl/code-snippet/gantt/columns-cs6" %}

### Responsive columns in Vue Gantt component

You can toggle the column visibility based on media queries, which are defined in the [hideAtMedia](#). The [hideAtMedia](#) accepts valid [Media Queries](#).

### APP.VUE

```

<template>
  <div>
    <ejs-gantt ref='gantt' id="GanttContainer" :dataSource="data"
:taskFields = "taskFields" :height = "height" :columns = "columns"
:splitterSettings = "splitterSettings" :allowResizing = 'true'></ejs-gantt>
  </div>
</template>
<script>
import Vue from "vue";
import { GanttPlugin, Resize } from "@syncfusion/ej2-vue-gantt";
import { editingData } from './data-source.js';
Vue.use(GanttPlugin);
export default {
  data: function() {
    return {
      data: editingData,
      taskFields: {
        id: 'TaskID',
        name: 'TaskName',
        startDate: 'StartDate',
        duration: 'Duration',
        progress: 'Progress',
        child: 'subtasks',
      },
      height: '450px',
      splitterSettings: {
        columnIndex: 4
      },
      columns: [
        { field: 'TaskID', headerText: 'Task ID', textAlign: 'Left',
width: '100' },
        { field: 'TaskName', headerText: 'Task Name', width:
'200',hideAtMedia: '(min-width: 700px)' },
        { field: 'StartDate', headerText: 'Start Date', width: '150' },
        { field: 'Duration', headerText: 'Duration', width: '100',
hideAtMedia: '(max-width: 500px)' },
        { field: 'Progress', headerText: 'Progress', width: '150' },
      ]
    };
  },
  provide: {
    gantt: [ Resize ]
  }
};
</script>

```

{% previewsample "page.domainurl/code-snippet/gantt/columns-cs18" %}

Check box columns in Vue Gantt component

To render boolean values as checkbox in columns, you need to set [displayAsCheckBox](#) property as **true**.

#### APP.VUE

```

<template>
  <div>

```

```

    <ejs-gantt ref='gantt' id="GanttContainer" :dataSource="data"
:taskFields = "taskFields" :height = "height" :columns = "columns"
:splitterSettings = "splitterSettings"></ejs-gantt>
  </div>
</template>
<script>
import Vue from "vue";
import { GanttPlugin } from "@syncfusion/ej2-vue-gantt";
import { checkBoxData } from './data-source.js';
Vue.use(GanttPlugin);
export default {
  data: function() {
    return{
      data: checkBoxData,
      taskFields: {
        id: 'TaskID',
        name: 'TaskName',
        startDate: 'StartDate',
        duration: 'Duration',
        progress: 'Progress',
        child: 'subtasks',
      },
      height: '450px',
      splitterSettings:{
        position: '80%'
      },
      columns: [
        { field: 'TaskID', headerText: 'Task ID', textAlign: 'Left',
width: '100' },
        { field: 'TaskName', headerText: 'Task Name', width: '150' },
        { field: 'StartDate', headerText: 'Start Date', width: '150' },
        { field: 'Duration', headerText: 'Duration', width: '150' },
        { field: 'verified', headerText: 'Verified', displayAsCheckBox:
true, type: 'boolean' },
        { field: 'Progress', headerText: 'Progress', width: '150' },
      ]
    };
  }
};
</script>

```

{% previewsample "page.domainurl/code-snippet/gantt/columns-cs1" %}

### Column spanning in Vue Gantt component

The gantt has option to span the adjacent cells. You need to define the [colSpan](#) attribute to span cells in the [QueryCellInfo](#) event.

In the following demo, **Work 1** cells have been spanned.

### APP.VUE

```

<template>
  <div>
    <ejs-gantt ref='gantt' id="GanttContainer" :dataSource="data"
:taskFields = "taskFields" :height = "height" :columns="columns"

```

```

:splitterSettings = "splitterSettings" :queryCellInfo='queryCellInfoEvent'
gridLines='Both'></ejs-gantt>
</div>
</template>
<script>
import Vue from "vue";
import { GanttPlugin } from "@syncfusion/ej2-vue-gantt";
import { colSpanData } from './data-source.js';
Vue.use(GanttPlugin);
export default {
  data: function() {
    return{
      data: colSpanData,
      taskFields: {
        id: 'TaskID',
        name: 'TaskName',
        startDate: 'StartDate',
        duration: 'Duration',
        progress: 'Progress',
        child: 'subtasks',
      },
      height: '450px',
      splitterSettings: {
        position: '80%'
      },
      columns: [
        { field: 'TaskID', headerText: 'Task ID' },
        { field: 'TaskName', headerText: 'Task Name' },
        { field: 'work1', headerText: 'Work 1' },
        { field: 'work2', headerText: 'Work 2' },
        { field: 'StartDate', headerText: 'Start Date' },
        { field: 'Duration', headerText: 'Duration' },
        { field: 'Progress', headerText: 'Progress' }
      ],
    };
  },
  methods: {
    queryCellInfoEvent: function (args) {
      switch (args.data.TaskID) {
        case 1:
          if ((args.column.field == 'work1') && (args.data.taskData.work1
== 'support')) {
            args.colSpan = 2;
          }
          break;
        case 2:
          if ((args.column.field == 'work1') && (args.data.taskData.work1
== 'support')) {
            args.colSpan = 2;
          }
          break;
        case 3:
          if ((args.column.field == 'work1') && (args.data.taskData.work1
== 'support')) {
            args.colSpan = 2;
          }
          break;
      }
    }
  }
}

```

```

        case 4:
            if ((args.column.field == 'work1') && (args.data.taskData.work1
== 'support')) {
                args.colSpan = 2;
            }
            break;
        case 5 :
            if ((args.column.field == 'work1') && (args.data.taskData.work1
== 'support')) {
                args.colSpan = 2;
            }
            break;
        case 7:
            if ((args.column.field == 'work1') && (args.data.taskData.work1
== 'support')) {
                args.colSpan = 2;
            }
            break;
    }
}
};
</script>

```

{% previewsample "page.domainurl/code-snippet/gantt/columns-cs12" %}

## Timeline

### Time line in Vue Gantt component

In the Gantt component, timeline is used to represent the project duration as individual cells with defined unit and formats.

#### Timeline view modes

Gantt contains the following in-built timeline view modes:

- Hour – Minute
- Day – Hour
- Week – Day
- Month – Week
- Year – Month

Timescale mode in the Gantt component can be defined using the [timelineViewMode](#) property, and you can define a timescale mode for the top tier and bottom tier using the [topTier.unit](#) and [bottomTier.unit](#) properties.

#### Week timeline mode

In the **Week** timeline mode, the upper part of the schedule header displays the weeks, whereas the bottom half of the header displays the days. Refer to the following code example.

#### APP.VUE

```

<template>
  <div>

```

```

    <ejs-gantt id="GanttContainer" :dataSource="data" :taskFields =
    "taskFields" :height = "height" :timelineSettings="timelineSettings"></ejs-
    gantt>
  </div>
</template>
<script>
import Vue from "vue";
import { GanttPlugin } from "@syncfusion/ej2-vue-gantt";
Vue.use(GanttPlugin);
export default {
  data: function() {
    return{
      data: [
        {
          TaskID: 1,
          TaskName: 'Project Initiation',
          StartDate: new Date('04/02/2019'),
          EndDate: new Date('04/21/2019'),
          subtasks: [
            { TaskID: 2, TaskName: 'Identify Site location',
StartDate: new Date('04/02/2019'), Duration: 4, Progress: 50 },
            { TaskID: 3, TaskName: 'Perform Soil test', StartDate:
new Date('04/02/2019'), Duration: 4, Progress: 50 },
            { TaskID: 4, TaskName: 'Soil test approval', StartDate:
new Date('04/02/2019'), Duration: 4, Progress: 50 },
          ]
        },
        {
          TaskID: 5,
          TaskName: 'Project Estimation',
          StartDate: new Date('04/02/2019'),
          EndDate: new Date('04/21/2019'),
          subtasks: [
            { TaskID: 6, TaskName: 'Develop floor plan for
estimation', StartDate: new Date('04/04/2019'), Duration: 4, Progress: 50 },
            { TaskID: 7, TaskName: 'List materials', StartDate: new
Date('04/04/2019'), Duration: 4, Progress: 50 },
            { TaskID: 8, TaskName: 'Estimation approval', StartDate:
new Date('04/04/2019'), Duration: 4, Progress: 50 }
          ]
        },
      ],
      height: '450px',
      taskFields: {
        id: 'TaskID',
        name: 'TaskName',
        startDate: 'StartDate',
        endDate: 'EndDate',
        duration: 'Duration',
        progress: 'Progress',
        dependency: 'Predecessor',
        child: 'subtasks'
      },
      timelineSettings: {
        timelineViewMode: 'Week',
      },
    };
  }
};

```

```

    },
  };
</script>

```

{% previewsample "page.domainurl/code-snippet/gantt/timeline-cs1" %}

### Month timeline mode

In the **Month** timeline mode, the upper part of the schedule header displays the months, whereas the bottom header of the schedule displays its corresponding weeks. Refer to the following code example.

### APP.VUE

```

<template>
  <div>
    <ejs-gantt id="GanttContainer" :dataSource="data" :taskFields =
      "taskFields" :height = "height" :timelineSettings="timelineSettings"></ejs-
      gantt>
    </div>
  </template>
<script>
import Vue from "vue";
import { GanttPlugin } from "@syncfusion/ej2-vue-gantt";
Vue.use(GanttPlugin);
export default {
  data: function() {
    return {
      data: [
        {
          TaskID: 1,
          TaskName: 'Project Initiation',
          StartDate: new Date('04/02/2019'),
          EndDate: new Date('04/21/2019'),
          subtasks: [
            { TaskID: 2, TaskName: 'Identify Site location',
              StartDate: new Date('04/02/2019'), Duration: 14, Progress: 50 },
            { TaskID: 3, TaskName: 'Perform Soil test', StartDate:
              new Date('04/02/2019'), Duration: 14, Progress: 50 },
            { TaskID: 4, TaskName: 'Soil test approval', StartDate:
              new Date('04/02/2019'), Duration: 14, Progress: 50 },
          ]
        },
        {
          TaskID: 5,
          TaskName: 'Project Estimation',
          StartDate: new Date('04/02/2019'),
          EndDate: new Date('04/21/2019'),
          subtasks: [
            { TaskID: 6, TaskName: 'Develop floor plan for
              estimation', StartDate: new Date('04/04/2019'), Duration: 14, Progress: 50
            },
            { TaskID: 7, TaskName: 'List materials', StartDate: new
              Date('04/04/2019'), Duration: 14, Progress: 50 },
            { TaskID: 8, TaskName: 'Estimation approval', StartDate:
              new Date('04/04/2019'), Duration: 14, Progress: 50 }
          ]
        }
      ]
    }
  },
};

```



```

    ],
    height: '450px',
    taskFields: {
        id: 'TaskID',
        name: 'TaskName',
        startDate: 'StartDate',
        endDate: 'EndDate',
        duration: 'Duration',
        progress: 'Progress',
        dependency: 'Predecessor',
        child: 'subtasks'
    },
    timelineSettings: {
        timelineViewMode: 'Month',
        timelineUnitSize: 150,
    },
    };
},
};
</script>

```

{% previewsample "page.domainurl/code-snippet/gantt/timeline-cs2" %}

#### Year timeline mode

In the **Year** timeline mode, the upper schedule header displays the years whereas, the bottom header displays its corresponding months. Refer to the following code example.

#### APP.VUE

```

<template>
  <div>
    <ejs-gantt id="GanttContainer" :dataSource="data" :taskFields =
    "taskFields" :height = "height" :timelineSettings="timelineSettings"></ejs-
    gantt>
  </div>
</template>
<script>
import Vue from "vue";
import { GanttPlugin } from "@syncfusion/ej2-vue-gantt";
import { editingData } from './data-source.js';
Vue.use(GanttPlugin);
export default {
  data: function() {
    return {
      data: [
        {
          TaskID: 1,
          TaskName: 'Project Initiation',
          StartDate: new Date('04/02/2019'),
          EndDate: new Date('04/21/2019'),
          subtasks: [
            { TaskID: 2, TaskName: 'Identify Site location',
              StartDate: new Date('04/02/2019'), Duration: 54, Progress: 50 },
            { TaskID: 3, TaskName: 'Perform Soil test', StartDate:
              new Date('04/02/2019'), Duration: 54, Progress: 50 },

```

```

        { TaskID: 4, TaskName: 'Soil test approval', StartDate:
new Date('04/02/2019'), Duration: 54, Progress: 50 },
    ],
    {
        TaskID: 5,
        TaskName: 'Project Estimation',
        StartDate: new Date('04/02/2019'),
        EndDate: new Date('04/21/2019'),
        subtasks: [
            { TaskID: 6, TaskName: 'Develop floor plan for
estimation', StartDate: new Date('04/04/2019'), Duration: 54, Progress: 50
},
            { TaskID: 7, TaskName: 'List materials', StartDate: new
Date('04/04/2019'), Duration: 54, Progress: 50 },
            { TaskID: 8, TaskName: 'Estimation approval', StartDate:
new Date('04/04/2019'), Duration: 54, Progress: 50 }
        ]
    },
],
height: '450px',
taskFields: {
    id: 'TaskID',
    name: 'TaskName',
    startDate: 'StartDate',
    endDate: 'EndDate',
    duration: 'Duration',
    progress: 'Progress',
    dependency: 'Predecessor',
    child: 'subtasks'
},
timelineSettings: {
    timelineViewMode: 'Year',
    timelineUnitSize: 70
},
},
};
},
};
</script>

```

{% previewsample "page.domainurl/code-snippet/gantt/timeline-cs3" %}

#### Day timeline mode

In the **Day** timeline mode, the upper part of the header displays the days whereas, the bottom schedule header displays its corresponding hours. Refer to the following code example.

#### APP.VUE

```

<template>
  <div>
    <ejs-gantt id="GanttContainer" :dataSource="data" :taskFields =
"taskFields" :height = "height" :timelineSettings="timelineSettings"></ejs-
gantt>
  </div>
</template>
<script>

```

```

import Vue from "vue";
import { GanttPlugin } from "@syncfusion/ej2-vue-gantt";
Vue.use(GanttPlugin);
export default {
  data: function() {
    return {
      data: [
        {
          TaskID: 1,
          TaskName: 'Project Initiation',
          StartDate: new Date('04/02/2019'),
          EndDate: new Date('04/21/2019'),
          isParent: true,
          subtasks: [
            { TaskID: 2, TaskName: 'Identify Site location', StartDate:
new Date('04/02/2019'), Duration: 2, Progress: 50, isParent: false },
            { TaskID: 3, TaskName: 'Perform Soil test', StartDate: new
Date('04/02/2019'), Duration: 2, Progress: 50, resources: [2, 3,
5], isParent: false },
            { TaskID: 4, TaskName: 'Soil test approval', StartDate: new
Date('04/02/2019'), Duration: 2, Predecessor: "2FS", Progress:
50, isParent: false },
          ]
        },
      ],
      height: '450px',
      taskFields: {
        id: 'TaskID',
        name: 'TaskName',
        startDate: 'StartDate',
        endDate: 'EndDate',
        duration: 'Duration',
        progress: 'Progress',
        dependency: 'Predecessor',
        child: 'subtasks'
      },
      timelineSettings: {
        timelineViewMode: 'Day',
      },
    };
  },
};
</script>

```

```
{% previewsample "page.domainurl/code-snippet/gantt/timeline-cs4" %}
```

#### Hour timeline mode

An **Hour** timeline mode tracks the tasks in minutes scale. In this mode, the upper schedule header displays hour scale and the lower schedule header displays its corresponding minutes.

#### APP.VUE

```

<template>
  <div>

```

```

    <ejs-gantt id="GanttContainer" :dataSource="data" :taskFields =
"taskFields" :height = "height" :timelineSettings="timelineSettings"></ejs-
gantt>
  </div>
</template>
<script>
import Vue from "vue";
import { GanttPlugin } from "@syncfusion/ej2-vue-gantt";
Vue.use(GanttPlugin);
export default {
  data: function() {
    return{
      data: [
        {
          TaskID: 1,
          TaskName: 'Project Initiation',
          StartDate: new Date('04/02/2019'),
          EndDate: new Date('04/21/2019'),
          isParent:true,
          subtasks: [
            { TaskID: 2, TaskName: 'Identify Site location', StartDate:
new Date('04/02/2019'), Duration: 2, Progress: 50,isParent:false },
            { TaskID: 3, TaskName: 'Perform Soil test', StartDate: new
Date('04/02/2019'), Duration: 2, Progress: 50, resources: [2, 3,
5],isParent:false },
            { TaskID: 4, TaskName: 'Soil test approval', StartDate: new
Date('04/02/2019'), Duration: 2,Predecessor:"2FS", Progress:
50,isParent:false },
          ]
        },
      ],
      height: '450px',
      taskFields: {
        id: 'TaskID',
        name: 'TaskName',
        startDate: 'StartDate',
        endDate: 'EndDate',
        duration: 'Duration',
        progress: 'Progress',
        dependency: 'Predecessor',
        child: 'subtasks'
      },
      timelineSettings: {
        timelineViewMode:'Hour',
      },
    };
  },
};
</script>

```

{% previewsample "page.domainurl/code-snippet/gantt/timeline-cs5" %}

*Week start day customization*

In the Gantt component, you can customize the week start day using the [weekStartDay](#) property. By default, the [weekStartDay](#) is set to 0, which specifies the Sunday as a start day of the week. But, you can customize the week start day by using the following code example.

**APP.VUE**

```
<template>
  <div>
    <ejs-gantt id="GanttContainer" :dataSource="data" :taskFields =
      "taskFields" :height = "height" :timelineSettings="timelineSettings"></ejs-
      gantt>
    </div>
  </template>
<script>
import Vue from "vue";
import { GanttPlugin } from "@syncfusion/ej2-vue-gantt";
Vue.use(GanttPlugin);
export default {
  data: function() {
    return {
      data: [
        {
          TaskID: 1,
          TaskName: 'Project Initiation',
          StartDate: new Date('04/02/2019'),
          EndDate: new Date('04/21/2019'),
          subtasks: [
            { TaskID: 2, TaskName: 'Identify Site location',
              StartDate: new Date('04/02/2019'), Duration: 4, Progress: 50 },
            { TaskID: 3, TaskName: 'Perform Soil test', StartDate:
              new Date('04/02/2019'), Duration: 4, Progress: 50 },
            { TaskID: 4, TaskName: 'Soil test approval', StartDate:
              new Date('04/02/2019'), Duration: 4, Progress: 50 },
          ]
        },
        {
          TaskID: 5,
          TaskName: 'Project Estimation',
          StartDate: new Date('04/02/2019'),
          EndDate: new Date('04/21/2019'),
          subtasks: [
            { TaskID: 6, TaskName: 'Develop floor plan for
              estimation', StartDate: new Date('04/04/2019'), Duration: 3, Progress: 50 },
            { TaskID: 7, TaskName: 'List materials', StartDate: new
              Date('04/04/2019'), Duration: 3, Progress: 50 },
            { TaskID: 8, TaskName: 'Estimation approval', StartDate:
              new Date('04/04/2019'), Duration: 3, Progress: 50 }
          ]
        },
      ],
      height: '450px',
      taskFields: {
        id: 'TaskID',
        name: 'TaskName',
        startDate: 'StartDate',
        endDate: 'EndDate',

```

```

        duration: 'Duration',
        progress: 'Progress',
        dependency: 'Predecessor',
        child: 'subtasks'
    },
    timelineSettings: {
        timelineViewMode: 'Week',
        weekStartDay: 3
    },
};
},
};
</script>

```

{% previewsample "page.domainurl/code-snippet/gantt/timeline-cs6" %}

#### *Customize automatic timescale update action*

In the Gantt component, the schedule timeline will be automatically updated when the tasks date values are updated beyond the project start date and end date ranges. This can be enabled or disabled using the [updateTimescaleView](#) property.

#### **APP.VUE**

```

<template>
  <div>
    <ejs-gantt id="GanttContainer" :dataSource="data" :taskFields =
    "taskFields" :height = "height" :timelineSettings="timelineSettings"
    :editSettings="editSettings"></ejs-gantt>
  </div>
</template>
<script>
import Vue from "vue";
import { GanttPlugin, Edit } from "@syncfusion/ej2-vue-gantt";
Vue.use(GanttPlugin);
export default {
  data: function() {
    return {
      data: [
        {
          TaskID: 1,
          TaskName: 'Project Initiation',
          StartDate: new Date('04/02/2019'),
          EndDate: new Date('04/21/2019'),
          subtasks: [
            { TaskID: 2, TaskName: 'Identify Site location',
              StartDate: new Date('04/02/2019'), Duration: 4, Progress: 50 },
            { TaskID: 3, TaskName: 'Perform Soil test', StartDate:
              new Date('04/02/2019'), Duration: 4, Progress: 50 },
            { TaskID: 4, TaskName: 'Soil test approval', StartDate:
              new Date('04/02/2019'), Duration: 4, Progress: 50 },
          ]
        },
        {
          TaskID: 5,
          TaskName: 'Project Estimation',
          StartDate: new Date('04/02/2019'),

```

```

        EndDate: new Date('04/21/2019'),
        subtasks: [
            { TaskID: 6, TaskName: 'Develop floor plan for
estimation', StartDate: new Date('04/04/2019'), Duration: 3, Progress: 50 },
            { TaskID: 7, TaskName: 'List materials', StartDate: new
Date('04/04/2019'), Duration: 3, Progress: 50 },
            { TaskID: 8, TaskName: 'Estimation approval', StartDate:
new Date('04/04/2019'), Duration: 3, Progress: 50 }
        ]
    },
],
height: '450px',
taskFields: {
    id: 'TaskID',
    name: 'TaskName',
    startDate: 'StartDate',
    endDate: 'EndDate',
    duration: 'Duration',
    progress: 'Progress',
    dependency: 'Predecessor',
    child: 'subtasks'
},
timelineSettings: {
    updateTimescaleView: false
},
editSettings: {
    allowEditing: true,
    allowTaskbarEditing: true
}
};

},
provide: {
    gantt: [ Edit ]
},
};
</script>

```

{% previewsample "page.domainurl/code-snippet/gantt/timeline-cs7" %}

#### *Timeline cells tooltip*

In the Gantt component, you can enable or disable the mouse hover tooltip of timeline cells using the [timelineSettings.showTooltip](#) property. The default value of this property is `true`. The following code example shows how to enable the timeline cells tooltip in Gantt.

#### **APP.VUE**

```

<template>
  <div>
    <ejs-gantt ref='gantt' :dataSource="data" id="GanttContainer"
:taskFields = "taskFields" :height = "height" :columns="columns"
:timelineSettings="timelineSettings" ></ejs-gantt>
  </div>
</template>
<script>
import Vue from "vue";
import { GanttPlugin } from "@syncfusion/ej2-vue-gantt";

```

```

import { editingData } from './data-source.js';
Vue.use(GanttPlugin);
export default {
  data: function() {
    return{
      data: editingData,
      height: '450px',
      taskFields: {
        id: 'TaskID',
        name: 'TaskName',
        startDate: 'StartDate',
        duration: 'Duration',
        progress: 'Progress',
        child: 'subtasks'
      },
      timelineSettings: {
        showTooltip: true
      },
      columns: [
        { field: 'TaskID', headerText: 'Task ID', textAlign: 'Left',
width: '100' },
        { field: 'TaskName', headerText: 'Task Name', width: '250' },
        { field: 'StartDate', headerText: 'Start Date', width: '150' },
        { field: 'Duration', headerText: 'Duration', width: '150' },
        { field: 'Progress', headerText: 'Progress', width: '150' },
      ]
    };
  },
};
</script>

```

{% previewsample "page.domainurl/code-snippet/gantt/tooltip-cs2" %}

### Top tier and bottom tier in Vue Gantt component

The Gantt component contains two tiers layout in timeline, you can customize the top tier and bottom tier using the [topTier](#) and [bottomTier](#) properties. Timeline tier's unit can be defined by using the [unit](#) property, and the [format](#) property is used to define the date format of timeline cell. The [count](#) property is used to define the number of units to be combined as a single cell and the [formatter](#) property is used to define the custom method to format the date value of timeline cell.

### APP.VUE

```

<template>
  <div>
    <ejs-gantt id="GanttContainer" :dataSource="data" :taskFields =
"taskFields" :height = "height" :timelineSettings="timelineSettings"></ejs-
gantt>
  </div>
</template>
<script>
import Vue from "vue";
import { GanttPlugin } from "@syncfusion/ej2-vue-gantt";
Vue.use(GanttPlugin);
export default {
  data: function() {
    return{

```



```

data: [
  {
    TaskID: 1,
    TaskName: 'Project Initiation',
    StartDate: new Date('04/02/2019'),
    EndDate: new Date('04/21/2019'),
    subtasks: [
      { TaskID: 2, TaskName: 'Identify Site location',
StartDate: new Date('04/02/2019'), Duration: 4, Progress: 50 },
      { TaskID: 3, TaskName: 'Perform Soil test', StartDate:
new Date('04/02/2019'), Duration: 4, Progress: 50 },
      { TaskID: 4, TaskName: 'Soil test approval', StartDate:
new Date('04/02/2019'), Duration: 4, Progress: 50 },
    ]
  },
  {
    TaskID: 5,
    TaskName: 'Project Estimation',
    StartDate: new Date('04/02/2019'),
    EndDate: new Date('04/21/2019'),
    subtasks: [
      { TaskID: 6, TaskName: 'Develop floor plan for
estimation', StartDate: new Date('04/04/2019'), Duration: 3, Progress: 50 },
      { TaskID: 7, TaskName: 'List materials', StartDate: new
Date('04/04/2019'), Duration: 3, Progress: 50 },
      { TaskID: 8, TaskName: 'Estimation approval', StartDate:
new Date('04/04/2019'), Duration: 3, Progress: 50 }
    ]
  },
],
height: '450px',
taskFields: {
  id: 'TaskID',
  name: 'TaskName',
  startDate: 'StartDate',
  endDate: 'EndDate',
  duration: 'Duration',
  progress: 'Progress',
  dependency: 'Predecessor',
  child: 'subtasks'
},
timelineSettings: {
  topTier: {
    format: 'MMM',
    unit: 'Month'
  },
  bottomTier: {
    unit: 'Day',
    count: 2
  }
},
};
};
</script>

```

```
{% previewsample "page.domainurl/code-snippet/gantt/timeline-cs8" %}
```

### Combining timeline cells

In the Gantt component, the timeline cells in top and bottom tiers can be combined with number of timeline units. This can be achieved by using the [topTier.count](#) and [bottomTier.count](#) properties. Refer to the following sample.

### APP.VUE

```
<template>
  <div>
    <ejs-gantt id="GanttContainer" :dataSource="data" :taskFields =
    "taskFields" :height = "height" :timelineSettings="timelineSettings"></ejs-
    gantt>
  </div>
</template>
<script>
import Vue from "vue";
import { GanttPlugin } from "@syncfusion/ej2-vue-gantt";
Vue.use(GanttPlugin);
export default {
  data: function() {
    return{
      data: [
        {
          TaskID: 1,
          TaskName: 'Project Initiation',
          StartDate: new Date('04/02/2019'),
          EndDate: new Date('09/16/2019'),
          subtasks: [
            { TaskID: 2, TaskName: 'Identify Site location', StartDate:
            new Date('04/02/2019'), Duration: 120, Progress: 50 },
            { TaskID: 3, TaskName: 'Perform Soil test', StartDate: new
            Date('04/02/2019'), Duration: 120, Progress: 50 },
            { TaskID: 4, TaskName: 'Soil test approval', StartDate: new
            Date('04/02/2019'), Duration: 120 , Progress: 50 },
          ]
        },
        {
          TaskID: 5,
          TaskName: 'Project Estimation',
          StartDate: new Date('04/04/2019'),
          EndDate: new Date('09/18/2019'),
          subtasks: [
            { TaskID: 6, TaskName: 'Develop floor plan for estimation',
            StartDate: new Date('04/04/2019'), Duration: 120, Progress: 50 },
            { TaskID: 7, TaskName: 'List materials', StartDate: new
            Date('04/04/2019'), Duration: 120, Progress: 50 },
            { TaskID: 8, TaskName: 'Estimation approval', StartDate: new
            Date('04/04/2019'), Duration: 120, Progress: 50 }
          ]
        }
      ],
      height: '450px',
      taskFields: {
        id: 'TaskID',
        name: 'TaskName',

```

```

        startDate: 'StartDate',
        endDate: 'EndDate',
        duration: 'Duration',
        progress: 'Progress',
        dependency: 'Predecessor',
        child: 'subtasks'
    },
    timelineSettings: {
        timelineUnitSize: 100,
        topTier: {
            unit: 'Year'
        },
        bottomTier: {
            unit: 'Month',
            count: 6
        }
    },
    };
},
};
</script>

```

{% previewsample "page.domainurl/code-snippet/gantt/timeline-cs9" %}

#### *Format value of timeline cell*

In the Gantt component, you can format the value of top and bottom timeline cells using the standard date format string or the custom formatter method. This can be done using the [topTier.format](#), [topTier.formatter](#), [bottomTier.format](#) and [bottomTier.formatter](#) properties. The following example shows how to use the formatter method for timeline cells.

#### **APP.VUE**

```

<template>
  <div>
    <ejs-gantt id="GanttContainer" :dataSource="data" :taskFields =
      "taskFields" :height = "height" :timelineSettings="timelineSettings"
      :projectStartDate="projectStartDate" :projectEndDate="projectEndDate"></ejs-
      gantt>
    </div>
  </template>
<script>
import Vue from "vue";
import { GanttPlugin } from "@syncfusion/ej2-vue-gantt";
Vue.use(GanttPlugin);
export default {
  data: function() {
    return {
      data: [
        {
          TaskID: 1,
          TaskName: 'Project Initiation',
          StartDate: new Date('04/02/2019'),
          EndDate: new Date('09/16/2019'),
          subtasks: [
            { TaskID: 2, TaskName: 'Identify Site location', StartDate:
              new Date('04/02/2019'), Duration: 120, Progress: 50 },

```

```

        { TaskID: 3, TaskName: 'Perform Soil test', StartDate: new
Date('04/02/2019'), Duration: 120, Progress: 50 },
        { TaskID: 4, TaskName: 'Soil test approval', StartDate: new
Date('04/02/2019'), Duration: 120 , Progress: 50 },
    ]
},
{
    TaskID: 5,
    TaskName: 'Project Estimation',
    StartDate: new Date('04/04/2019'),
    EndDate: new Date('09/18/2019'),
    subtasks: [
        { TaskID: 6, TaskName: 'Develop floor plan for estimation',
StartDate: new Date('04/04/2019'), Duration: 120, Progress: 50 },
        { TaskID: 7, TaskName: 'List materials', StartDate: new
Date('04/04/2019'), Duration: 120, Progress: 50 },
        { TaskID: 8, TaskName: 'Estimation approval', StartDate: new
Date('04/04/2019'), Duration: 120, Progress: 50 }
    ]
},
],
    height: '450px',
    taskFields: {
        id: 'TaskID',
        name: 'TaskName',
        startDate: 'StartDate',
        endDate: 'EndDate',
        duration: 'Duration',
        progress: 'Progress',
        dependency: 'Predecessor',
        child: 'subtasks'
    },
    timelineSettings: {
        topTier: {
            unit: 'Month',
            count: 3,
            formatter: (date) => {
                var month = date.getMonth();
                if (month >= 0 && month <= 2) {
                    return 'Q1';
                } else if (month >= 3 && month <= 5) {
                    return 'Q2';
                } else if (month >= 6 && month <= 8) {
                    return 'Q3';
                } else {
                    return 'Q4';
                }
            }
        },
        bottomTier: {
            unit: 'Month',
            format: 'MMM'
        }
    },
    projectStartDate: new Date('01/04/2019'),
    projectEndDate: new Date('12/30/2019')
};

```

```

},
}
</script>

```

```
{% previewsample "page.domainurl/code-snippet/gantt/timeline-cs10" %}
```

### Timeline cell width

In the Gantt component, you can define the width value of timeline cell using the [timelineSettings.timelineUnitSize](#) property. This value will be set to the bottom timeline cell, and the width value of top timeline cell will be calculated automatically based on bottom tier cell width using the [topTier.unit](#) and [timelineSettings.timelineUnitSize](#) properties. Refer to the following example.

### APP.VUE

```

<template>
  <div>
    <ejs-gantt id="GanttContainer" :dataSource="data" :taskFields =
      "taskFields" :height = "height" :timelineSettings="timelineSettings"></ejs-
      gantt>
    </div>
  </template>
<script>
import Vue from "vue";
import { GanttPlugin } from "@syncfusion/ej2-vue-gantt";
Vue.use(GanttPlugin);
export default {
  data: function() {
    return {
      data: [
        {
          TaskID: 1,
          TaskName: 'Project Initiation',
          StartDate: new Date('04/02/2019'),
          EndDate: new Date('04/21/2019'),
          subtasks: [
            { TaskID: 2, TaskName: 'Identify Site location',
              StartDate: new Date('04/02/2019'), Duration: 4, Progress: 50 },
            { TaskID: 3, TaskName: 'Perform Soil test', StartDate:
              new Date('04/02/2019'), Duration: 4, Progress: 50 },
            { TaskID: 4, TaskName: 'Soil test approval', StartDate:
              new Date('04/02/2019'), Duration: 4, Progress: 50 },
          ]
        },
        {
          TaskID: 5,
          TaskName: 'Project Estimation',
          StartDate: new Date('04/02/2019'),
          EndDate: new Date('04/21/2019'),
          subtasks: [
            { TaskID: 6, TaskName: 'Develop floor plan for
              estimation', StartDate: new Date('04/04/2019'), Duration: 3, Progress: 50 },
            { TaskID: 7, TaskName: 'List materials', StartDate: new
              Date('04/04/2019'), Duration: 3, Progress: 50 },
            { TaskID: 8, TaskName: 'Estimation approval', StartDate:
              new Date('04/04/2019'), Duration: 3, Progress: 50 }
          ]
        }
      ]
    }
  }
}

```

```

    },
    ],
    height: '450px',
    taskFields: {
        id: 'TaskID',
        name: 'TaskName',
        startDate: 'StartDate',
        endDate: 'EndDate',
        duration: 'Duration',
        progress: 'Progress',
        dependency: 'Predecessor',
        child: 'subtasks'
    },
    timelineSettings: {
        timelineUnitSize: 200
    },
    },
    };
},
};
</script>

```

{% previewsample "page.domainurl/code-snippet/gantt/timeline-cs11" %}

### Zooming in Vue Gantt component

The zooming support provides options to increase or decrease the width of timeline cells and also provides options to change the timeline units dynamically. This support enables you to view the tasks in a project clearly from minute to decade timespan. To enable the zooming features, define the **ZoomIn**, **ZoomOut**, and **ZoomToFit** items to toolbar items collections, and this action can be performed on external actions such as button click using the [zoomIn](#), [zoomOut](#), and [fitToProject](#) built-in methods. The following zooming options are available to view the project:

#### Zoom in

This support is used to increase the timeline width and timeline unit from years to minutetimespan. When the **ZoomIn** icon was clicked, the timeline cell width is increased when the cellsize exceeds the specified range and the timeline unit is changed based on the current zoom levels.

#### Zoom out

This support is used to increase the timeline width and timeline unit from minutes to years timespan. When the **ZoomOut** icon was clicked, the timeline cell width is decreased when the cell size falls behind the specified range and the timeline view mode is changed based on the current zooming levels.

#### Zoom to fit

This support is used to view all the tasks available in a project within available area on the chart part of Gantt. When users click the **ZoomToFit** icon, then all the tasks are rendered within the available chart container width.

### APP.VUE

```

<template>
  <div>
    <ejs-gantt ref='gantt' id="GanttContainer" :dataSource="data"
    :taskFields = "taskFields" :height = "height" :toolbar="toolbar"></ejs-
    gantt>
  </div>

```

```

</template>
<script>
import Vue from "vue";
import { GanttPlugin, Toolbar } from "@syncfusion/ej2-vue-gantt";
import { projectNewData } from './data-source.js';
Vue.use(GanttPlugin);
export default {
  data: function() {
    return{
      data: projectNewData,
      height: '450px',
      taskFields: {
        id: 'TaskID',
        name: 'TaskName',
        startDate: 'StartDate',
        duration: 'Duration',
        progress: 'Progress',
        dependency: 'Predecessor',
        child: 'subtasks'
      },
      toolbar: ['ZoomIn', 'ZoomOut', 'ZoomToFit'],
    };
  },
  provide: {
    gantt: [Toolbar ]
  }
};
</script>

```

{% previewsample "page.domainurl/code-snippet/gantt/zooming-cs1" %}

### Customizing zooming levels

In Gantt, the zoom in and zoom out actions are performed based on the predefined zooming levels in the `zoomingLevels` property. You can customize the zooming actions by defining the required zooming collection to the `zoomingLevels` property.

### APP.VUE

```

<template>
  <div>
    <ejs-gantt ref='gantt' id="GanttContainer" :dataSource="data"
      :dataBound="dataBound" :taskFields = "taskFields" :height = "height"
      :toolbar="toolbar"></ejs-gantt>
  </div>
</template>
<script>
import Vue from "vue";
import { GanttPlugin, Toolbar } from "@syncfusion/ej2-vue-gantt";
import { projectNewData } from './data-source.js';
let customZoomingLevels = [{
  topTier: { unit: 'Month', format: 'MMM, yy', count: 1 },
  bottomTier: { unit: 'Week', format: 'dd', count: 1 },
  timelineUnitSize: 33, level: 0,
  timelineViewMode: 'Month', weekStartDay: 0,
  updateTimescaleView: true, weekendBackground: null, showTooltip: true
},

```

```

    {
      topTier: { unit: 'Month', format: 'MMM, yyyy', count: 1 },
      bottomTier: { unit: 'Week', format: 'dd MMM', count: 1 },
      timelineUnitSize: 66, level: 1,
      timelineViewMode: 'Month', weekStartDay: 0,
      updateTimescaleView: true, weekendBackground: null, showTooltip: true
    },
    {
      topTier: { unit: 'Month', format: 'MMM, yyyy', count: 1 },
      bottomTier: { unit: 'Week', format: 'dd MMM', count: 1 },
      timelineUnitSize: 99, level: 2,
      timelineViewMode: 'Month', weekStartDay: 0,
      updateTimescaleView: true, weekendBackground: null, showTooltip: true
    },
    {
      topTier: { unit: 'Week', format: 'MMM dd, yyyy', count: 1 },
      bottomTier: { unit: 'Day', format: 'd', count: 1 },
      timelineUnitSize: 33, level: 3,
      timelineViewMode: 'Week', weekStartDay: 0,
      updateTimescaleView: true, weekendBackground: null, showTooltip: true
    },
    {
      topTier: { unit: 'Week', format: 'MMM dd, yyyy', count: 1 },
      bottomTier: { unit: 'Day', format: 'd', count: 1 },
      timelineUnitSize: 66, level: 4,
      timelineViewMode: 'Week', weekStartDay: 0,
      updateTimescaleView: true, weekendBackground: null, showTooltip: true
    },
    {
      topTier: { unit: 'Day', format: 'E dd yyyy', count: 1 },
      bottomTier: { unit: 'Hour', format: 'hh a', count: 12 },
      timelineUnitSize: 66, level: 5,
      timelineViewMode: 'Day', weekStartDay: 0,
      updateTimescaleView: true, weekendBackground: null, showTooltip: true
    },
    {
      topTier: { unit: 'Day', format: 'E dd yyyy', count: 1 },
      bottomTier: { unit: 'Hour', format: 'hh a', count: 6 },
      timelineUnitSize: 99, level: 6,
      timelineViewMode: 'Day', weekStartDay: 0,
      updateTimescaleView: true, weekendBackground: null, showTooltip: true
    },
  ];
Vue.use(GanttPlugin);
export default {
  data: function() {
    return {
      data: projectNewData,
      height: '450px',
      taskFields: {
        id: 'TaskID',
        name: 'TaskName',
        startDate: 'StartDate',
        duration: 'Duration',
        progress: 'Progress',
        dependency: 'Predecessor',
        child: 'subtasks'
      }
    }
  }
}

```



```

    },
    toolbar: ['ZoomIn', 'ZoomOut', 'ZoomToFit'],
    dataBound: function () {
        var ganttObj =
document.getElementById('GanttContainer').ej2_instances[0];
        ganttObj.zoomingLevels = customZoomingLevels;
    },
    };
},
provide: {
    gantt: [Toolbar ]
}
};
</script>

```

{% previewsample "page.domainurl/code-snippet/gantt/zooming-cs2" %}

### *Zoom action by methods*

You can perform the various zoom actions dynamically or on external click action using the following methods:

- Zoom in - [zoomIn](#)
- Zoom out - [zoomOut](#)
- Fit to project - [fitToProject](#)

### **APP.VUE**

```

<template>
  <div>
    <ejs-button id="zoomIn" cssClass="e-info" v-
on:click.native="zoomIn">ZoomIn</ejs-button>
    <ejs-button id="zoomOut" cssClass="e-info" v-
on:click.native="zoomOut">ZoomOut</ejs-button>
    <ejs-button id="fitToProject" cssClass="e-info" v-
on:click.native="fitToProject">FitToProject</ejs-button>
    <ejs-gantt ref='gantt' id="GanttContainer" :dataSource="data"
:taskFields = "taskFields" :height = "height"></ejs-gantt>
  </div>
</template>
<script>
import Vue from "vue";
import { GanttPlugin } from "@syncfusion/ej2-vue-gantt";
import { ButtonPlugin } from "@syncfusion/ej2-vue-buttons";
import { projectNewData } from './data-source.js';
Vue.use(GanttPlugin);
Vue.use(ButtonPlugin);
export default {
  data: function() {
    return{
      data: projectNewData,
      height: '450px',
      taskFields: {
        id: 'TaskID',
        name: 'TaskName',
        startDate: 'StartDate',

```

```

        duration: 'Duration',
        progress: 'Progress',
        dependency: 'Predecessor',
        child: 'subtasks'
    }
    };
},
methods: {
    zoomIn: function() {
        this.$refs.gantt.zoomIn();
    },
    zoomOut: function() {
        this.$refs.gantt.zoomOut();
    },
    fitToProject: function() {
        this.$refs.gantt.fitToProject();
    }
},
};
</script>

```

{% previewsample "page.domainurl/code-snippet/gantt/zooming-cs3" %}

### Event markers in Vue Gantt component

The event markers in the Gantt component is used to highlight the important events in a project. Event markers can be initialized by using the [eventMarkers](#) property, and you can define date and label for the event markers using the [day](#) and [label](#) properties. You can also customize it using the [cssClass](#) properties. The following code example shows how to add event markers in the Gantt component.

To highlight the days, inject the [DayMarkers](#) module in the `provide` section.

### APP.VUE

```

<template>
  <div>
    <ejs-gantt ref='gantt' id="GanttContainer" :dataSource="data"
    :taskFields = "taskFields" :height = "height" :eventMarkers =
    "eventMarkers"></ejs-gantt>
  </div>
</template>
<script>
import Vue from "vue";
import { GanttPlugin, DayMarkers } from "@syncfusion/ej2-vue-gantt";
import { editingData } from './data-source.js';
Vue.use(GanttPlugin);
export default {
  data: function() {
    return {
      data: editingData,
      height: '450px',
      taskFields: {
        id: 'TaskID',
        name: 'TaskName',
        startDate: 'StartDate',
        resourceInfo: 'resources',
        duration: 'Duration',

```

```

        progress: 'Progress',
        dependency: 'Predecessor',
        child: 'subtasks'
    },
    eventMarkers: [
        {
            day: '04/10/2019',
            cssClass: 'e-custom-event-marker',
            label: 'Project approval and kick-off'
        }
    ]
    };
},
provide: {
    gantt: [DayMarkers]
}
};
</script>

```

{% previewsample "page.domainurl/code-snippet/gantt/eventmarkers-cs1" %}

### Holidays in Vue Gantt component

Non-working days in a project can be displayed in the Gantt component using the [holidays](#) property. Each holiday can be defined with the following properties:

- [from](#): Defines start date of the holiday(s).
- [to](#): Defines end date of the holiday(s).
- [label](#): Defines the description or label for the holiday.
- [cssClass](#): Formats the holidays label in the Gantt chart.

To highlight the days, inject the [DayMarkers](#) module in the `provide` section.

The following code example shows how to display the non-working days in the Gantt component.

### APP.VUE

```

<template>
  <div>
    <ejs-gantt ref='gantt' id="GanttContainer" :dataSource="data"
    :taskFields = "taskFields" :height = "height" :holidays = "holidays"></ejs-gantt>
  </div>
</template>
<script>
import Vue from "vue";
import { GanttPlugin, DayMarkers } from "@syncfusion/ej2-vue-gantt";
import { projectNewData } from './data-source.js';
Vue.use(GanttPlugin);
export default {
  data: function() {
    return{
      data: projectNewData,
      height: '450px',
      taskFields: {
        id: 'TaskID',

```

```

        name: 'TaskName',
        startDate: 'StartDate',
        duration: 'Duration',
        progress: 'Progress',
        child: 'subtasks'
    },
    holidays: [{
        from: "04/04/2019",
        to: "04/05/2019",
        label: " Public holidays",
        cssClass: "e-custom-holiday"
    },
    {
        from: "04/12/2019",
        to: "04/12/2019",
        label: " Local holidays",
        cssClass: "e-custom-holiday"
    }
    ]
};
},
provide: {
    gantt: [DayMarkers]
}
};
</script>

```

{% previewsample "page.domainurl/code-snippet/gantt/holidays-cs1" %}

## Splitter in Vue Gantt component

### Splitter

In the Gantt component, the Splitter separates the TreeGrid section from the Chart section. You can change the position of the Splitter when loading the Gantt component using the [splitterSettings](#) property. By splitting the TreeGrid from the chart, the width of the TreeGrid and chart sections will vary in the component. The [splitterSettings.position](#) property denotes the percentage of the TreeGrid section's width to be rendered and this property supports both pixels and percentage values. You can define the splitter position as column index value using the [splitterSettings.columnIndex](#) property. You can also define the splitter position with built-in splitter view modes by using the [splitterSettings.view](#) property. The following list is the possible values for this property:

- **Default:** Shows Grid side and Gantt side.
- **Grid:** Shows Grid side alone in Gantt.
- **Chart:** Shows chart side alone in Gantt.

### APP.VUE

```

<template>
    <div>
        <ejs-gantt id="GanttContainer" :dataSource="data" :taskFields =
"taskFields" :height = "height" :splitterSettings="splitterSettings"></ejs-
gantt>
    </div>
</template>
<script>

```

```
import Vue from "vue";
import { GanttPlugin } from "@syncfusion/ej2-vue-gantt";
import { editingData } from './data-source.js';
Vue.use(GanttPlugin);
export default {
  data: function() {
    return {
      data: editingData,
      height: '450px',
      taskFields: {
        id: 'TaskID',
        name: 'TaskName',
        startDate: 'StartDate',
        duration: 'Duration',
        progress: 'Progress',
        child: 'subtasks'
      },
      splitterSettings: {
        position: "50%"
      },
    };
  },
};
</script>
```

{% previewsample "page.domainurl/code-snippet/gantt/appearance-customization-cs8" %}

### Change splitter position dynamically

In Gantt, we can change the splitter position dynamically by using [setSplitterPosition](#) method. Either We can change the splitter position with splitter position or columnIndex values by passing these values as arguments to [setSplitterPosition](#) method. The following code example shows how to use this methods.

### APP.VUE

```
<template>
  <div>
    <ejs-button id="changebypostion" cssClass="e-info" v-
on:click.native="change">Change By Postion</ejs-button>
    <br><br>
    <ejs-button id="changebyindex" cssClass="e-info" v-
on:click.native="changei">Change By Index</ejs-button>
    <br><br>
    <table>
      <tr>
        <td style="width: 70%">
          <ejs-dropdownlist id="splitter-type" value='Default'
:dataSource="dataSource" :fields = "fields" :change ="change"> Splitter View
</ejs-dropdownlist>
        </td>
      </tr>
    </table>
    <ejs-gantt id="GanttContainer" :dataSource="data" :taskFields =
"taskFields" :height = "height"></ejs-gantt>
  </div>
</template>
<script>
```

```

import Vue from "vue";
import { GanttPlugin } from "@syncfusion/ej2-vue-gantt";
import { ButtonPlugin } from "@syncfusion/ej2-vue-buttons";
import { DropDownListPlugin } from "@syncfusion/ej2-vue-dropdowns";
import { editingData } from './data-source.js';
Vue.use(GanttPlugin);
Vue.use(ButtonPlugin);
Vue.use(DropDownListPlugin);
export default {
  data: function() {
    return{
      data: editingData,
      height: '450px',
      taskFields: {
        id: 'TaskID',
        name: 'TaskName',
        startDate: 'StartDate',
        duration: 'Duration',
        progress: 'Progress',
        child: 'subtasks'
      },
      dataSource: [
        { id: 'Default', mode: 'Default' },
        { id: 'Grid', mode: 'Grid' },
        { id: 'Chart', mode: 'Chart' },
      ],
      fields: { text: 'mode', value: 'id' },
    };
  },
  methods: {
    changep: function(e){
      var ganttChart =
document.getElementById('GanttContainer').ej2_instances[0];
      ganttChart.setSplitterPosition('50%', 'position');
    }
    changei: function(e){
      var ganttChart =
document.getElementById('GanttContainer').ej2_instances[0];
      ganttChart.setSplitterPosition(1, 'columnIndex');
    }
    change: function (e) {
      var ganttChart =
document.getElementById('GanttContainer').ej2_instances[0];
      var viewType = e.value;
      ganttChart.setSplitterPosition(viewType, 'view');
    }
  },
};
</script>

```

{% previewsample "page.domainurl/code-snippet/gantt/appearance-customization-cs9" %}

## Scheduling Tasks

### Task scheduling in Vue Gantt component

The Gantt provides support for automatic and manual task scheduling modes. It is used to indicate whether the start date and end date of all the tasks will be automatically validated or not. [taskMode](#) is the property used to change the schedule mode of a task.

The Gantt control supports three types of mode. They are:

- **Auto**: All the tasks are automatically validate.
- **Manual**: All the tasks are manually validate by the user.
- **Custom**: Both Auto and Manual tasks are render by mapped from data source.

Note: The default value of [taskMode](#) is [Link to the Video](#).

To learn about Gantt Chart Scheduling Concepts, you can check on this video:

### *Automatically scheduled tasks*

When the [taskMode](#) property is set as **Auto**, the start date and end date of all the tasks in the project will be automatically validated. That is, dates are validated based on various factors such as working time, holidays, weekends and predecessors.

### **APP.VUE**

```
<template>
  <div>
    <ejs-gantt ref = 'gantt' id = "GanttContainer" :dataSource = "data"
    :taskFields = "taskFields" :treeColumnIndex = "1" :height = "height"
    :editSettings = "editSettings" :toolbar = "toolbar" :taskMode= "taskMode" >
    </ejs-gantt>
  </div>
</template>
<script>
import Vue from "vue";
import { GanttPlugin, Edit, Selection, Toolbar } from "@syncfusion/ej2-vue-gantt";
Vue.use(GanttPlugin);
export default {
  data: function () {
    return {
      data: [
        {
          TaskID: 1,
          TaskName: 'Project initiation',
          StartDate: new Date('03/29/2019'),
          EndDate: new Date('04/21/2019'),
          subtasks: [
            {
              TaskID: 2, TaskName: 'Identify site location', StartDate:
new Date('03/29/2019'), Duration: 2,
              Progress: 30, Work: 16, resources: [{ resourceId: 1, Unit:
70 }, 6]
            },
            {

```

```

        TaskID: 3, TaskName: 'Perform soil test', StartDate: new
Date('03/29/2019'), Duration: 4,
        resources: [2, 3, 5], Work: 96
    },
    {
        TaskID: 4, TaskName: 'Soil test approval', StartDate: new
Date('03/29/2019'), Duration: 1,
        Work: 16, resources: [8, { resourceId: 9, Unit: 50 }],
Progress: 30
    },
]
},
{
    TaskID: 5,
    TaskName: 'Project estimation', StartDate: new Date('03/29/2019'),
EndDate: new Date('04/21/2019'),
    subtasks: [
        {
            TaskID: 6, TaskName: 'Develop floor plan for estimation',
StartDate: new Date('03/29/2019'),
            Duration: 3, Progress: 30, resources: [{ resourceId: 4,
Unit: 50 }], Work: 30
        },
        {
            TaskID: 7, TaskName: 'List materials', StartDate: new
Date('04/01/2019'), Duration: 3,
            Work: 48, resources: [4, 8]
        },
        {
            TaskID: 8, TaskName: 'Estimation approval', StartDate: new
Date('04/01/2019'),
            Duration: 2, Work: 60, resources: [12, { resourceId: 5,
Unit: 70 }]
        }
    ]
},
{
    TaskID: 9, TaskName: 'Sign contract', StartDate: new
Date('04/01/2019'), Duration: 1,
    Progress: 30, resources: [12], Work: 24
}
],
    height: '450px',
    taskMode: 'Auto',
    taskFields: {
        id: 'TaskID',
        name: 'TaskName',
        startDate: 'StartDate',
        duration: 'Duration',
        progress: 'Progress',
        endDate: 'EndDate',
        dependency: 'Predecessor',
        child: 'subtasks'
    },
    editSettings: {
        allowEditing: true,
        allowDeleting: true,

```



```

        allowTaskbarEditing: true,
        showDeleteConfirmDialog: true
    },
    toolbar: ['Add', 'Edit', 'Update', 'Delete', 'Cancel',
'ExpandAll', 'CollapseAll', 'Search'],
    };
},
provide: {
    gantt: [Edit, Selection, Toolbar]
}
};
</script>

```

{% previewsample "page.domainurl/code-snippet/gantt/task-scheduling-cs1" %}

### *Manually scheduled tasks*

When the [taskMode](#) property is set as **Manual**, the start date and end date of all the tasks in the project will be same as given in the data source. That is, dates are not validated based on various factors such as dependencies between tasks, holidays, weekends, working time.

We can restrict this mode in predecessor validation alone. That is, we can automatically validate the dates based on predecessor values by enabling the [validateManualTasksOnLinking](#) property.

### **APP.VUE**

```

<template>
  <div>
    <ejs-gantt ref = 'gantt' id = "GanttContainer" :dataSource = "data"
:taskFields = "taskFields" :treeColumnIndex = "1" :height = "height"
:editSettings = "editSettings" :toolbar = "toolbar" :taskMode= "taskMode" >
  </ejs-gantt>
  </div>
</template>
<script>
import Vue from "vue";
import { GanttPlugin, Edit, Selection, Toolbar } from "@syncfusion/ej2-vue-
gantt";
Vue.use(GanttPlugin);
export default {
  data: function () {
    return {
      data: [
        {
          TaskID: 1,
          TaskName: 'Project initiation',
          StartDate: new Date('03/29/2019'),
          EndDate: new Date('04/21/2019'),
          subtasks: [
            {
              TaskID: 2, TaskName: 'Identify site location', StartDate:
new Date('03/29/2019'), Duration: 2,
              Progress: 30, Work: 16, resources: [{ resourceId: 1, Unit:
70 }, 6]
            },
            {

```

```

        TaskID: 3, TaskName: 'Perform soil test', StartDate: new
Date('03/29/2019'), Duration: 4,
        resources: [2, 3, 5], Work: 96
    },
    {
        TaskID: 4, TaskName: 'Soil test approval', StartDate: new
Date('03/29/2019'), Duration: 1,
        Work: 16, resources: [8, { resourceId: 9, Unit: 50 }],
Progress: 30
    },
]
},
{
    TaskID: 5,
    TaskName: 'Project estimation', StartDate: new Date('03/29/2019'),
EndDate: new Date('04/21/2019'),
    subtasks: [
        {
            TaskID: 6, TaskName: 'Develop floor plan for estimation',
StartDate: new Date('03/29/2019'),
            Duration: 3, Progress: 30, resources: [{ resourceId: 4,
Unit: 50 }], Work: 30
        },
        {
            TaskID: 7, TaskName: 'List materials', StartDate: new
Date('04/01/2019'), Duration: 3,
            Work: 48, resources: [4, 8]
        },
        {
            TaskID: 8, TaskName: 'Estimation approval', StartDate: new
Date('04/01/2019'),
            Duration: 2, Work: 60, resources: [12, { resourceId: 5,
Unit: 70 }]
        }
    ]
},
{
    TaskID: 9, TaskName: 'Sign contract', StartDate: new
Date('04/01/2019'), Duration: 1,
    Progress: 30, resources: [12], Work: 24
}
],
    height: '450px',
    taskMode: 'Manual',
    taskFields: {
        id: 'TaskID',
        name: 'TaskName',
        startDate: 'StartDate',
        duration: 'Duration',
        progress: 'Progress',
        endDate: 'EndDate',
        dependency: 'Predecessor',
        child: 'subtasks'
    },
    editSettings: {
        allowEditing: true,
        allowDeleting: true,

```

```

        allowTaskbarEditing: true,
        showDeleteConfirmDialog: true
    },
    toolbar: ['Add', 'Edit', 'Update', 'Delete', 'Cancel',
'ExpandAll', 'CollapseAll', 'Search'],
    };
},
provide: {
    gantt: [Edit, Selection, Toolbar]
}
};
</script>

```

{% previewsample "page.domainurl/code-snippet/gantt/task-scheduling-cs2" %}

### Custom

When the [taskMode](#) property is set as Custom, the scheduling mode for each tasks will be mapped from the data source field. The [Boolean](#) property [taskFields.manual](#) is used to map the manual scheduling mode field from the data source.

### APP.VUE

```

<template>
  <div>
    <ejs-gantt ref = 'gantt' id = "GanttContainer" :dataSource = "data"
:taskFields = "taskFields" :columns= "columns" :treeColumnIndex = "1"
:height = "height" :editSettings = "editSettings" :toolbar = "toolbar"
:taskMode= "taskMode" > </ejs-gantt>
  </div>
</template>
<script>
import Vue from "vue";
import { GanttPlugin, Edit, Selection, Toolbar } from "@syncfusion/ej2-vue-gantt";
Vue.use(GanttPlugin);
export default {
  data: function () {
    return {
      data: [
        {
          'TaskID': 1,
          'TaskName': 'Parent Task 1',
          'StartDate': new Date('02/27/2017'),
          'EndDate': new Date('03/03/2017'),
          'Progress': '40',
          'isManual': true,
          'Children': [
            {
              'TaskID': 2, 'TaskName': 'Child Task 1',
              'StartDate': new Date('02/27/2017'),
              'EndDate': new Date('03/03/2017'), 'Progress':
              '40'
            },
            {
              'TaskID': 3, 'TaskName': 'Child Task 2',
              'StartDate': new Date('02/26/2017'),

```

```

        'EndDate': new Date('03/03/2017'), 'Progress':
'40', 'isManual': true
    },
    {
        'TaskID': 4, 'TaskName': 'Child Task 3',
        'StartDate': new Date('02/27/2017'),
        'EndDate': new Date('03/03/2017'), 'Duration':
5, 'Progress': '40',
    }
]
},
{
    'TaskID': 5,
    'TaskName': 'Parent Task 2',
    'StartDate': new Date('03/05/2017'),
    'EndDate': new Date('03/09/2017'),
    'Progress': '40',
    'isManual': true,
    'Children': [
        {
            'TaskID': 6, 'TaskName': 'Child Task 1',
            'StartDate': new Date('03/06/2017'),
            'EndDate': new Date('03/09/2017'), 'Progress':
'40'
        },
        {
            'TaskID': 7, 'TaskName': 'Child Task 2',
            'StartDate': new Date('03/06/2017'),
            'EndDate': new Date('03/09/2017'), 'Progress':
'40',
        },
        {
            'TaskID': 8, 'TaskName': 'Child Task 3',
            'StartDate': new Date('02/28/2017'),
            'EndDate': new Date('03/05/2017'), 'Progress':
'40', 'isManual': true
        },
        {
            'TaskID': 9, 'TaskName': 'Child Task 4',
            'StartDate': new Date('03/04/2017'),
            'EndDate': new Date('03/09/2017'), 'Progress':
'40', 'isManual': true
        }
    ]
},
{
    'TaskID': 10,
    'TaskName': 'Parent Task 3',
    'StartDate': new Date('03/13/2017'),
    'EndDate': new Date('03/17/2017'),
    'Progress': '40',
    'Children': [
        {
            'TaskID': 11, 'TaskName': 'Child Task 1',
            'StartDate': new Date('03/13/2017'),
            'EndDate': new Date('03/17/2017'), 'Progress':
'40'
        }
    ]
}

```

```

        },
        {
            'TaskID': 12, 'TaskName': 'Child Task 2',
            'StartDate': new Date('03/13/2017'),
            'EndDate': new Date('03/17/2017'), 'Progress':
            '40',
        },
        {
            'TaskID': 13, 'TaskName': 'Child Task 3',
            'StartDate': new Date('03/13/2017'),
            'EndDate': new Date('03/17/2017'), 'Progress':
            '40',
        },
        {
            'TaskID': 14, 'TaskName': 'Child Task 4',
            'StartDate': new Date('03/12/2017'),
            'EndDate': new Date('03/17/2017'), 'Progress':
            '40', 'isManual': true
        },
        {
            'TaskID': 15, 'TaskName': 'Child Task 5',
            'StartDate': new Date('03/13/2017'),
            'EndDate': new Date('03/17/2017'), 'Progress':
            '40'
        }
    ]
},
height: '450px',
taskMode: 'Custom',
taskFields: {
    id: 'TaskID',
    name: 'TaskName',
    startDate: 'StartDate',
    duration: 'Duration',
    progress: 'Progress',
    endDate: 'EndDate',
    dependency: 'Predecessor',
    child: 'Children',
    manual: 'isManual'
},
editSettings: {
    allowEditing: true,
    allowDeleting: true,
    allowTaskbarEditing: true,
    showDeleteConfirmDialog: true
},
toolbar: ['Add', 'Edit', 'Update', 'Delete', 'Cancel',
'ExpandAll', 'CollapseAll', 'Search'],
columns: [
    { field: 'TaskID', visible: false },
    { field: 'TaskName' },
    { field: 'isManual' }
],
};
},
provide: {

```

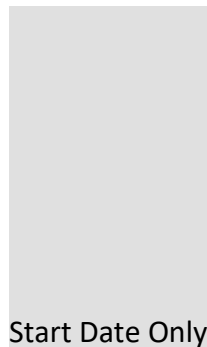
```
gantt: [Edit, Selection, Toolbar]
    }
};
</script>
```

{% previewsample "page.domainurl/code-snippet/gantt/task-scheduling-cs3" %}

#### *Unscheduled tasks*

Unscheduled tasks are planned for a project without any definite schedule dates. The Gantt control supports rendering the unscheduled tasks. You can create or update the tasks with anyone of start date, end date, and duration values or none. You can enable or disable the unscheduled tasks by using the [allowUnscheduledTasks](#) property. The following images represent the various types of unscheduled tasks in Gantt.

Taskbar state | Auto | Manual



End Date Only |



Note: A milestone is a task that has no start and end dates, but it has a duration value of zero

[Define unscheduled tasks in data source](#)

You can define the various types of unscheduled tasks in the data source as follows

#### APP.VUE

```
<template>
  <div>
    <ejs-gantt ref='gantt' id="GanttContainer" :dataSource="data"
    :taskFields = "taskFields" :height = "height" :editSettings= "editSettings"
    :allowUnscheduledTasks='true'></ejs-gantt>
  </div>
</template>
<script>
import Vue from "vue";
import { GanttPlugin, Edit } from "@syncfusion/ej2-vue-gantt";
Vue.use(GanttPlugin);
export default {
  data: function() {
    return{
      data: [
        {
          TaskID: 1,
          TaskName: 'Project Initiation',
          StartDate: new Date('04/02/2019'),
          EndDate: new Date('04/21/2019'),
          isParent:true,
          subtasks: [
            { TaskID: 2, TaskName: 'Identify Site location', Duration:
3, Progress: 50},
            { TaskID: 3, TaskName: 'Perform Soil test', StartDate: new
Date('04/02/2019'), Progress: 50 },
            { TaskID: 4, TaskName: 'Soil test approval', EndDate: new
Date('04/08/2019'), Progress: 50 },
          ]
        },
      ],
    }
  },
}
```

```

    {
      TaskID: 5,
      TaskName: 'Project Estimation',
      StartDate: new Date('04/02/2019'),
      EndDate: new Date('04/21/2019'),
      isParent:true,
      subtasks: [
        { TaskID: 6, TaskName: 'Develop floor plan for estimation',
          StartDate: new Date('04/04/2019'), Progress: 50, resources:
            [4],isParent:false },
        { TaskID: 7, TaskName: 'List materials', StartDate: new
          Date('04/04/2019'), Progress: 50 },
        { TaskID: 8, TaskName: 'Estimation approval',Duration:
          0,Progress: 50}
      ]
    },
  ],
  height: '450px',
  taskFields: {
    id: 'TaskID',
    name: 'TaskName',
    startDate: 'StartDate',
    endDate: 'EndDate',
    duration: 'Duration',
    progress: 'Progress',
    child: 'subtasks'
  },
  editSettings: {
    allowEditing:true,
    allowTaskbarEditing:true
  }
};
},
provide: {
  gantt: [ Edit ]
}
};
</script>

```

{% previewsample "page.domainurl/code-snippet/gantt/task-scheduling-cs4" %}

#### NOTE

If the [allowUnscheduledTasks](#) property is set to false, then the Gantt control automatically calculates the scheduled date values with a default value of duration 1 and the project start date is considered as the start date for the task.

#### Working time range

In the Gantt control, working hours in a day for a project can be defined by using the [dayWorkingTime](#) property. Based on the working hours, automatic date scheduling and duration validations for a task are performed.

The following code snippet explains how to define the working time range for the project in Gantt.

#### APP.VUE

```
<template>
```



```

<div>
  <ejs-gantt ref='gantt' id="GanttContainer" :dataSource="data"
:taskFields = "taskFields" :height = "height" :highlightWeekends='true'
:splitterSettings= "splitterSettings" :dayWorkingTime="dayWorkingTime"
:timelineSettings="timelineSettings"></ejs-gantt>
</div>
</template>
<script>
import Vue from "vue";
import { GanttPlugin, Edit, DayMarkers } from "@syncfusion/ej2-vue-gantt";
import { editingData } from './data-source.js';
Vue.use(GanttPlugin);
export default {
  data: function() {
    return{
      data: editingData,
      height: '450px',
      taskFields: {
        id: 'TaskID',
        name: 'TaskName',
        startDate: 'StartDate',
        duration: 'Duration',
        progress: 'Progress',
        child: 'subtasks'
      },
      splitterSettings: {
        columnIndex: 1
      },
      dayWorkingTime: [
        {from: 9,
        to: 18 }
      ],
      timelineSettings:{
        timelineViewMode:'Day'
      }
    };
  },
  provide: {
    gantt: [ DayMarkers, Edit ]
  },
};
</script>

```

{% previewsample "page.domainurl/code-snippet/gantt/task-scheduling-cs5" %}

#### NOTE

- \* Individual tasks can lie between any time within the defined working time range of the project.
- \* The [dayWorkingTime](#) property is used to define the working time for the whole project.

#### *Weekend/Non-working days*

Non-working days/weekend are used to represent the non-productive days in a project. You can define the non-working days in a week using the [workWeek](#) property in Gantt.

#### **APP.VUE**

```

<<template>
  <div>
    <ejs-gantt id="GanttContainer" :dataSource="data" :taskFields =
"taskFields" :height = "height" :highlightWeekends='true'
:workWeek="workWeek"></ejs-gantt>
  </div>
</template>
<script>
import Vue from "vue";
import { GanttPlugin, DayMarkers } from "@syncfusion/ej2-vue-gantt";
import { editingData } from './data-source.js';
Vue.use(GanttPlugin);
export default {
  data: function() {
    return{
      data: editingData,
      height: '450px',
      taskFields: {
        id: 'TaskID',
        name: 'TaskName',
        startDate: 'StartDate',
        duration: 'Duration',
        progress: 'Progress',
        child: 'subtasks'
      },
      workWeek: ["Sunday", "Monday", "Tuesday", "Wednesday", "Thursday"],
    };
  },
  provide: {
    gantt: [ DayMarkers ]
  }
};
</script>

```

{% previewsample "page.domainurl/code-snippet/gantt/task-scheduling-cs6" %}

By default, Saturdays and Sundays are considered as non-working days/weekend in a project.

In the Gantt control, you can make weekend as working day by setting the [includeWeekend](#) property to true.

#### Duration units

In Gantt, the task's duration value can be measured by the following duration units,

- Day
- Hour
- Minute

In Gantt, we can define duration unit for whole project by using [durationUnit](#) property, when we defines the value for this property, this unit will be applied for all task which don't has duration unit value. And each task in the project can be defined with different duration units and the duration unit of a task can be defined by the following ways,

- Using [taskFields.durationUnit](#) property, to map the duration unit data source field.

- Defining the duration unit value along with the duration field in the data source.

#### Mapping the duration unit field

The below code snippet explains the mapping of duration unit data source field to the Gantt control using the [taskFields.durationUnit](#) property.

#### APP.VUE

```
<template>
  <div>
    <ejs-gantt ref='gantt' id="GanttContainer" :dataSource="data"
    :taskFields = "taskFields" :height = "height" :splitterSettings =
    "splitterSettings"></ejs-gantt>
  </div>
</template>
<script>
import Vue from "vue";
import { GanttPlugin } from "@syncfusion/ej2-vue-gantt";
import { editingData } from './data-source.js';
Vue.use(GanttPlugin);
export default {
  data: function() {
    return{
      data: editingData,
      height: '450px',
      taskFields: {
        id: 'TaskID',
        name: 'TaskName',
        startDate: 'StartDate',
        duration: 'Duration',
        progress: 'Progress',
        durationUnit: 'DurationUnit',
        child: 'subtasks'
      },
      splitterSettings:{
        columnIndex:4
      }
    };
  },
};
</script>
```

{% previewsample "page.domainurl/code-snippet/gantt/scheduling-tasks-cs1" %}

#### NOTE

The default value of the [durationUnit](#) property is **day**.

#### Defining duration unit along with duration field

A duration unit for a task can be defined along with duration value, the following code snippet explains the duration unit for a task along with duration value.

#### APP.VUE

```
<template>
  <div>
```

```

    <ejs-gantt ref='gantt' id="GanttContainer" :dataSource="data"
:taskFields = "taskFields" :height = "height" :splitterSettings =
"splitterSettings"></ejs-gantt>
  </div>
</template>
<script>
import Vue from "vue";
import { GanttPlugin } from "@syncfusion/ej2-vue-gantt";
Vue.use(GanttPlugin);
export default {
  data: function() {
    return{
      data: [
        {
          TaskID: 1,
          TaskName: 'Project Initiation',
          StartDate: new Date('04/02/2019'),
          EndDate: new Date('04/21/2019'),
          isParent:true,
          subtasks: [
            { TaskID: 2, TaskName: 'Identify Site location', StartDate:
new Date('04/02/2019'), Duration: '3days', DurationUnit:'day', Progress:
50,isParent:false },
            { TaskID: 3, TaskName: 'Perform Soil test', StartDate: new
Date('04/02/2019'), Duration: '12hours',DurationUnit:'hour', Progress: 70,
resources: [2, 3, 5],isParent:false },
            { TaskID: 4, TaskName: 'Soil test approval', StartDate: new
Date('04/02/2019'), Duration: '1800minutes',DurationUnit:'minute',
Predecessor:"2FS", Progress: 80,isParent:false },
          ]
        },
        {
          TaskID: 5,
          TaskName: 'Project Estimation',
          StartDate: new Date('04/02/2019'),
          EndDate: new Date('04/21/2019'),
          isParent:true,
          subtasks: [
            { TaskID: 6, TaskName: 'Develop floor plan for estimation',
StartDate: new Date('04/04/2019'), Duration: '12hours', DurationUnit:'hour',
Progress: 50, resources: [4],isParent:false },
            { TaskID: 7, TaskName: 'List materials', StartDate: new
Date('04/04/2019'), Duration: '3days', Progress: 50, DurationUnit:'day',
resources: [4, 8],isParent:false },
            { TaskID: 8, TaskName: 'Estimation approval', StartDate: new
Date('04/04/2019'), Duration: '480minutes',Predecessor:"6SS",
DurationUnit:'minute', Progress: 70, resources: [12, 5],isParent:false }
          ]
        },
      ],
      height:'450px',
      taskFields: {
        id: 'TaskID',
        name: 'TaskName',
        startDate: 'StartDate',
        duration: 'Duration',
        progress: 'Progress',

```

```

        child: 'subtasks'
      },
      splitterSettings: {
        columnIndex: 4
      }
    };
  },
};
</script>

```

{% previewsample "page.domainurl/code-snippet/gantt/scheduling-tasks-cs2" %}

#### NOTE:

The edit type of the duration column in Gantt is string to edit the duration field along with duration units.

#### Task dependency in Vue Gantt component

Task dependency or task relationship can be established between two tasks in Gantt. This dependency affects the project schedule. If you change the predecessor of a task, it will affect the successor task, which will affect the next task, and so on. Relationship can be established between parent-parent tasks, child-child tasks, parent-child and child-parent task.

In Gantt, you can enable or disable the parent predecessor using [allowParentDependency](#) property.

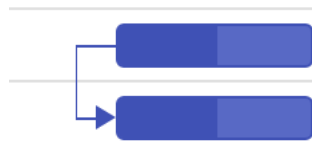
By default, the `allowParentDependency` property will be `true`.

#### Task relationship types

Task relationships are categorized into four types based on the start and finish dates of the task.

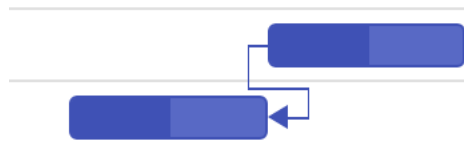
##### Start to Start(SS)

You cannot start a task until the dependent task also starts.



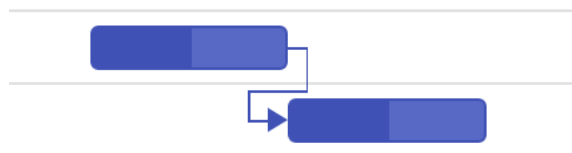
##### Start to Finish(SF)

You cannot finish a task until the dependent task is started.



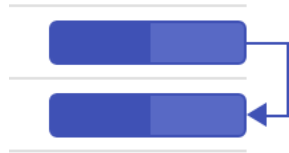
##### Finish to Start(FS)

You cannot start a task until the dependent task is completed.



### Finish to Finish(FF)

You cannot finish a task until the dependent task is completed.



### Define task relationship

Task relationship is defined in the data source as a string value, and this value is mapped to the Gantt component by using the [taskFields.dependency](#) property. The following code example demonstrates how to enable the predecessor in the Gantt component.

### APP.VUE

```

<template>
  <div>
    <ejs-gantt ref='gantt' id="GanttContainer" :dataSource="data"
    :taskFields = "taskFields" :height = "height"></ejs-gantt>
  </div>
</template>
<script>
import Vue from "vue";
import { GanttPlugin } from "@syncfusion/ej2-vue-gantt";
Vue.use(GanttPlugin);
export default {
  data: function() {
    return {
      data: [
        {
          TaskID: 1,
          TaskName: 'Project Initiation',
          StartDate: new Date('04/02/2019'),
          EndDate: new Date('04/21/2019'),
          subtasks: [
            { TaskID: 2, TaskName: 'Identify Site location', StartDate:
            new Date('04/02/2019'), Duration: 0, Progress: 50 },
            { TaskID: 3, TaskName: 'Perform Soil test', StartDate: new
            Date('04/02/2019'), Duration: 4, Progress: 50 },
            { TaskID: 4, TaskName: 'Soil test approval', StartDate: new
            Date('04/02/2019'), Duration: 4,Predecessor:"2FS", Progress: 50 },
          ]
        },
        {
          TaskID: 5,
          TaskName: 'Project Estimation',
          StartDate: new Date('04/02/2019'),
          EndDate: new Date('04/21/2019'),
          subtasks: [
            { TaskID: 6, TaskName: 'Develop floor plan for estimation',
            StartDate: new Date('04/04/2019'), Duration: 3, Progress: 50 },
            { TaskID: 7, TaskName: 'List materials', StartDate: new
            Date('04/04/2019'), Duration: 3, Progress: 50 },
            { TaskID: 8, TaskName: 'Estimation approval', StartDate: new
            Date('04/04/2019'), Duration: 0,Predecessor:"6SS", Progress: 50 }
          ]
        }
      ]
    }
  }
}

```

```

    },
    ],
    height: '450px',
    taskFields: {
      id: 'TaskID',
      name: 'TaskName',
      startDate: 'StartDate',
      endDate: 'EndDate',
      duration: 'Duration',
      progress: 'Progress',
      dependency: 'Predecessor',
      child: 'subtasks'
    },
  };
},
};
</script>

```

{% previewsample "page.domainurl/code-snippet/gantt/taskdependency-cs1" %}

#### *Predecessor offset with duration units*

In the Gantt component, the predecessor offset can be defined with the following duration units:

- Day
- Hour
- Minute

You can define an offset with various offset duration units for predecessors by using the following code example.

#### **APP.VUE**

```

<template>
  <div>
    <ejs-gantt ref='gantt' id="GanttContainer" :dataSource="data"
    :taskFields = "taskFields" :height = "height" :columns="columns"></ejs-
    gantt>
  </div>
</template>
<script>
import Vue from "vue";
import { GanttPlugin } from "@syncfusion/ej2-vue-gantt";
Vue.use(GanttPlugin);
export default {
  data: function() {
    return{
      data: [
        {
          TaskID: 1,
          TaskName: 'Project Initiation',
          StartDate: new Date('04/02/2019'),
          EndDate: new Date('04/21/2019'),
          subtasks: [
            { TaskID: 2, TaskName: 'Identify Site location', StartDate:
            new Date('04/02/2019'), Duration: 0, Progress: 50 },

```

```

        { TaskID: 3, TaskName: 'Perform Soil test', StartDate: new
Date('04/02/2019'), Duration: 4, Progress: 50 },
        { TaskID: 4, TaskName: 'Soil test approval', StartDate: new
Date('04/04/2019'), Duration: 4,Predecessor:"2FS+2days", Progress: 50 },
        { TaskID: 5, TaskName: 'Clear the building site', StartDate:
new Date('04/04/2019'), Duration: 2, Progress: 30, Predecessor: '4FF+960m'}
    ]
},
{
    TaskID: 6,
    TaskName: 'Project Estimation',
    StartDate: new Date('04/02/2019'),
    EndDate: new Date('04/21/2019'),
    subtasks: [
        { TaskID: 7, TaskName: 'Develop floor plan for estimation',
StartDate: new Date('04/04/2019'), Duration: 3, Progress: 50 },
        { TaskID: 8, TaskName: 'List materials', StartDate: new
Date('04/04/2019'), Duration: 3, Progress: 50 },
        { TaskID: 9, TaskName: 'Estimation approval', StartDate: new
Date('04/06/2019'), Duration: 0,Predecessor:"7SS+16h", Progress: 50 }
    ]
},
],
    height: '450px',
    taskFields: {
        id: 'TaskID',
        name: 'TaskName',
        startDate: 'StartDate',
        endDate: 'EndDate',
        duration: 'Duration',
        progress: 'Progress',
        dependency: 'Predecessor',
        child: 'subtasks'
    },
    columns: [
        { field: 'TaskID', headerText: 'Task ID', textAlign: 'Left',
width: '100' },
        { field: 'Predecessor', headerText: 'Depedency', width:
'150'},
        { field: 'TaskName', headerText: 'Task Name', width: '150'
},
        { field: 'StartDate', headerText: 'Start Date', width: '150'
},
        { field: 'Duration', headerText: 'Duration', width: '150' },
        { field: 'Progress', headerText: 'Progress', width: '150'}
    ]
};
},
};
</script>

```

{% previewsample "page.domainurl/code-snippet/gantt/taskdependency-cs2" %}

#### [Disabling automatic dependency offset updates](#)

By default, the dependency offsets are automatically updated in the Gantt chart whenever a task's start or end date is changed. However, if you want to disable this feature, you can do so by disabling the



[updateOffsetOnTaskbarEdit](#) property. Once this property is disabled, you can only update the offset value by editing the predecessor column cell or the offset column in the dependency tab of the edit dialog.

### APP.VUE

```
<template>
  <div>
    <ejs-gantt ref='gantt' id="GanttContainer" :dataSource="data"
    :taskFields = "taskFields" :height = "height" :toolbar="toolbar"
    :columns="columns" :updateOffsetOnTaskbarEdit ="false"></ejs-gantt>
  </div>
</template>
<script>
import Vue from "vue";
import { GanttPlugin, Toolbar } from "@syncfusion/ej2-vue-gantt";
Vue.use(GanttPlugin);
export default {
  data: function() {
    return{
      data: [
        {
          TaskID: 1,
          TaskName: 'Project Initiation',
          StartDate: new Date('04/02/2019'),
          EndDate: new Date('04/21/2019'),
          subtasks: [
            { TaskID: 2, TaskName: 'Identify Site location', StartDate:
new Date('04/02/2019'), Duration: 0, Progress: 50 },
            { TaskID: 3, TaskName: 'Perform Soil test', StartDate: new
Date('04/02/2019'), Duration: 4, Progress: 50 },
            { TaskID: 4, TaskName: 'Soil test approval', StartDate: new
Date('04/04/2019'), Duration: 4,Predecessor:"2FS+2days", Progress: 50 },
            { TaskID: 5, TaskName: 'Clear the building site', StartDate:
new Date('04/04/2019'), Duration: 2, Progress: 30, Predecessor: '4FF+960m'}
          ]
        },
        {
          TaskID: 6,
          TaskName: 'Project Estimation',
          StartDate: new Date('04/02/2019'),
          EndDate: new Date('04/21/2019'),
          subtasks: [
            { TaskID: 7, TaskName: 'Develop floor plan for estimation',
StartDate: new Date('04/04/2019'), Duration: 3, Progress: 50 },
            { TaskID: 8, TaskName: 'List materials', StartDate: new
Date('04/04/2019'), Duration: 3, Progress: 50 },
            { TaskID: 9, TaskName: 'Estimation approval', StartDate: new
Date('04/06/2019'), Duration: 0,Predecessor:"7SS+16h", Progress: 50 }
          ]
        }
      ],
      height: '450px',
      taskFields: {
        id: 'TaskID',
        name: 'TaskName',
        startDate: 'StartDate',
```

```

        endDate: 'EndDate',
        duration: 'Duration',
        progress: 'Progress',
        dependency: 'Predecessor',
        child: 'subtasks'
    },
    columns: [
        { field: 'TaskID', headerText: 'Task ID', textAlign: 'Left',
width: '100' },
        { field: 'Predecessor', headerText: 'Depedency', width:
'150'},
        { field: 'TaskName', headerText: 'Task Name', width: '150'
},
        { field: 'StartDate', headerText: 'Start Date', width: '150'
},
        { field: 'Duration', headerText: 'Duration', width: '150' },
        { field: 'Progress', headerText: 'Progress', width: '150'}
    ],
    toolbar: ['Add', 'Cancel', 'CollapseAll', 'Delete', 'Edit',
'ExpandAll', 'NextTimeSpan', 'PrevTimeSpan', 'Search', 'Update', 'Indent',
'Outdent'],
    };
    },
    provide: {
        gantt: [Toolbar ]
    }
};
</script>

```

{% previewsample "page.domainurl/code-snippet/gantt/taskdependency-cs5" %}

#### [Validate predecessor links on editing](#)

In the Gantt component, the task relationship link can be broken by editing the start date, end date, and duration value of task. When the task relationship is broken on any edit action, this can be handled in the Gantt component using the following two ways:

- `actionBegin` event.
- Predecessor validation dialog.

#### [Using `actionBegin` event](#)

When the task relationship link is broken on any edit action, then the [actionBegin](#) event will be triggered with `requestType` argument as `validateLinkedTask`. You can validate the editing action within the [actionBegin](#) event using the `validateMode` event argument. The `validateMode` event argument has the following properties:

Argument | Default value | Description

`args.validateMode.respectLink` | `false` | In this validation mode, the predecessor links will be considered as high priority. With this mode enabled, when the successor task is moved before predecessor task's end date, the editing will be reverted and dates will be validated based on the dependency links.

`args.validateMode.removeLink` | `false` | In this validation mode, the taskbar editing will be considered as high priority, where in the case of inappropriate task dates the dependency links will be removed and tasks will be moved to the edited date.

`args.validateMode.preserveLinkWithEditing | true` | In this validation mode, taskbar editing will be considered along with the dependency links. This relationship will be maintained by updating offset value of predecessors.

By default, the `preserveLinkWithEditing` validation mode will be enabled, so the predecessors are updated with offset values.

The following sample explains enabling the `respectLink` validation mode while editing the linked tasks in the `actionBegin` event.

### APP.VUE

```
<template>
  <div>
    <ejs-gantt ref='gantt' id="GanttContainer" :dataSource="data"
    :taskFields = "taskFields" :height = "height" :columns="columns"
    :actionBegin="actionBegin" :editSettings="editSettings"></ejs-gantt>
  </div>
</template>
<script>
import Vue from "vue";
import { GanttPlugin,Edit } from "@syncfusion/ej2-vue-gantt";
Vue.use(GanttPlugin);
export default {
  data: function() {
    return{
      data: [
        {
          TaskID: 1,
          TaskName: 'Project Initiation',
          StartDate: new Date('04/02/2019'),
          EndDate: new Date('04/21/2019'),
          subtasks: [
            { TaskID: 2, TaskName: 'Identify Site location', StartDate:
new Date('04/02/2019'), Duration: 0, Progress: 50 },
            { TaskID: 3, TaskName: 'Perform Soil test', StartDate: new
Date('04/02/2019'), Duration: 4, Progress: 50 },
            { TaskID: 4, TaskName: 'Soil test approval', StartDate: new
Date('04/02/2019'), Duration: 4,Predecessor:"2FS", Progress: 50 },
          ]
        },
        {
          TaskID: 5,
          TaskName: 'Project Estimation',
          StartDate: new Date('04/02/2019'),
          EndDate: new Date('04/21/2019'),
          subtasks: [
            { TaskID: 6, TaskName: 'Develop floor plan for estimation',
StartDate: new Date('04/04/2019'), Duration: 3, Progress: 50 },
            { TaskID: 7, TaskName: 'List materials', StartDate: new
Date('04/04/2019'), Duration: 3, Progress: 50 },
            { TaskID: 8, TaskName: 'Estimation approval', StartDate: new
Date('04/04/2019'), Duration: 0,Predecessor:"6SS", Progress: 50 }
          ]
        }
      ],
      height: '450px',
    }
  }
}
```

```

        taskFields: {
            id: 'TaskID',
            name: 'TaskName',
            startDate: 'StartDate',
            endDate: 'EndDate',
            duration: 'Duration',
            progress: 'Progress',
            dependency: 'Predecessor',
            child: 'subtasks'
        },
        editSettings: {
            allowTaskbarEditing: true
        },
        actionBegin: function (args) {
            if (args.requestType == "validateLinkedTask") {
                args.validateMode.respectLink = true;
            }
        },
        columns: [
            { field: 'TaskID', headerText: 'Task ID', textAlign: 'Left',
width: '100' },
            { field: 'Predecessor', headerText: 'Depedency', width:
'150'},
            { field: 'TaskName', headerText: 'Task Name', width: '150'
},
            { field: 'StartDate', headerText: 'Start Date', width: '150'
},
            { field: 'Duration', headerText: 'Duration', width: '150' },
            { field: 'Progress', headerText: 'Progress', width: '150' }
        ]
    };
    },
    provide: {
        gantt: [ Edit ]
    }
};
</script>

```

{% previewsample "page.domainurl/code-snippet/gantt/taskdependency-cs3" %}

#### Using validation dialog

When disabling all the validation modes in the [actionBegin](#) event, a validation pop-up will be displayed prompting users to select the validation mode to validate taskbar editing.

This validation pop-up will display different options based on the successor task's start date after editing.

If you move the successor task that starts after the predecessor task's end date, then a dialog will be rendered with the following options:

- Cancel, Keep the existing link.
- Remove the link and move the task to start on edited date.
- Move the task to start on edited date and keep the link.

If you move the successor task that starts before the predecessor task's end date, then a dialog will be rendered with the following options:

- Cancel, Keep the existing link.
- Remove the link and move the task to start on edited date.

The following code example shows how to enable the predecessor validation dialog in Gantt.

#### APP.VUE

```
<template>
  <div>
    <ejs-gantt ref='gantt' id="GanttContainer" :dataSource="data"
:taskFields = "taskFields" :height = "height" :columns="columns"
:actionBegin="actionBegin" :editSettings="editSettings"></ejs-gantt>
  </div>
</template>
<script>
import Vue from "vue";
import { GanttPlugin,Edit } from "@syncfusion/ej2-vue-gantt";
Vue.use(GanttPlugin);
export default {
  data: function() {
    return{
      data: [
        {
          TaskID: 1,
          TaskName: 'Project Initiation',
          StartDate: new Date('04/02/2019'),
          EndDate: new Date('04/21/2019'),
          subtasks: [
            { TaskID: 2, TaskName: 'Identify Site location', StartDate:
new Date('04/02/2019'), Duration: 0, Progress: 50 },
            { TaskID: 3, TaskName: 'Perform Soil test', StartDate: new
Date('04/02/2019'), Duration: 4, Progress: 50 },
            { TaskID: 4, TaskName: 'Soil test approval', StartDate: new
Date('04/02/2019'), Duration: 4,Predecessor:"2FS", Progress: 50 },
          ]
        },
        {
          TaskID: 5,
          TaskName: 'Project Estimation',
          StartDate: new Date('04/02/2019'),
          EndDate: new Date('04/21/2019'),
          subtasks: [
            { TaskID: 6, TaskName: 'Develop floor plan for estimation',
StartDate: new Date('04/04/2019'), Duration: 3, Progress: 50 },
            { TaskID: 7, TaskName: 'List materials', StartDate: new
Date('04/04/2019'), Duration: 3, Progress: 50 },
            { TaskID: 8, TaskName: 'Estimation approval', StartDate: new
Date('04/04/2019'), Duration: 0,Predecessor:"6SS", Progress: 50 }
          ]
        },
      ],
      height: '450px',
      taskFields: {
```

```

        id: 'TaskID',
        name: 'TaskName',
        startDate: 'StartDate',
        endDate: 'EndDate',
        duration: 'Duration',
        progress: 'Progress',
        dependency: 'Predecessor',
        child: 'subtasks'
    },
    editSettings: {
        allowTaskbarEditing: true
    },
    actionBegin: function (args) {
        if (args.requestType == "validateLinkedTask") {
            args.validateMode.preserveLinkWithEditing = false;
        }
    },
    columns: [
        { field: 'TaskID', headerText: 'Task ID', textAlign: 'Left',
width: '100' },
        { field: 'Predecessor', headerText: 'Depedency', width:
'150'},
        { field: 'TaskName', headerText: 'Task Name', width: '150'
},
        { field: 'StartDate', headerText: 'Start Date', width: '150'
},
        { field: 'Duration', headerText: 'Duration', width: '150' },
        { field: 'Progress', headerText: 'Progress', width: '150' }
    ],
    ];
    },
    provide: {
        gantt: [ Edit ]
    }
};
</script>

```

{% previewsample "page.domainurl/code-snippet/gantt/taskdependency-cs4" %}

#### [Dynamically show/hide the dependency line](#)

By default, mapping the dependency field in taskFields displays dependency lines in the Gantt chart. To hide the dependency line upon button click, set **visibility** style to hidden for the CSS class name **.e-gantt-dependency-view-container**.

#### **APP.VUE**

```

<template>
  <div>
    <div class="switch-container">
      <label for="switch">Show/Hide DependencyLine</label>
      <ejs-switch id="switch" ref="toggleSwitch" checked=false
:change="change"></ejs-switch>
    </div>
    <ejs-gantt ref='gantt' :dataSource="data" id="GanttContainer"
:taskFields = "taskFields" :height = "height"></ejs-gantt>
  </div>

```

```

</template>
<script>
import Vue from "vue";
import { GanttPlugin } from "@syncfusion/ej2-vue-gantt";
import { SwitchPlugin } from "@syncfusion/ej2-vue-buttons";
Vue.use(GanttPlugin);
Vue.use(SwitchPlugin);
export default {
  data: function() {
    return {
      data: [
        {
          TaskID: 1,
          TaskName: 'Project Initiation',
          StartDate: new Date('04/02/2019'),
          EndDate: new Date('04/21/2019'),
          subtasks: [
            { TaskID: 2, TaskName: 'Identify Site location',
StartDate: new Date('04/02/2019'), Duration: 4, Progress: 50 },
            { TaskID: 3, TaskName: 'Perform Soil test', StartDate:
new Date('04/02/2019'), Duration: 4, Progress: 50 , Predecessor: '4FS' },
            { TaskID: 4, TaskName: 'Soil test approval', StartDate:
new Date('04/02/2019'), Duration: 4, Progress: 50 },
          ]
        },
        {
          TaskID: 5,
          TaskName: 'Project Estimation',
          StartDate: new Date('04/02/2019'),
          EndDate: new Date('04/21/2019'),
          subtasks: [
            { TaskID: 6, TaskName: 'Develop floor plan for
estimation', StartDate: new Date('04/04/2019'), Duration: 3, Progress: 50 },
            { TaskID: 7, TaskName: 'List materials', StartDate: new
Date('04/04/2019'), Duration: 3, Progress: 50, Predecessor: '6FS' },
            { TaskID: 8, TaskName: 'Estimation approval', StartDate:
new Date('04/04/2019'), Duration: 3, Progress: 50, Predecessor: '7SS' }
          ],
        },
      ],
      height: '450px',
      taskFields: {
        id: 'TaskID',
        name: 'TaskName',
        startDate: 'StartDate',
        endDate: 'EndDate',
        duration: 'Duration',
        progress: 'Progress',
        child: 'subtasks',
        dependency: 'Predecessor',
      },
    };
  },
  methods: {
    change: function(args) {
      var ganttDependencyViewContainer = document.querySelector('.e-gantt-
dependency-view-container');
    }
  }
};

```

```

        if (args.checked) {
            ganttDependencyViewContainer.style.visibility = 'hidden';
        }
        else {
            ganttDependencyViewContainer.style.visibility = 'visible';
        }
    }
}
};
</script>
<style>
.switch-container {
    display: flex;
    align-items: center;
    padding: 10px 0px 10px 0px;
}
.switch {
    margin-left: 10px;
}
</style>

```

{% previewsample "page.domainurl/code-snippet/gantt/taskdependency-cs6" %}

## Taskbar

### Taskbar in Vue Gantt component

#### Taskbar template

You can design your own taskbars to view the tasks in Gantt using the [taskbarTemplate](#) property. You can customize the parent taskbars and milestones with custom templates using the [parentTaskbarTemplate](#) and [milestoneTemplate](#) properties.

### APP.VUE

```

<template>
  <div>
    <ejs-gantt ref='gantt' id="GanttContainer" :dataSource="data"
    :taskFields="taskFields" :height="height"
    :rowHeight="rowHeight" :taskbarTemplate="'taskbarTemplate'"
    :parentTaskbarTemplate="'parentTaskbarTemplate'"
    :milestoneTemplate="'milestoneTemplate'">
      <template v-slot:taskbarTemplate="{data}">
        <div class="e-gantt-child-taskbar e-custom-moments"
        style="height:100%;border-radius:5px;">
          <span class="e-task-label"
            style="position:absolute;top:5px;font-size:12px;text-
            overflow:ellipsis;height:90%;overflow:hidden;">{{data.TaskName}}</span>
        </div>
      </template>
      <template v-slot:parentTaskbarTemplate="{data}">
        <div class="e-gantt-child-taskbar e-custom-parent"
        style="height:100%;border-radius:5px;">
          <span class="e-task-label"
            style="position:absolute;top:5px;font-size:12px;text-
            overflow:ellipsis;height:90%;overflow:hidden;">{{data.TaskName}}</span>
        </div>
      </template>
    </div>
  </div>

```



```

    <template v-slot:milestoneTemplate="{data}">
      <div class="e-gantt-milestone" style="position:absolute;">
        <div class="e-milestone-top"
          style="border-right-width:26px; margin-top: -9px;border-left-
width:26px;border-bottom-width:26px;"></div>
        <div class="e-milestone-bottom"
          style="top:26px;border-right-width:26px; border-left-width:26px;
border-top-width:26px;"></div>
      </div>
    </template>
  </ejs-gantt>
</div>
</template>
<script>
import Vue from "vue";
import { GanttPlugin } from "@syncfusion/ej2-vue-gantt";
Vue.use(GanttPlugin);
export default {
  data: function() {
    return{
      data: [
        {
          TaskID: 1,
          TaskName: 'Project Initiation',
          StartDate: new Date('04/02/2019'),
          EndDate: new Date('04/21/2019'),
          isParent:true,
          subtasks: [
            { TaskID: 2, TaskName: 'Identify Site location', StartDate:
new Date('04/02/2019'), Duration: 0, Progress: 50,isParent:false },
            { TaskID: 3, TaskName: 'Perform Soil test', StartDate: new
Date('04/02/2019'), Duration: 4, Progress: 50, resources: [2, 3,
5],isParent:false },
            { TaskID: 4, TaskName: 'Soil test approval', StartDate: new
Date('04/02/2019'), Duration: 4,Predecessor:"2FS", Progress:
50,isParent:false },
          ]
        },
        {
          TaskID: 5,
          TaskName: 'Project Estimation',
          StartDate: new Date('04/02/2019'),
          EndDate: new Date('04/21/2019'),
          isParent:true,
          subtasks: [
            { TaskID: 6, TaskName: 'Develop floor plan for estimation',
StartDate: new Date('04/04/2019'), Duration: 3, Progress: 50, resources:
[4],isParent:false },
            { TaskID: 7, TaskName: 'List materials', StartDate: new
Date('04/04/2019'), Duration: 3, Progress: 50, resources: [4,
8],isParent:false },
            { TaskID: 8, TaskName: 'Estimation approval', StartDate: new
Date('04/04/2019'), Duration: 0,Predecessor:"6SS", Progress: 50, resources:
[12, 5],isParent:false }
          ]
        },
      ],
    }
  }
}

```

```

        height: '450px',
        taskFields: {
            id: 'TaskID',
            name: 'TaskName',
            startDate: 'StartDate',
            duration: 'Duration',
            progress: 'Progress',
            child: 'subtasks'
        },
        rowHeight: 75,
    };
},
];
</script>
<style>
.e-custom-parent {
    background-color: #6d619b;
    border: 1px solid #3f51b5;
}
.e-custom-moments {
    background-color: #7ab748;
    border: 1px solid #3f51b5;
}
.e-custom-performance {
    background-color: #ad7a66;
    border: 1px solid #3f51b5;
}
#taskbarTemplate .e-milestone-top {
    border-bottom-color: #7ab748 !important;
    border-bottom: 1px solid #3f51b5;
}
#taskbarTemplate .e-milestone-bottom {
    border-top-color: #7ab748 !important;
    border-top: 1px solid #3f51b5;
}
</style>

```

{% previewsample "page.domainurl/code-snippet/gantt/appearance-customization-cs2" %}

### Taskbar customization

#### Taskbar height

The height of child taskbars and parent taskbars can be customized by using [taskbarHeight](#) property. The following code example shows how to use the [taskbarHeight](#) property.

#### APP.VUE

```

<template>
  <div>
    <ejs-gantt ref='gantt' id="GanttContainer" :dataSource="data"
      :taskFields = "taskFields" :height = "height" :taskbarHeight="taskbarHeight"
      :rowHeight="rowHeight"></ejs-gantt>
  </div>
</template>
<script>
import Vue from "vue";
import { GanttPlugin } from "@syncfusion/ej2-vue-gantt";

```

```

Vue.use(GanttPlugin);
export default {
  data: function() {
    return{
      data: [
        {
          TaskID: 1,
          TaskName: 'Project Initiation',
          StartDate: new Date('04/02/2019'),
          EndDate: new Date('04/21/2019'),
          isParent:true,
          subtasks: [
            { TaskID: 2, TaskName: 'Identify Site location', StartDate:
new Date('04/02/2019'), Duration: 0, Progress: 50,isParent:false },
            { TaskID: 3, TaskName: 'Perform Soil test', StartDate: new
Date('04/02/2019'), Duration: 4, Progress: 50, resources: [2, 3,
5],isParent:false },
            { TaskID: 4, TaskName: 'Soil test approval', StartDate: new
Date('04/02/2019'), Duration: 4,Predecessor:"2FS", Progress:
50,isParent:false },
          ]
        },
        {
          TaskID: 5,
          TaskName: 'Project Estimation',
          StartDate: new Date('04/02/2019'),
          EndDate: new Date('04/21/2019'),
          isParent:true,
          subtasks: [
            { TaskID: 6, TaskName: 'Develop floor plan for estimation',
StartDate: new Date('04/04/2019'), Duration: 3, Progress: 50, resources:
[4],isParent:false },
            { TaskID: 7, TaskName: 'List materials', StartDate: new
Date('04/04/2019'), Duration: 3, Progress: 50, resources: [4,
8],isParent:false },
            { TaskID: 8, TaskName: 'Estimation approval', StartDate: new
Date('04/04/2019'), Duration: 0,Predecessor:"6SS", Progress: 50, resources:
[12, 5],isParent:false }
          ]
        },
      ],
      height: '450px',
      taskFields: {
        id: 'TaskID',
        name: 'TaskName',
        startDate: 'StartDate',
        duration: 'Duration',
        progress: 'Progress',
        child: 'subtasks'
      },
      taskbarHeight:60,
      rowHeight: 60
    };
  },
};
</script>

```

```
{% previewsample "page.domainurl/code-snippet/gantt/appearance-customization-cs1" %}
```

#### NOTE

The [taskbarHeight](#) value should be lower than [rowHeight](#) property value and these properties accept only pixel values.

#### Conditional formatting

The default taskbar UI can be replaced with custom templates by using the [queryTaskbarInfo](#) event. The following code example shows customizing the taskbar UI based on task progress values in Gantt component.

#### APP.VUE

```
<template>
  <div>
    <ejs-gantt ref='gantt' id="GanttContainer" :dataSource="data"
    :taskFields = "taskFields" :height = "height" :queryTaskbarInfo =
    "queryTaskbarInfo"></ejs-gantt>
  </div>
</template>
<script>
import Vue from "vue";
import { GanttPlugin } from "@syncfusion/ej2-vue-gantt";
Vue.use(GanttPlugin);
export default {
  data: function() {
    return {
      data: [
        {
          TaskID: 1,
          TaskName: 'Project Initiation',
          StartDate: new Date('04/02/2019'),
          EndDate: new Date('04/21/2019'),
          isParent:true,
          subtasks: [
            { TaskID: 2, TaskName: 'Identify Site location', StartDate:
            new Date('04/02/2019'), Duration: 0, Progress: 50,isParent:false },
            { TaskID: 3, TaskName: 'Perform Soil test', StartDate: new
            Date('04/02/2019'), Duration: 4, Progress: 70, resources: [2, 3,
            5],isParent:false },
            { TaskID: 4, TaskName: 'Soil test approval', StartDate: new
            Date('04/02/2019'), Duration: 4,Predecessor:"2FS", Progress:
            50,isParent:false },
          ]
        },
        {
          TaskID: 5,
          TaskName: 'Project Estimation',
          StartDate: new Date('04/02/2019'),
          EndDate: new Date('04/21/2019'),
          isParent:true,
          subtasks: [
```

```

        { TaskID: 6, TaskName: 'Develop floor plan for estimation',
        StartDate: new Date('04/04/2019'), Duration: 3, Progress: 50, resources:
        [4],isParent:false },
        { TaskID: 7, TaskName: 'List materials', StartDate: new
        Date('04/04/2019'), Duration: 3, Progress: 80, resources: [4,
        8],isParent:false },
        { TaskID: 8, TaskName: 'Estimation approval', StartDate: new
        Date('04/04/2019'), Duration: 0,Predecessor:"6SS", Progress: 70, resources:
        [12, 5],isParent:false }
    ],
    },
    ],
    height: '450px',
    taskFields: {
    id: 'TaskID',
    name: 'TaskName',
    startDate: 'StartDate',
    duration: 'Duration',
    progress: 'Progress',
    child: 'subtasks'
    },
    queryTaskbarInfo: function(args) {
        if (args.data.Progress == 50) {
            args.progressBarBgColor = "red";
        } else if (args.data.Progress == 70) {
            args.progressBarBgColor = "yellow";
        } else if (args.data.Progress == 80) {
            args.progressBarBgColor = "lightgreen";
        }
    }
    };
},
};
</script>

```

{% previewsample "page.domainurl/code-snippet/gantt/appearance-customization-cs3" %}

### Change gripper icon in taskbar

You can change the gripper icon in the taskbar by applying styles to their respective class elements.

### APP.VUE

```

<template>
<div>
<ejs-gantt id="gantt" :dataSource="data" :taskFields = "taskFields"
:editSettings = "editSettings" :height="height"></ejs-gantt>
</div>
</template>
<script>
import Vue from 'vue';
import { GanttPlugin, Edit } from "@syncfusion/ej2-vue-gantt";
Vue.use(GanttPlugin);
export default {
    data: function() {
        return{
            data: [

```

```

        {
            TaskID: 1,
            TaskName: 'Project Initiation',
            StartDate: new Date('04/02/2019'),
            EndDate: new Date('04/21/2019'),
            subtasks: [
                { TaskID: 2, TaskName: 'Identify Site location',
                StartDate: new Date('04/02/2019'), Duration: 4, Progress: 50 },
                { TaskID: 3, TaskName: 'Perform Soil test', StartDate:
                new Date('04/02/2019'), Duration: 4, Progress: 50 },
                { TaskID: 4, TaskName: 'Soil test approval', StartDate:
                new Date('04/02/2019'), Duration: 4, Progress: 50 },
            ]
        },
        {
            TaskID: 5,
            TaskName: 'Project Estimation',
            StartDate: new Date('04/02/2019'),
            EndDate: new Date('04/21/2019'),
            subtasks: [
                { TaskID: 6, TaskName: 'Develop floor plan for
                estimation', StartDate: new Date('04/04/2019'), Duration: 3, Progress: 50 },
                { TaskID: 7, TaskName: 'List materials', StartDate: new
                Date('04/04/2019'), Duration: 3, Progress: 50 },
                { TaskID: 8, TaskName: 'Estimation approval', StartDate:
                new Date('04/04/2019'), Duration: 3, Progress: 50 }
            ]
        },
    ],
    height: '450px',
    taskFields: {
        id: 'TaskID',
        name: 'TaskName',
        startDate: 'StartDate',
        endDate: 'EndDate',
        duration: 'Duration',
        progress: 'Progress',
        dependency: 'Predecessor',
        child: 'subtasks'
    },
    editSettings: {
        allowEditing: true,
        editMode: 'Auto',
        allowTaskbarEditing: true
    },
};

},
provide: {
    gantt: [ Edit ]
}
};
</script>
<style>
/* change gripper icon */
.e-gantt .e-left-resize-gripper::before, .e-gantt .e-right-resize-
gripper::before {
    content: '\e934';

```

```

}
.e-gantt .e-left-resize-gripper, .e-gantt .e-right-resize-gripper {
  transform: rotate(90deg);
}
</style>

```

{% previewsample "page.domainurl/code-snippet/gantt/change-gripper-icon-cs1" %}

### Connector lines

The width and background color of connector lines in Gantt can be customized using the [connectorLineWidth](#) and [connectorLineBackground](#) properties. The following code example shows how to use these properties.

### APP.VUE

```

<template>
  <div>
    <ejs-gantt ref='gantt' id="GanttContainer" :dataSource="data"
      :taskFields = "taskFields" :height = "height"
      :connectorLineBackground="connectorLineBackground"
      :connectorLineWidth="connectorLineWidth"></ejs-gantt>
  </div>
</template>
<script>
import Vue from "vue";
import { GanttPlugin } from "@syncfusion/ej2-vue-gantt";
Vue.use(GanttPlugin);
export default {
  data: function() {
    return{
      data: [
        {
          TaskID: 1,
          TaskName: 'Project Initiation',
          StartDate: new Date('04/02/2019'),
          EndDate: new Date('04/21/2019'),
          subtasks: [
            { TaskID: 2, TaskName: 'Identify Site location', StartDate:
new Date('04/02/2019'), Duration: 0, Progress: 50 },
            { TaskID: 3, TaskName: 'Perform Soil test', StartDate: new
Date('04/02/2019'), Duration: 4, Progress: 50 },
            { TaskID: 4, TaskName: 'Soil test approval', StartDate: new
Date('04/02/2019'), Duration: 4,Predecessor:"2FS", Progress: 50 },
          ]
        },
        {
          TaskID: 5,
          TaskName: 'Project Estimation',
          StartDate: new Date('04/02/2019'),
          EndDate: new Date('04/21/2019'),
          subtasks: [
            { TaskID: 6, TaskName: 'Develop floor plan for estimation',
StartDate: new Date('04/04/2019'), Duration: 3, Progress: 50 },
            { TaskID: 7, TaskName: 'List materials', StartDate: new
Date('04/04/2019'), Duration: 3, Progress: 50 },
          ]
        }
      ]
    }
  }
}

```

```

        { TaskID: 8, TaskName: 'Estimation approval', StartDate: new
Date('04/04/2019'), Duration: 0,Predecessor:"6SS", Progress: 50 }
    ]
  },
],
    height: '450px',
    taskFields: {
      id: 'TaskID',
      name: 'TaskName',
      startDate: 'StartDate',
      duration: 'Duration',
      dependency: 'Predecessor',
      progress: 'Progress',
      child: 'subtasks'
    },
    connectorLineBackground:"red",
    connectorLineWidth:3
  };
},
};
</script>

```

{% previewsample "page.domainurl/code-snippet/gantt/appearance-customization-cs5" %}

### Tooltip

In the Gantt component, you can enable or disable the mouse hover tooltip for the following UI elements using the [tooltipSettings.showTooltip](#) property:

- Taskbar
- Connector line
- Baseline
- Event marker

### APP.VUE

```

<template>
  <div>
    <ejs-gantt ref='gantt' :dataSource="data" id="GanttContainer"
:taskFields = "taskFields" :height = "height" :eventMarkers="eventMarkers"
:renderBaseline="renderBaseline" :treeColumnIndex='1'
:baselineColor="baselineColor" :tooltipSettings="tooltipSettings"></ejs-
gantt>
  </div>
</template>
<script>
import Vue from "vue";
import { GanttPlugin, Selection, DayMarkers } from "@syncfusion/ej2-vue-
gantt";
import { baselineData } from './data-source.js';
Vue.use(GanttPlugin);
export default {
  data: function() {
    return{
      data: [
        {

```



```

    TaskID: 1,
    TaskName: 'Project Initiation',
    StartDate: new Date('04/02/2019'),
    EndDate: new Date('04/21/2019'),
    subtasks: [
      { TaskID: 2, TaskName: 'Identify Site
location',BaselineStartDate: new Date('04/02/2019'),BaselineEndDate: new
Date('04/02/2019'), StartDate: new Date('04/02/2019'), Duration: 0,
Progress: 50 },
      { TaskID: 3, TaskName: 'Perform Soil test', StartDate: new
Date('04/02/2019'),BaselineStartDate: new
Date('04/04/2019'),BaselineEndDate: new Date('04/09/2019'), Duration: 4,
Progress: 50 },
      { TaskID: 4, TaskName: 'Soil test approval', StartDate: new
Date('04/02/2019'),BaselineStartDate: new
Date('04/08/2019'),BaselineEndDate: new Date('04/12/2019'), Duration:
4,Predecessor:"2FS", Progress: 50 },
    ]
  },
  {
    TaskID: 5,
    TaskName: 'Project Estimation',
    StartDate: new Date('04/02/2019'),
    EndDate: new Date('04/21/2019'),
    subtasks: [
      { TaskID: 6, TaskName: 'Develop floor plan for
estimation',BaselineStartDate: new Date('04/04/2019'),BaselineEndDate: new
Date('04/08/2019'), StartDate: new Date('04/04/2019'), Duration: 3,
Progress: 50 },
      { TaskID: 7, TaskName: 'List materials', StartDate: new
Date('04/04/2019'),BaselineStartDate: new
Date('04/02/2019'),BaselineEndDate: new Date('04/04/2019'), Duration: 3,
Progress: 50 },
      { TaskID: 8, TaskName: 'Estimation approval', StartDate: new
Date('04/02/2019'),BaselineStartDate: new
Date('04/02/2019'),BaselineEndDate: new Date('04/08/2019'), Duration:
0,Predecessor:"6SS", Progress: 50 }
    ]
  },
],
  taskFields: {
    id: 'TaskID',
    name: 'TaskName',
    startDate: 'StartDate',
    duration: 'Duration',
    baselineStartDate: "BaselineStartDate",
    baselineEndDate: "BaselineEndDate",
    progress: 'Progress',
    dependency: 'Predecessor',
    child: 'subtasks'
  },
  eventMarkers: [
    {
      day: '04/10/2019',
      label: 'Project approval and kick-off'
    }
  ],

```

```

        renderBaseline:true,
        baselineColor:'red',
        tooltipSettings:{
            showTooltip:true
        }
        height: '450px',
    };
},
provide: {
    gantt: [ Selection, DayMarkers ]
}
};
</script>

```

{% previewsample "page.domainurl/code-snippet/gantt/tooltip-cs1" %}

The default value of the [tooltipSettings.showTooltip](#) property is `true`.

#### Tooltip template

#### Taskbar tooltip

The default tooltip in the Gantt component can be customized using the [tooltipSettings.taskbar](#) property.

### APP.VUE

```

<template>
  <div>
    <ejs-gantt ref='gantt' :dataSource="data" id="GanttContainer"
    :taskFields="taskFields" :height="height"
    :tooltipSettings="tooltipSettings">
      <template v-slot:taskbarTooltipTemplate="{data}">
        <div>TaskID: {{data.TaskID}}</div>
      </template>
    </ejs-gantt>
  </div>
</template>
<script>
import Vue from "vue";
import { GanttPlugin } from "@syncfusion/ej2-vue-gantt";
import { editingData } from './data-source.js';
import taskbarTemplate from './taskbartemplate.vue';
Vue.use(GanttPlugin);
export default {
  data: function() {
    return{
      data: editingData,
      height: '450px',
      taskFields: {
        id: 'TaskID',
        name: 'TaskName',
        startDate: 'StartDate',
        duration: 'Duration',
        baselineStartDate:"BaselineStartDate",
        baselineEndDate:"BaselineEndDate",
        progress: 'Progress',
        dependency: 'Predecessor',

```

```

        child: 'subtasks'
      },
      tooltipSettings: {
        showTooltip: true,
        taskbar: "taskbarTooltipTemplate"
      },
    };
  },
};
</script>

```

{% previewsample "page.domainurl/code-snippet/gantt/tooltip-cs4" %}

#### Connector line tooltip

The default connector line tooltip in the Gantt component can be customized using the [tooltipSettings.connectorLine](#) property. The following code example shows how to use the [tooltipSettings.connectorLine](#) property.

#### APP.VUE

```

<template>
  <div>
    <ejs-gantt ref='gantt' :dataSource="data" id="GanttContainer"
      :taskFields="taskFields" :height="height"
      :tooltipSettings="tooltipSettings">
      <template v-slot:connectorLineTooltipTemplate="{data}">
        <div>Offset : {{data.offsetString}}</div>
      </template>
    </ejs-gantt>
  </div>
</template>
<script>
import Vue from "vue";
import { GanttPlugin } from "@syncfusion/ej2-vue-gantt";
import { editingData } from './data-source.js';
Vue.use(GanttPlugin);
export default {
  data: function() {
    return{
      data: editingData,
      height: '450px',
      taskFields: {
        id: 'TaskID',
        name: 'TaskName',
        startDate: 'StartDate',
        duration: 'Duration',
        progress: 'Progress',
        dependency: 'Predecessor',
        child: 'subtasks'
      },
      tooltipSettings: {
        showTooltip: true,
        connectorLine: "connectorLineTooltipTemplate"
      }
    };
  },
},

```

```
};
</script>
```

{% previewsample "page.domainurl/code-snippet/gantt/tooltip-cs5" %}

### Baseline tooltip

A baseline tooltip can be customized using the [tooltipSettings.baseline](#) property. The following code example shows how to customize the baseline tooltip in Gantt.

### APP.VUE

```
<template>
  <div>
    <ejs-gantt ref='gantt' id="GanttContainer" :dataSource="data"
    :renderBaseline="true" :taskFields="taskFields"
      :columns="columns" :treeColumnIndex="1" :includeWeekend="true"
    :timelineSettings="timelineSettings"
      :height="height" :dayWorkingTime="dayWorkingTime"
    :projectStartDate="projectStartDate"
      :projectEndDate="projectEndDate" :tooltipSettings="tooltipSettings"
    baselineColor='red'>
      <template v-slot:baselineTooltipTemplate="{data}">
        <div>Baseline StartDate : {{format(data.BaselineStartDate)}}</div>
      </template>
    </ejs-gantt>
  </div>
</template>
<script>
import Vue from "vue";
import { GanttPlugin, DayMarkers } from "@syncfusion/ej2-vue-gantt";
import { baselineData } from './data-source.js';
import { Internationalization } from '@syncfusion/ej2-base';
let instance = new Internationalization();
Vue.use(GanttPlugin);
export default {
  data: function() {
    return{
      data: baselineData,
      taskFields: {
        id: 'TaskId',
        name: 'TaskName',
        startDate: 'StartDate',
        endDate: 'EndDate',
        baselineStartDate: 'BaselineStartDate',
        baselineEndDate: 'BaselineEndDate'
      },
      columns: [
        { field: 'TaskName', headerText: 'Service Name', width:
'250', clipMode: 'EllipsisWithTooltip' },
        { field: 'BaselineStartDate', headerText: 'Planned start
time' },
        { field: 'BaselineEndDate', headerText: 'Planned end time'
},
        { field: 'StartDate', headerText: 'Start time' },
        { field: 'EndDate', headerText: 'End time' },
      ]
    }
  }
}
```

```

        timelineSettings: {
            timelineUnitSize: 65,
            topTier: {
                unit: 'None',
            },
            bottomTier: {
                unit: 'Minutes',
                count: 15,
                format: 'hh:mm a'
            },
        },
        dateFormat: 'hh:mm a',
        height: '450px',
        dayWorkingTime: [{ from: 1, to: 24 }],
        projectStartDate: new Date('03/05/2018 09:30:00 AM'),
        projectEndDate: new Date('03/05/2018 07:00:00 PM'),
        tooltipSettings: {
            showTooltip: true,
            baseline: "baselineTooltipTemplate"
        },
    };
},
provide: {
    gantt: [ DayMarkers ]
},
methods: {
    format: function(value) {
        return instance.formatDate(value, { skeleton: 'yMd', type:
'date' });
    }
};
</script>

```

{% previewsample "page.domainurl/code-snippet/gantt/tooltip-cs7" %}

### Baseline in Vue Gantt component

The baseline feature enables users to view the deviation between the planned dates and actual dates of the tasks in a project. Baseline dates or planned dates of a task may or may not be same as the actual task dates. The baseline can be enabled by setting the [renderBaseline](#) property to `true` and the baseline color can be changed using the [baselineColor](#) property. To render the baseline, you should map the baseline start and end date values from the data source. This can be done using the [baselineStartDate](#) and [baselineEndDate](#) properties. The following code example shows how to enable a baseline in the Gantt component.

### APP.VUE

```

<template>
  <div>
    <ejs-gantt ref='gantt' id="GanttContainer" :dataSource="data"
:renderBaseline="true" baselineColor='red' :taskFields="taskFields"
:columns="columns" :treeColumnIndex="1" :allowSelection="true"
:includeWeekend="true" :timelineSettings="timelineSettings" :height="height"
:dayWorkingTime="dayWorkingTime" :projectStartDate="projectStartDate"
:projectEndDate="projectEndDate"></ejs-gantt>
  </div>
</template>

```

```

    </div>
</template>
<script>
import Vue from "vue";
import { GanttPlugin, Selection } from "@syncfusion/ej2-vue-gantt";
import { baselineData } from './data-source.js';
Vue.use(GanttPlugin);
export default {
  data: function() {
    return{
      data: baselineData,
      taskFields: {
        id: 'TaskId',
        name: 'TaskName',
        startDate: 'StartDate',
        endDate: 'EndDate',
        baselineStartDate: 'BaselineStartDate',
        baselineEndDate: 'BaselineEndDate'
      },
      columns: [
        { field: 'TaskName', headerText: 'Service Name', width:
'250', clipMode: 'EllipsisWithTooltip' },
        { field: 'BaselineStartDate', headerText: 'Planned start
time' },
        { field: 'BaselineEndDate', headerText: 'Planned end time'
},
        { field: 'StartDate', headerText: 'Start time' },
        { field: 'EndDate', headerText: 'End time' },
      ],
      timelineSettings: {
        timelineUnitSize: 65,
        topTier: {
          unit: 'None',
        },
        bottomTier: {
          unit: 'Minutes',
          count: 15,
          format: 'hh:mm a'
        },
      },
      dateFormat: 'hh:mm a',
      height: '450px',
      dayWorkingTime: [{ from: 1, to: 24 }],
      projectStartDate: new Date('03/05/2018 09:30:00 AM'),
      projectEndDate: new Date('03/05/2018 07:00:00 PM'),
    };
  },
  provide: {
    gantt: [ Selection ]
  }
};
</script>

```

{% previewsample "page.domainurl/code-snippet/gantt/baseline-cs1" %}

## Task labels in Vue Gantt component

### Task labels

The Gantt component maps any data source fields to task labels using the [labelSettings.leftLabel](#), [labelSettings.rightLabel](#), and [labelSettings.taskLabel](#) properties. You can customize the task labels with templates.

### APP.VUE

```
<template>
  <div>
    <ejs-gantt ref='gantt' id="GanttContainer" :dataSource="data"
:height = "height" :taskFields = "taskFields" :labelSettings="labelSettings"
:projectStartDate="projectStartDate" :projectEndDate="projectEndDate"></ejs-
gantt>
  </div>
</template>
<script>
import Vue from "vue";
import { GanttPlugin } from "@syncfusion/ej2-vue-gantt";
import { editingData } from './data-source.js';
Vue.use(GanttPlugin);
export default {
  data: function() {
    return{
      data: editingData,
      height: '450px',
      taskFields: {
        id: 'TaskID',
        name: 'TaskName',
        startDate: 'StartDate',
        duration: 'Duration',
        progress: 'Progress',
        child: 'subtasks'
      },
      labelSettings: {
        leftLabel: 'Task ID: ${taskData.TaskID}',
        rightLabel: 'Progress Value: ${taskData.Progress}',
        taskLabel: '${Progress}%'
      },
      projectStartDate: new Date('03/28/2019'),
      projectEndDate: new Date('04/14/2019'),
    };
  },
};
</script>
```

{% previewsample "page.domainurl/code-snippet/gantt/appearance-customization-cs4" %}

## Data markers in Vue Gantt component

Data markers are a set of events used to represent the schedule events for a task. Data markers are defined in data source as array of objects, and this value is mapped to the Gantt control using the [taskFields.indicators](#) property. You can represent more than one data marker in a task.

Data markers can be defined using the following properties:

- [date](#): Defines the date of indicator.
- [iconClass](#): Defines the icon class of indicator.
- [name](#): Defines the name of indicator.
- [tooltip](#): Defines the tooltip of indicator.

Note: Data Marker **tooltip** will be rendered only if tooltip property has value.

The following code example demonstrates how to implement data markers in the Gantt chart.

#### APP.VUE

```
<template>
  <div>
    <ejs-gantt ref='gantt' id="GanttContainer" :dataSource="data"
    :taskFields = "taskFields" :height = "height"></ejs-gantt>
  </div>
</template>
<script>
import Vue from "vue";
import { GanttPlugin } from "@syncfusion/ej2-vue-gantt";
Vue.use(GanttPlugin);
export default {
  data: function() {
    return {
      data: [
        {
          TaskID: 1,
          TaskName: 'Project Initiation',
          StartDate: new Date('04/02/2019'),
          EndDate: new Date('04/21/2019'),
          subtasks: [
            {
              TaskID: 2, TaskName: 'Identify Site location',
              StartDate: new Date('04/02/2019'), Duration: 4, Progress: 50,
              Indicators: [
                {
                  'date': '04/08/2019',
                  'iconClass': 'e-btn-icon e-notes-info e-icons e-
icon-left e-gantt e-notes-info::before',
                  'name': 'Custom String',
                  'tooltip': 'Follow up'
                },
                {
                  'date': '04/11/2019',
                  'iconClass': 'e-btn-icon e-notes-info e-icons e-
icon-left e-gantt e-notes-info::before',
                  'name': '<span style="color:red">String
Template</span>',
                }
              ]
            },
            { TaskID: 3, TaskName: 'Perform Soil test', StartDate: new
Date('04/02/2019'), Duration: 4, Progress: 50 },
            {
              TaskID: 4, TaskName: 'Soil test approval', StartDate:
new Date('04/02/2019'), Duration: 4, Progress: 50, },

```



```

    ],
    {
      TaskID: 5,
      TaskName: 'Project Estimation',
      StartDate: new Date('04/02/2019'),
      EndDate: new Date('04/21/2019'),
      subtasks: [
        {
          TaskID: 6, TaskName: 'Develop floor plan for
estimation', StartDate: new Date('04/04/2019'), Duration: 3, Progress: 50,
          Indicators: [
            {
              'date': '04/10/2019',
              'iconClass': 'e-btn-icon e-notes-info e-icons e-
icon-left e-gantt e-notes-info::before',
              'name': 'Indicator title',
              'tooltip': 'tooltip'
            }
          ]
        },
        { TaskID: 7, TaskName: 'List materials', StartDate: new
Date('04/04/2019'), Duration: 3, Progress: 50 },
        { TaskID: 8, TaskName: 'Estimation approval', StartDate: new
Date('04/04/2019'), Duration: 3, Progress: 50 }
      ]
    },
  ],
  height: '450px',
  taskFields: {
    id: 'TaskID',
    name: 'TaskName',
    startDate: 'StartDate',
    resourceInfo: 'resources',
    duration: 'Duration',
    progress: 'Progress',
    dependency: 'Predecessor',
    child: 'subtasks',
    indicators: 'Indicators'
  }
};
}
};
</script>

```

{% previewsample "page.domainurl/code-snippet/gantt/data-markers-cs1" %}

### Critical path in Vue Gantt component

The critical path in a project is indicated by a single task or a series of tasks. If a task in critical path is delayed, the entire project will be delayed. A task is considered to be critical if any delay to this task would affect the project end date.

The critical path can be enabled in Gantt by using the built-in toolbar button or [enableCriticalPath](#) property.

The following code example shows how to display the critical path in the Gantt control:

**APP.VUE**

```

<template>
  <div>
    <ejs-gantt ref='gantt' id="GanttContainer" :dataSource="data"
    :taskFields = "taskFields" :height = "height" :holidays = "holidays"
    :enableCriticalPath="true" :editSettings="editSettings"></ejs-gantt>
  </div>
</template>
<script>
import Vue from "vue";
import { GanttPlugin, CriticalPath, Edit, Toolbar } from "@syncfusion/ej2-vue-gantt";
import { projectNewData } from './data-source.js';
Vue.use(GanttPlugin);
export default {
  data: function() {
    return{
      data: projectNewData,
      height: '450px',
      taskFields: {
        id: 'TaskID',
        name: 'TaskName',
        startDate: 'StartDate',
        duration: 'Duration',
        progress: 'Progress',
        child: 'subtasks'
      },
      editSettings: {
        allowAdding: true,
        allowEditing: true,
        allowDeleting: true,
        allowTaskbarEditing: true,
        showDeleteConfirmDialog: true
      },
      toolbar: ['CriticalPath'],
    };
  },
  provide: {
    gantt: [CriticalPath, Edit, Toolbar]
  }
};
</script>

```

{% previewsample "page.domainurl/code-snippet/gantt/criticalpath-cs1" %}

**Customize taskbar in critical path**

The taskbar in critical path can be customized by using [queryTaskbarInfo](#) event and [isCritical](#) property of row [data](#) in the event argument.

The following code example shows how to customize the critical path taskbar in the Gantt control:

**APP.VUE**

```

<template>
  <div>

```

```

<ejs-gantt ref='gantt' id="GanttContainer" :dataSource="data"
:taskFields = "taskFields" :height = "height"
:queryTaskbarInfo="queryTaskbarInfo" :holidays = "holidays"
:enableCriticalPath="true" :editSettings="editSettings"></ejs-gantt>
</div>
</template>
<script>
import Vue from "vue";
import { GanttPlugin, CriticalPath, Edit, Toolbar } from "@syncfusion/ej2-vue-gantt";
import { projectNewData } from './data-source.js';
Vue.use(GanttPlugin);
export default {
  data: function() {
    return {
      data: projectNewData,
      height: '450px',
      taskFields: {
        id: 'TaskID',
        name: 'TaskName',
        startDate: 'StartDate',
        duration: 'Duration',
        progress: 'Progress',
        child: 'subtasks'
      },
      editSettings: {
        allowAdding: true,
        allowEditing: true,
        allowDeleting: true,
        allowTaskbarEditing: true,
        showDeleteConfirmDialog: true
      },
      toolbar: ['CriticalPath'],
      queryTaskbarInfo: function(args) {
        if ((args.data.isCritical || args.data.slack === '0 day') &&
!args.data.hasChildRecords) {
          args.taskbarBgColor = 'rgb(242, 210, 189)';
          args.progressBarBgColor = 'rgb(201, 169, 166)';
        }
      }
    };
  },
  provide: {
    gantt: [CriticalPath, Edit, Toolbar]
  }
};
</script>

```

{% previewsample "page.domainurl/code-snippet/gantt/customizeCriticalPath-cs1" %}

### Toolbar in Vue Gantt component

The Gantt component provides the toolbar support to handle Gantt actions. The [toolbar](#) property accepts the collection of built-in toolbar items and `ItemModel` objects for custom toolbar items.

To use toolbar feature, inject the `Toolbar` module in the [Link to the Video](#) section.

To learn about Gantt Chart Toolbar Options, you can check on this video:

### Built-in toolbar items

Built-in toolbar items execute standard actions of the Gantt component, and these items can be added to toolbar by defining the [toolbar](#) as a collection of built-in items. It renders the button with icon and text.

The following table shows built-in toolbar items and its actions.

Built-in Toolbar Items   Actions	
----- -----	
ExpandAll	Expands all the rows.
CollapseAll	Collapses all the rows.
Add	Adds a new record.
Edit	Edits the selected record.
Indent	Indent the selected record to one level.
Outdent	Outdent the elected record to one level.
Update	Updates the edited record.
Delete	Deletes the selected record.
Cancel	Cancels the edit state.
Search	Searches the records by the given key.
PrevTimeSpan	Navigate the Gantt timeline to previous time span.
NextTimeSpan	Navigate the Gantt timeline to Next time span.

### APP.VUE

```

<template>
  <div>
    <ejs-gantt ref='gantt' id="GanttContainer" :dataSource="data"
    :taskFields = "taskFields" :height = "height" :toolbar="toolbar"
    :editSettings= "editSettings"></ejs-gantt>
  </div>
</template>
<script>
import Vue from "vue";
import { GanttPlugin, Edit, Selection, Toolbar } from "@syncfusion/ej2-vue-gantt";
import { projectNewData } from './data-source.js';
Vue.use(GanttPlugin);
export default {
  data: function() {
    return{
      data: projectNewData,
      height: '450px',
      taskFields: {
        id: 'TaskID',
        name: 'TaskName',
        startDate: 'StartDate',

```

```

        endDate: 'EndDate',
        duration: 'Duration',
        progress: 'Progress',
        dependency: 'Predecessor',
        child: 'subtasks'
    },
    toolbar: ['Add', 'Edit', 'Delete', 'Cancel', 'ExpandAll',
    'CollapseAll', 'PrevTimeSpan', 'NextTimeSpan', 'Update', 'Indent', 'Outdent'],
    editSettings: {
        allowAdding: true,
        allowEditing: true,
        allowDeleting: true,
        allowTaskbarEditing: true,
        showDeleteConfirmDialog: true
    },
    };
},
provide: {
    gantt: [ Edit, Selection, Toolbar ]
}
};
</script>

```

{% previewsample "page.domainurl/code-snippet/gantt/toolbar-cs1" %}

\* The [toolbar](#) has options to define both built-in and custom toolbar items.

#### Custom toolbar items

Custom toolbar items can be added to the toolbar by defining the [toolbar](#) property as a collection of **ItemModels**.

Actions for this customized toolbar items are defined in the [toolbarClick](#) event.

By default, the custom toolbar items are at left position. You can change the position by using the **align** property. In the following sample, the **Quick Filter** toolbar item is positioned at right.

#### APP.VUE

```

<template>
  <div>
    <ejs-gantt ref='gantt' id="GanttContainer" :dataSource="data"
    :taskFields = "taskFields" :height = "height" :toolbarClick="toolbarClick"
    :columns="columns" :toolbar="toolbar" :allowFiltering='true'></ejs-gantt>
  </div>
</template>
<script>
import Vue from "vue";
import { GanttPlugin, Toolbar, Filter } from "@syncfusion/ej2-vue-gantt";
import { ClickEventArgs } from "@syncfusion/ej2-navigations";
import { editingData } from './data-source.js';
Vue.use(GanttPlugin);
export default {
  data: function() {
    return{
      data: editingData,
      height: '450px',
    }
  }
}

```

```

        taskFields: {
            id: 'TaskID',
            name: 'TaskName',
            startDate: 'StartDate',
            endDate: 'EndDate',
            duration: 'Duration',
            progress: 'Progress',
            dependency: 'Predecessor',
            child: 'subtasks'
        },
        columns: [
            { field: 'TaskID', headerText: 'Task ID', textAlign: 'Left',
width: '100' },
            { field: 'TaskName', headerText: 'Task Name', width: '250' },
            { field: 'StartDate', headerText: 'Start Date', width: '150' },
            { field: 'Duration', headerText: 'Duration', width: '150' },
            { field: 'Progress', headerText: 'Progress', width: '150' }
        ],
        toolbarClick: (args: ClickEventArgs) => {
            if (args.item.id === 'toolbarfilter') {
                var ganttObj =
document.getElementById('GanttContainer').ej2_instances[0];
                ganttObj.filterByColumn('TaskName', 'startswith',
'Identify');
            }
        },
        toolbar: [{text: 'Quick Filter', tooltipText: 'Quick Filter',
id: 'toolbarfilter',align:'Right', prefixIcon: 'e-quickfilter' }],
    },
    provide: {
        gantt: [ Toolbar, Filter ]
    }
};
</script>
<style>
    .e-quickfilter::before {
        content: '\e7ee';
    }
</style>

```

{% previewsample "page.domainurl/code-snippet/gantt/toolbar-cs2" %}

\* The [toolbar](#) has options to define both built-in and custom toolbar items.

\* If a toolbar item does not match the built-in items, it will be treated as a custom toolbar item.

#### Built-in and custom items in toolbar

The Gantt component has an option to use both built-in and custom toolbar items at the same time.

In the following example, the **ExpandAll** and **CollapseAll** are built-in toolbar items and **Test** is the custom toolbar item.

#### APP.VUE

```

<template>
  <div>

```

```

<ejs-gantt ref='gantt' id="GanttContainer" :dataSource="data"
:taskFields = "taskFields" :height = "height" :columns="columns"
:toolbarClick="toolbarClick" :toolbar="toolbar"
:allowFiltering='true'></ejs-gantt>
</div>
</template>
<script>
import Vue from "vue";
import { GanttPlugin, Toolbar, Filter } from "@syncfusion/ej2-vue-gantt";
import { EmitType } from '@syncfusion/ej2-base';
import { ClickEventArgs } from '@syncfusion/ej2-navigations';
import { editingData } from './data-source.js';
Vue.use(GanttPlugin);
export default {
  data: function() {
    return{
      data: editingData,
      height: '450px',
      taskFields: {
        id: 'TaskID',
        name: 'TaskName',
        startDate: 'StartDate',
        endDate: 'EndDate',
        duration: 'Duration',
        progress: 'Progress',
        dependency: 'Predecessor',
        child: 'subtasks'
      },
      columns: [
        { field: 'TaskID', headerText: 'Task ID', textAlign: 'Left',
width: '100' },
        { field: 'TaskName', headerText: 'Task Name', width: '250' },
        { field: 'StartDate', headerText: 'Start Date', width: '150' },
        { field: 'Duration', headerText: 'Duration', width: '150' },
        { field: 'Progress', headerText: 'Progress', width: '150' },
      ],
      toolbarClick: (args: ClickEventArgs) => {
        if (args.item.text === 'Click') {
          alert("Custom toolbar click...");
        }
      },
      toolbar: ['ExpandAll', 'CollapseAll', { text: 'Click',
tooltipText: 'Click',id: 'Click' }],
    };
  },
  provide: {
    gantt: [ Toolbar, Filter ]
  },
};
</script>

```

{% previewsample "page.domainurl/code-snippet/gantt/toolbar-cs3" %}

### Enable/disable toolbar items

You can enable or disable the toolbar items by using the `enableItems` method.

**APP.VUE**

```

<template>
  <div>
    <ejs-button id="enable" cssClass="e-info" v-
on:click.native="enable">Enable</ejs-button>
    <ejs-button id="disable" cssClass="e-info" v-
on:click.native="disable">Disable</ejs-button>
    <br>
    <ejs-gantt ref='gantt' id="GanttContainer" :dataSource="data"
:taskFields = "taskFields" :height = "height" :toolbar="toolbar"
:toolbarClick="toolbarClick" :allowFiltering='true'></ejs-gantt>
  </div>
</template>
<script>
import Vue from "vue";
import { GanttPlugin, Toolbar, Filter } from "@syncfusion/ej2-vue-gantt";
import { ButtonPlugin } from "@syncfusion/ej2-vue-buttons";
import { ClickEventArgs } from '@syncfusion/ej2-navigations';
import { editingData } from './data-source.js';
Vue.use(GanttPlugin);
Vue.use(ButtonPlugin);
export default {
  data: function() {
    return{
      data: editingData,
      height: '450px',
      taskFields: {
        id: 'TaskID',
        name: 'TaskName',
        startDate: 'StartDate',
        endDate: 'EndDate',
        duration: 'Duration',
        progress: 'Progress',
        dependency: 'Predecessor',
        child: 'subtasks'
      },
      toolbar: ['QuickFilter', 'ClearFilter'],
      toolbarClick: (args: ClickEventArgs) => {
        if (args.item.text === 'QuickFilter') {
          var ganttChart =
document.getElementById('GanttContainer').ej2_instances[0];
          ganttChart.filterByColumn('TaskName', 'startswith',
'Identify');
        }
        if (args.item.text === 'ClearFilter') {
          var ganttChart =
document.getElementById('GanttContainer').ej2_instances[0];
          ganttChart.clearFiltering();
        }
      },
    };
  },
  provide: {
    gantt: [ Toolbar, Filter ]
  },
  methods: {

```



```

        enable: function(e) {
            var ganttChart =
document.getElementById('GanttContainer').ej2_instances[0];
            ganttChart.toolbarModule.enableItems([ganttChart.element.id +
'_QuickFilter', ganttChart.element.id + '_ClearFilter'], true); // enable
toolbar items.
        },
        disable: function(e) {
            var ganttChart =
document.getElementById('GanttContainer').ej2_instances[0];
            ganttChart.toolbarModule.enableItems([ganttChart.element.id +
'_QuickFilter', ganttChart.element.id + '_ClearFilter'], false); // disable
toolbar items.
        },
    },
};
</script>

```

{% previewsample "page.domainurl/code-snippet/gantt/toolbar-cs4" %}

### Add input elements in toolbar

In the Gantt toolbar, you can add EJ2 editor elements like numeric text box, drop-down list, and date picker controls. The following code snippets demonstrates how to add EJ2 editors to the Gantt toolbar.

### APP.VUE

```

<template>
  <div>
    <ejs-numerictextbox id="inputEle" format='c2' value='1'
width='150'></ejs-numerictextbox>
    <br>
    <ejs-gantt ref='gantt' id="GanttContainer" :dataSource="data"
:taskFields = "taskFields" :height = "height"></ejs-gantt>
  </div>
</template>
<script>
import Vue from "vue";
import { GanttPlugin, Toolbar } from "@syncfusion/ej2-vue-gantt";
import { NumericTextBoxPlugin } from "@syncfusion/ej2-vue-inputs";
import { projectNewData } from './data-source.js';
Vue.use(GanttPlugin);
Vue.use(NumericTextBoxPlugin);
export default {
  data: function() {
    return{
      data: projectNewData,
      height: '450px',
      taskFields: {
        id: 'TaskID',
        name: 'TaskName',
        startDate: 'StartDate',
        endDate: 'EndDate',
        duration: 'Duration',
        progress: 'Progress',
        dependency: 'Predecessor',
        child: 'subtasks'
      }
    }
  }
}

```

```

    },
    };
  },
  provide: {
    gantt: [Toolbar ]
  }
};
</script>

```

```
{% previewsample "page.domainurl/code-snippet/gantt/toolbar-cs5" %}
```

## Managing Tasks

### Managing tasks in Vue Gantt component

The Gantt component has options to dynamically insert, delete, and update tasks in the project. The primary key column is necessary to manage the tasks and perform CRUD operations in Gantt. To define the primary key, set the [columns.isPrimaryKey](#) property to **true** in the particular column.

To use CRUD, inject the [Edit](#) module in the **provide** section.

**Note:** If the [Edit](#) module is not injected, you cannot edit the tasks through the TreeGrid cells.

The following code example demonstrates how to enable cell editing in the Gantt component.

### APP.VUE

```

<template>
  <div>
    <ejs-gantt ref='gantt' id="GanttContainer" :dataSource="data"
    :taskFields = "taskFields" :height = "height" :columns = "columns"
    :editSettings= "editSettings"></ejs-gantt>
  </div>
</template>
<script>
import Vue from "vue";
import { GanttPlugin, Edit } from "@syncfusion/ej2-vue-gantt";
Vue.use(GanttPlugin);
export default {
  data: function() {
    return {
      data: [
        {
          TaskID: 1,
          TaskName: 'Project Initiation',
          StartDate: new Date('04/02/2019'),
          EndDate: new Date('04/21/2019'),
          subtasks: [
            { TaskID: 2, TaskName: 'Identify Site location',
              StartDate: new Date('04/02/2019'), Duration: 4, Progress: 50 },
            { TaskID: 3, TaskName: 'Perform Soil test', StartDate:
              new Date('04/02/2019'), Duration: 4, Progress: 50 },
            { TaskID: 4, TaskName: 'Soil test approval', StartDate:
              new Date('04/02/2019'), Duration: 4, Progress: 50 },
          ]
        },
        {
          TaskID: 5,

```

```

        TaskName: 'Project Estimation',
        StartDate: new Date('04/02/2019'),
        EndDate: new Date('04/21/2019'),
        subtasks: [
            { TaskID: 6, TaskName: 'Develop floor plan for
estimation', StartDate: new Date('04/04/2019'), Duration: 3, Progress: 50 },
            { TaskID: 7, TaskName: 'List materials', StartDate: new
Date('04/04/2019'), Duration: 3, Progress: 50 },
            { TaskID: 8, TaskName: 'Estimation approval', StartDate:
new Date('04/04/2019'), Duration: 3, Progress: 50 }
        ]
    },
],
height: '450px',
taskFields: {
    id: 'TaskID',
    name: 'TaskName',
    startDate: 'StartDate',
    endDate: 'EndDate',
    duration: 'Duration',
    progress: 'Progress',
    child: 'subtasks'
},
columns: [
    { field: 'TaskID', headerText: 'Task ID', textAlign: 'Left',
width: '100' },
    { field: 'TaskName', headerText: 'Task Name', width: '250'
},
    { field: 'StartDate', headerText: 'Start Date', width: '150'
},
    { field: 'Duration', headerText: 'Duration', width: '150' },
    { field: 'Progress', headerText: 'Progress', width: '150' },
],
editSettings: {
    allowEditing: true,
    mode: "Auto"
},
];
},
provide: {
    gantt: [ Edit ]
}
};
</script>

```

{% previewsample "page.domainurl/code-snippet/gantt/managing-tasks-cs11" %}

**Note:** When the edit mode is set to **Auto**, on performing double-click action on TreeGrid side, the cells will be changed to editable mode and on performing double-click action on chart side, the edit dialog will appear for editing the task details.

*Cell edit type and its params*

The [columns.editType](#) is used to define the edit type for any particular column.

You can set the [columns.editType](#) based on data type of the column.

- **numericedit** - [NumericTextBox](#) component for integers, double, and decimal data types.
- **defaultedit** - [TextBox](#) component for string data type.
- **dropdownedit** - [DropDownList](#) component to show all unique values related to that field.
- **booleanedit** - [CheckBox](#) component for boolean data type.
- **datepickeredit** - [DatePicker](#) component for date data type.
- **datetimepickeredit** - [DateTimePicker](#) component for date time data type.

Also, you can customize the behavior of the editor component through the [columns.edit.params](#).

The following table describes cell edit type component and their corresponding edit params of the column.

Edit Type | Component | Example

[Cell editing](#) | To perform **double tap** on a specific cell, initiate the cell to be in edit state.

[Dialog editing](#) | To perform **double tap** on a specific row, initiate the edit dialog to be opened.

[Taskbar editing](#) | Taskbar editing action is initiated using the **tap** action on the taskbar. **Parent taskbar** : Once you tap on the parent taskbar, it will be changed to editing state. Perform only dragging

action on parent taskbar editing. **Child taskbar**

: Once you tap the child taskbar, it will be changed to editing state.



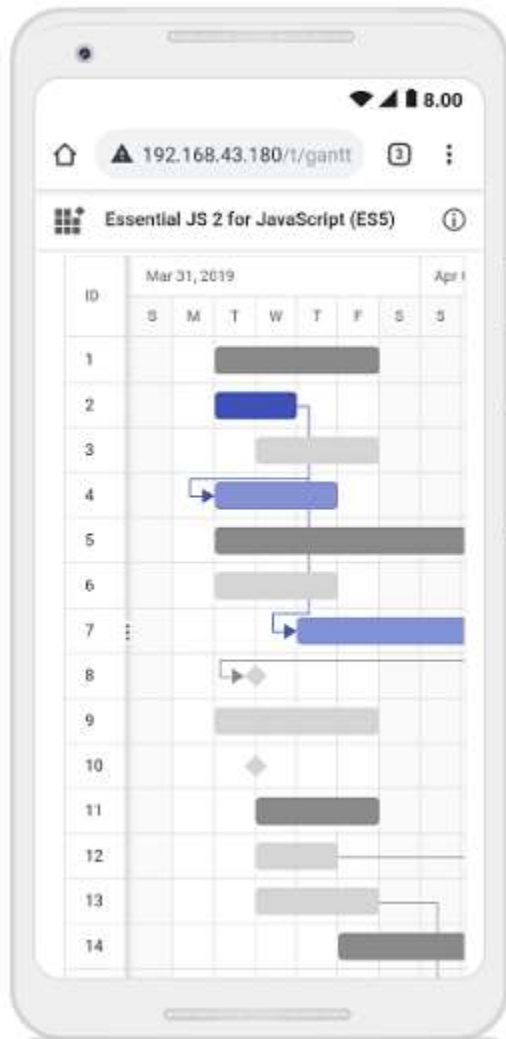
**Dragging taskbar** : To drag a taskbar to the

left or right in editing state. **Resizing taskbar** : To resize a taskbar, drag the left/right resize icon. **Progress resizing** : To change the progress, drag the progress resize icon to the left or right direction.

[Task dependency editing](#)

You can **tap** the left/right connector point to initiate [task dependencies](#) edit mode and again tap another taskbar to establish the dependency line between two taskbars.

The following table explains the taskbar state in dependency edit mode.



Taskbar state | Description

Parent taskbar | You cannot create dependency relationship to parent tasks. <br>



Taskbar without dependency | If you tap a valid child taskbar, it will create FS type dependency line between tasks, otherwise exits from task dependency edit mode. <br>

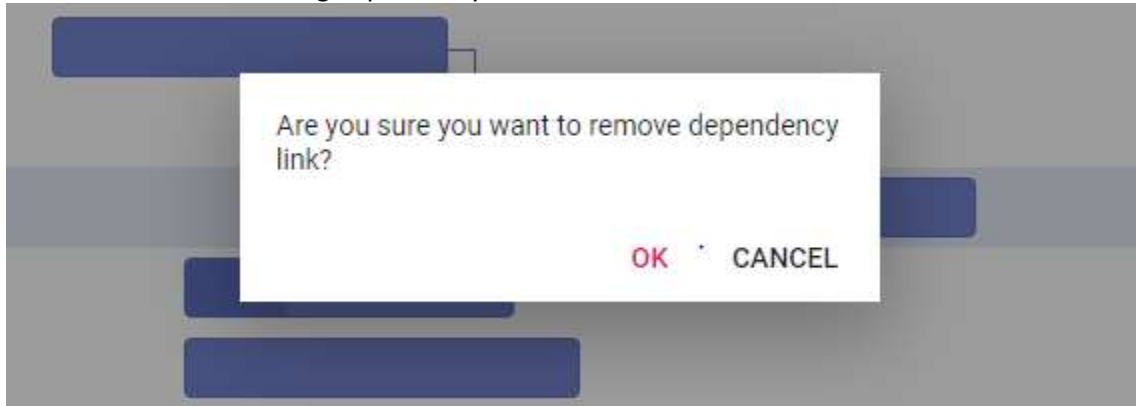


Taskbar with dependency | If you tap the second taskbar, which has already been directly connected,



it will ask to remove it. <br>

**Removing dependency** | Once you tap the taskbar with direct dependency, then confirmation dialog will be shown for removing dependency. <br>



### APP.VUE

```
<template>
  <div>
    <ejs-gantt ref='gantt' id="GanttContainer" :dataSource="data"
:taskFields = "taskFields" :height = "height" :editSettings= "editSettings"
:load= "load"></ejs-gantt>
  </div>
</template>
<script>
import Vue from "vue";
import { GanttPlugin, Edit, Selection } from "@syncfusion/ej2-vue-gantt";
Vue.use(GanttPlugin);
export default {
  data: function() {
    return{
      data: [
        {
          TaskID: 1,
          TaskName: 'Project Initiation',
          StartDate: new Date('04/02/2019'),
          EndDate: new Date('04/21/2019'),
          subtasks: [
            { TaskID: 2, TaskName: 'Identify Site location',
StartDate: new Date('04/02/2019'), Duration: 3, Progress: 50 },
            { TaskID: 3, TaskName: 'Perform Soil test', StartDate:
new Date('04/02/2019'), Duration: 4, Progress: 50 },
            { TaskID: 4, TaskName: 'Soil test approval', StartDate:
new Date('04/02/2019'), Duration: 4,Predecessor:"2FS", Progress: 50 },
          ]
        },
        {
          TaskID: 5,
          TaskName: 'Project Estimation',
          StartDate: new Date('04/02/2019'),
          EndDate: new Date('04/21/2019'),
          subtasks: [
            { TaskID: 6, TaskName: 'Develop floor plan for
estimation', StartDate: new Date('04/04/2019'), Duration: 3, Progress: 50 },
```

```

        { TaskID: 7, TaskName: 'List materials', StartDate: new
Date('04/04/2019'), Duration: 3, Progress: 50 },
        { TaskID: 8, TaskName: 'Estimation approval', StartDate:
new Date('04/04/2019'), Duration: 4, Predecessor: "6SS", Progress: 50 }
    ]
  },
],
height: '450px',
taskFields: {
  id: 'TaskID',
  name: 'TaskName',
  startDate: 'StartDate',
  duration: 'Duration',
  progress: 'Progress',
  dependency: 'Predecessor',
  child: 'subtasks'
},
editSettings: {
  allowTaskbarEditing: true
},
load: function() {
  var ganttObj =
document.getElementById('GanttContainer').ej2_instances[0];
  ganttObj.isAdaptive = true; // Forcing desktop layout to
change as mobile layout
}
};
},
provide: {
  gantt: [ Edit, Selection ]
}
};
</script>
<style>
@import '../node_modules/@syncfusion/ej2-vue-gantt/styles/material.css';
</style>

```

{% previewsample "page.domainurl/code-snippet/gantt/touch-cs1" %}

Note: In mobile device, you cannot create dependency other than **FS** by taskbar editing. By using cell/dialog editing, you can add all type of dependencies.

#### Taskbar editing tooltip

The taskbar editing tooltip can be customized using the [tooltipSettings.editing](#) property. The following code example shows how to customize the taskbar editing tooltip in Gantt.

#### APP.VUE

```

<template>
  <div>
    <ejs-gantt ref='gantt' :dataSource='data' id='GanttContainer'
:taskFields='taskFields' :height='height'
:editSettings='editSettings' :tooltipSettings='tooltipSettings'>
      <template v-slot:editingTooltipTemplate='{data}'>
        <div>Duration : {{data.duration}}</div>
      </template>
    </div>
  </div>

```

```

    </ejs-gantt>
  </div>
</template>
<script>
import Vue from "vue";
import { GanttPlugin, Edit } from "@syncfusion/ej2-vue-gantt";
import { editingData } from './data-source.js';
Vue.use(GanttPlugin);
export default {
  data: function() {
    return{
      data: editingData,
      height: '450px',
      taskFields: {
        id: 'TaskID',
        name: 'TaskName',
        startDate: 'StartDate',
        duration: 'Duration',
        baselineStartDate: "BaselineStartDate",
        baselineEndDate: "BaselineEndDate",
        progress: 'Progress',
        dependency: 'Predecessor',
        child: 'subtasks'
      },
      editSettings: {
        allowEditing: true,
        allowTaskbarEditing: true
      },
      tooltipSettings: {
        showTooltip: true,
        editing: "editingTooltipTemplate"
      },
    };
  },
  provide: {
    gantt: [ Edit ]
  },
};
</script>

```

{% previewsample "page.domainurl/code-snippet/gantt/tooltip-cs6" %}

### Adding new tasks in Vue Gantt component

Tasks can be dynamically added to the Gantt project by enabling the [editSettings.allowAdding](#) property.

#### Toolbar

A row can be added to the Gantt component from the toolbar while the [editSettings.allowAdding](#) property is set to `true`. On clicking the toolbar add icon, you should provide the task information in the add dialog.

### APP.VUE

```

<template>
  <div>

```



```

    <ejs-gantt ref='gantt' id="GanttContainer" :dataSource="data"
:taskFields = "taskFields" :height = "height" :toolbar="toolbar"
:editSettings= "editSettings"></ejs-gantt>
  </div>
</template>
<script>
import Vue from "vue";
import { GanttPlugin, Edit, Toolbar } from "@syncfusion/ej2-vue-gantt";
Vue.use(GanttPlugin);
export default {
  data: function() {
    return{
      data: [
        {
          TaskID: 1,
          TaskName: 'Project Initiation',
          StartDate: new Date('04/02/2019'),
          EndDate: new Date('04/21/2019'),
          subtasks: [
            { TaskID: 2, TaskName: 'Identify Site location',
StartDate: new Date('04/02/2019'), Duration: 4, Progress: 50 },
            { TaskID: 3, TaskName: 'Perform Soil test', StartDate:
new Date('04/02/2019'), Duration: 4, Progress: 50 },
            { TaskID: 4, TaskName: 'Soil test approval', StartDate:
new Date('04/02/2019'), Duration: 4, Progress: 50 },
          ]
        },
        {
          TaskID: 5,
          TaskName: 'Project Estimation',
          StartDate: new Date('04/02/2019'),
          EndDate: new Date('04/21/2019'),
          subtasks: [
            { TaskID: 6, TaskName: 'Develop floor plan for
estimation', StartDate: new Date('04/04/2019'), Duration: 3, Progress: 50 },
            { TaskID: 7, TaskName: 'List materials', StartDate: new
Date('04/04/2019'), Duration: 3, Progress: 50 },
            { TaskID: 8, TaskName: 'Estimation approval', StartDate:
new Date('04/04/2019'), Duration: 3, Progress: 50 }
          ]
        },
      ],
      height: '450px',
      taskFields: {
        id: 'TaskID',
        name: 'TaskName',
        startDate: 'StartDate',
        duration: 'Duration',
        progress: 'Progress',
        child: 'subtasks'
      },
      editSettings:{
        allowAdding:true
      },
      toolbar: ['Add']
    };
  },
};

```

```

    provide: {
      gantt: [ Edit, Toolbar ]
    }
  };
</script>

```

{% previewsample "page.domainurl/code-snippet/gantt/managing-tasks-cs1" %}

By default, the new row will be added to the top most row in the Gantt component.

### Context menu

A row can also be added above, below or child of the selected row by using context menu support. For this, we need to enable the property [enableContextMenu](#) and inject the [ContextMenu](#) module into the Gantt control.

### APP.VUE

```

<template>
  <div>
    <ejs-gantt ref='gantt' id="contextMenu" :dataSource="data"
    :taskFields = "taskFields" :height = "height" :editSettings="editSettings"
    :toolbar="toolbar" :enableContextMenu="true" :allowSorting="true"
    :allowResizing= "true"></ejs-gantt>
  </div>
</template>
<script>
import Vue from "vue";
import { GanttPlugin, ContextMenu, Edit, Toolbar, Selection } from
"@syncfusion/ej2-vue-gantt";
import { editingData } from './data-source.js';
Vue.use(GanttPlugin);
export default {
  data: function() {
    return{
      data: editingData,
      height: '450px',
      taskFields: {
        id: 'TaskID',
        name: 'TaskName',
        startDate: 'StartDate',
        duration: 'Duration',
        progress: 'Progress',
        dependency: 'Predecessor',
        child: 'subtasks'
      },
      editSettings: {
        allowAdding: true
      },
      toolbar: ['Add']
    };
  },
  provide: {
    gantt: [ ContextMenu, Edit, Toolbar, Selection]
  }
};
</script>

```

```
{% previewsample "page.domainurl/code-snippet/gantt/managing-tasks-cs2" %}
```

### Using method

You can add rows to the Gantt component dynamically using the [addRecord](#) method and you can define the add position of the default new record by using the [rowPosition](#) property. You can also pass the `rowIndex` as an additional parameter.

- Top of all the rows
- Bottom to all the existing rows
- Above the selected row
- Below the selected row
- As child to the selected row

### APP.VUE

```
<template>
  <div>
    <ejs-button id="addRow" cssClass="e-info" v-
on:click.native="add">Add Row</ejs-button>
    <br><br><br>
    <ejs-gantt ref='gantt' id="GanttContainer" :dataSource="data"
:taskFields = "taskFields" :height = "height" :editSettings=
"editSettings"></ejs-gantt>
  </div>
</template>
<script>
import Vue from "vue";
import { GanttPlugin, Edit } from "@syncfusion/ej2-vue-gantt";
import { ButtonPlugin } from "@syncfusion/ej2-vue-buttons";
import { projectNewData } from './data-source.js';
Vue.use(GanttPlugin);
Vue.use(ButtonPlugin);
export default {
  data: function() {
    return{
      data: [
        {
          TaskID: 1,
          TaskName: 'Project Initiation',
          StartDate: new Date('04/02/2019'),
          EndDate: new Date('04/21/2019'),
          isParent:true,
          subtasks: [
            { TaskID: 2, TaskName: 'Identify Site location', StartDate:
new Date('04/02/2019'), Duration: 3, Progress: 50,isParent:false },
            { TaskID: 3, TaskName: 'Perform Soil test', StartDate: new
Date('04/02/2019'), Duration: 3, Progress: 70, resources: [2, 3,
5],isParent:false },
            { TaskID: 4, TaskName: 'Soil test approval', StartDate: new
Date('04/02/2019'), Duration: 3, Predecessor:"2FS", Progress:
80,isParent:false },
          ]
        }
      ],
    }
  },
}
```

```

        {
            TaskID: 5,
            TaskName: 'Project Estimation',
            StartDate: new Date('04/02/2019'),
            EndDate: new Date('04/21/2019'),
            isParent:true,
            subtasks: [
                { TaskID: 6, TaskName: 'Develop floor plan for estimation',
                StartDate: new Date('04/04/2019'), Duration: 4, Progress: 50, resources:
                [4],isParent:false },
                { TaskID: 7, TaskName: 'List materials', StartDate: new
                Date('04/04/2019'), Duration: 4, Progress: 50, DurationUnit:'day',
                resources: [4, 8],isParent:false },
                { TaskID: 8, TaskName: 'Estimation approval', StartDate: new
                Date('04/04/2019'), Duration: 4,Predecessor:"6SS", DurationUnit:'minute',
                Progress: 70, resources: [12, 5],isParent:false }
            ]
        },
    ],
    height: '450px',
    taskFields: {
        id: 'TaskID',
        name: 'TaskName',
        startDate: 'StartDate',
        duration: 'Duration',
        progress: 'Progress',
        child: 'subtasks'
    },
    editSettings:{
        allowAdding:true
    },
};

},
provide: {
    gantt: [ Edit ]
},
methods: {
    add: function(e){
        var ganttObj =
document.getElementById('GanttContainer').ej2_instances[0];
        var record= {
            TaskID: 9,
            TaskName: 'Identify Site location',
            StartDate: new Date('04/02/2019'),
            Duration: 3,
            Progress: 50
        };
        ganttObj.editModule.addRecord(record, 'Below', 2);
    }
},
};
</script>

```

{% previewsample "page.domainurl/code-snippet/gantt/managing-tasks-cs3" %}

### Editing tasks in Vue Gantt component

The editing feature can be enabled in the Gantt component by enabling the [editSettings.allowEditing](#) and [Link to the Video](#) properties.

The following editing options are available to update the tasks in Gantt,

- Cell
- Dialog
- Taskbar
- Connector line

To get start quickly with Gantt Chart Editing Options, you can check on this video:

#### Cell editing

By setting the edit mode to auto using the [mode](#) property, the tasks can be edited through TreeGrid cells by double-clicking.

**Note:** If the [Edit](#) module is not injected, you cannot edit the tasks through the TreeGrid cells.

The following code example demonstrates how to enable cell editing in the Gantt component.

#### APP.VUE

```
<template>
  <div>
    <ejs-gantt ref='gantt' id="GanttContainer" :dataSource="data"
:taskFields = "taskFields" :height = "height" :columns = "columns"
:editSettings= "editSettings"></ejs-gantt>
  </div>
</template>
<script>
import Vue from "vue";
import { GanttPlugin, Edit } from "@syncfusion/ej2-vue-gantt";
Vue.use(GanttPlugin);
export default {
  data: function() {
    return {
      data: [
        {
          TaskID: 1,
          TaskName: 'Project Initiation',
          StartDate: new Date('04/02/2019'),
          EndDate: new Date('04/21/2019'),
          subtasks: [
            { TaskID: 2, TaskName: 'Identify Site location',
StartDate: new Date('04/02/2019'), Duration: 4, Progress: 50 },
            { TaskID: 3, TaskName: 'Perform Soil test', StartDate:
new Date('04/02/2019'), Duration: 4, Progress: 50 },
            { TaskID: 4, TaskName: 'Soil test approval', StartDate:
new Date('04/02/2019'), Duration: 4, Progress: 50 },
          ]
        },
        {
          TaskID: 5,
          TaskName: 'Project Estimation',
          StartDate: new Date('04/02/2019'),
```

```

        EndDate: new Date('04/21/2019'),
        subtasks: [
            { TaskID: 6, TaskName: 'Develop floor plan for
estimation', StartDate: new Date('04/04/2019'), Duration: 3, Progress: 50 },
            { TaskID: 7, TaskName: 'List materials', StartDate: new
Date('04/04/2019'), Duration: 3, Progress: 50 },
            { TaskID: 8, TaskName: 'Estimation approval', StartDate:
new Date('04/04/2019'), Duration: 3, Progress: 50 }
        ]
    },
    ],
    height: '450px',
    taskFields: {
        id: 'TaskID',
        name: 'TaskName',
        startDate: 'StartDate',
        endDate: 'EndDate',
        duration: 'Duration',
        progress: 'Progress',
        child: 'subtasks'
    },
    columns: [
        { field: 'TaskID', headerText: 'Task ID', textAlign: 'Left',
width: '100' },
        { field: 'TaskName', headerText: 'Task Name', width: '250'
},
        { field: 'StartDate', headerText: 'Start Date', width: '150'
},
        { field: 'Duration', headerText: 'Duration', width: '150' },
        { field: 'Progress', headerText: 'Progress', width: '150' },
    ],
    editSettings: {
        allowEditing: true,
        mode: "Auto"
    },
    };
    },
    provide: {
        gantt: [ Edit ]
    }
    };
</script>

```

{% previewsample "page.domainurl/code-snippet/gantt/managing-tasks-cs6" %}

**Note:** When the edit mode is set to **Auto**, on performing double-click action on TreeGrid side, the cells will be changed to editable mode and on performing double-click action on chart side, the edit dialog will appear for editing the task details.

#### Dialog editing

Modify the task details through the edit dialog by setting the edit [mode](#) property as **Dialog**.

#### APP.VUE

```

<template>
  <div>

```

```

    <ejs-gantt ref='gantt' id="GanttContainer" :dataSource="data"
:taskFields = "taskFields" :height = "height" :columns = "columns"
:editSettings= "editSettings"></ejs-gantt>
  </div>
</template>
<script>
import Vue from "vue";
import { GanttPlugin, Edit } from "@syncfusion/ej2-vue-gantt";
Vue.use(GanttPlugin);
export default {
  data: function() {
    return {
      data: [
        {
          TaskID: 1,
          TaskName: 'Project Initiation',
          StartDate: new Date('04/02/2019'),
          EndDate: new Date('04/21/2019'),
          subtasks: [
            { TaskID: 2, TaskName: 'Identify Site location',
StartDate: new Date('04/02/2019'), Duration: 4, Progress: 50 },
            { TaskID: 3, TaskName: 'Perform Soil test', StartDate:
new Date('04/02/2019'), Duration: 4, Progress: 50 },
            { TaskID: 4, TaskName: 'Soil test approval', StartDate:
new Date('04/02/2019'), Duration: 4, Progress: 50 },
          ]
        },
        {
          TaskID: 5,
          TaskName: 'Project Estimation',
          StartDate: new Date('04/02/2019'),
          EndDate: new Date('04/21/2019'),
          subtasks: [
            { TaskID: 6, TaskName: 'Develop floor plan for
estimation', StartDate: new Date('04/04/2019'), Duration: 3, Progress: 50 },
            { TaskID: 7, TaskName: 'List materials', StartDate: new
Date('04/04/2019'), Duration: 3, Progress: 50 },
            { TaskID: 8, TaskName: 'Estimation approval', StartDate:
new Date('04/04/2019'), Duration: 3, Progress: 50 }
          ]
        }
      ],
      height: '450px',
      taskFields: {
        id: 'TaskID',
        name: 'TaskName',
        startDate: 'StartDate',
        endDate: 'EndDate',
        duration: 'Duration',
        progress: 'Progress',
        child: 'subtasks'
      },
      columns: [
        { field: 'TaskID', headerText: 'Task ID', textAlign: 'Left',
width: '100' },
        { field: 'TaskName', headerText: 'Task Name', width: '250'
}
      ],
    },
  },
};

```

```

        { field: 'StartDate', headerText: 'Start Date', width: '150'
    },
        { field: 'Duration', headerText: 'Duration', width: '150' },
        { field: 'Progress', headerText: 'Progress', width: '150' },
    ],
    editSettings: {
        allowEditing: true,
        mode: "Dialog"
    },
    };
},
provide: {
    gantt: [ Edit ]
}
};
</script>

```

{% previewsample "page.domainurl/code-snippet/gantt/managing-tasks-cs7" %}

**Note:** In dialog editing mode, the edit dialog appears when performing double-click action on both TreeGrid or Gantt chart sides.

#### Sections or tabs in dialog

In the Gantt dialog, you can define the required tabs or editing sections using the [addDialogFields](#) and [editDialogFields](#) properties. Every tab is defined using the [type](#) property.

#### APP.VUE

```

<template>
  <div>
    <ejs-gantt ref='gantt' id="GanttContainer" :dataSource="data"
      :taskFields= "taskFields" :height= "height"
      :editDialogFields="editDialogFields" :addDialogFields="addDialogFields"
      :editSettings="editSettings" :toolbar="toolbar" :resourceFields=
        "resourceFields" :resources= "resources"
      :labelSettings="labelSettings"></ejs-gantt>
    </div>
  </template>
<script>
import Vue from "vue";
import { GanttPlugin, Edit, Toolbar } from "@syncfusion/ej2-vue-gantt";
import { editingResources } from './data-source.js';
Vue.use(GanttPlugin);
export default {
  data: function() {
    return{
      data: [
        {
          TaskID: 1,
          TaskName: 'Project initiation',
          StartDate: new Date('04/02/2019'),
          EndDate: new Date('04/21/2019'),
          subtasks: [
            {TaskID: 2, TaskName: 'Identify site location',
              StartDate: new Date('04/02/2019'), Duration: 0, Progress: 50, resources:
                [1]},

```



```

        {TaskID: 3, TaskName: 'Perform soil test',
StartDate: new Date('04/02/2019'), Duration: 4, Predecessor: '2', Progress:
50, resources: [2, 3, 5]},
        {TaskID: 4, TaskName: 'Soil test approval',
StartDate: new Date('04/02/2019'), Duration: 0, Predecessor: '3', Progress:
50 },
    ],
    {
        TaskID: 5,
        TaskName: 'Project estimation',
        StartDate: new Date('04/02/2019'),
        EndDate: new Date('04/21/2019'),
        subtasks: [
            {TaskID: 6, TaskName: 'Develop floor plan for
estimation', StartDate: new Date('04/04/2019'), Duration: 3, Predecessor:
'4', Progress: 50, resources: [4]},
            {TaskID: 7, TaskName: 'List materials', StartDate:
new Date('04/04/2019'), Duration: 3, Predecessor: '6', resources: [4,
8], Progress: 50},
            {TaskID: 8, TaskName: 'Estimation approval',
StartDate: new Date('04/04/2019'), Duration: 0, Predecessor: '7', resources:
[12, 5]}
        ]
    },
    ],
    taskFields: {
        id: 'TaskID',
        name: 'TaskName',
        startDate: 'StartDate',
        endDate: 'EndDate',
        duration: 'Duration',
        progress: 'Progress',
        dependency: 'Predecessor',
        child: 'subtasks',
        notes: 'info',
        resourceInfo: 'resources'
    },
    editDialogFields: [
        { type: 'General', headerText: 'General' },
        { type: 'Dependency' },
        { type: 'Resources' },
        { type: 'Notes' }
    ],
    addDialogFields : [
        { type: 'General', headerText: 'General' },
        { type: 'Dependency' }
    ],
    height: '450px',
    resourceFields: {
        id: 'resourceId',
        name: 'resourceName',
    },
    resources: editingResources,
    editSettings: {
        allowAdding: true,
        allowEditing: true,
    },

```

```

        mode: 'Dialog',
        allowTaskbarEditing: true
    },
    toolbar: ['Add'],
    labelSettings: {
        leftLabel: 'TaskName',
        rightLabel: 'resources'
    },
    };
},
provide: {
    gantt: [ Edit, Toolbar ]
}
};
</script>

```

{% previewsample "page.domainurl/code-snippet/gantt/managing-tasks-cs8" %}

#### Limiting data fields in general tab

In the Gantt dialog, you can make only specific data source fields visible for editing by using the [addDialogFields](#) and [editDialogFields](#) properties. The data fields are defined with [type](#) and [fields](#) properties.

#### APP.VUE

```

<template>
  <div>
    <ejs-gantt ref='gantt' id="GanttContainer" :dataSource="data"
    :taskFields= "taskFields" :height= "height"
    :editDialogFields="editDialogFields" :editSettings="editSettings"
    :toolbar="toolbar" :resourceFields= "resourceFields" :resources= "resources"
    :labelSettings="labelSettings"></ejs-gantt>
  </div>
</template>
<script>
import Vue from "vue";
import { GanttPlugin, Edit, Toolbar } from "@syncfusion/ej2-vue-gantt";
import { editingResources } from './data-source.js';
Vue.use(GanttPlugin);
export default {
  data: function() {
    return{
      data: [
        {
          TaskID: 1,
          TaskName: 'Project initiation',
          StartDate: new Date('04/02/2019'),
          EndDate: new Date('04/21/2019'),
          subtasks: [
            {TaskID: 2, TaskName: 'Identify site location',
              StartDate: new Date('04/02/2019'), Duration: 0, Progress: 50, resources:
                [1]},
            {TaskID: 3, TaskName: 'Perform soil test',
              StartDate: new Date('04/02/2019'), Duration: 4, Predecessor: '2', Progress:
                50, resources: [2, 3, 5]},

```

```

        {TaskID: 4, TaskName: 'Soil test approval',
StartDate: new Date('04/02/2019'), Duration: 0, Predecessor: '3', Progress:
50 },
    ],
    },
    {
        TaskID: 5,
        TaskName: 'Project estimation',
        StartDate: new Date('04/02/2019'),
        EndDate: new Date('04/21/2019'),
        subtasks: [
            {TaskID: 6, TaskName: 'Develop floor plan for
estimation', StartDate: new Date('04/04/2019'),Duration: 3, Predecessor:
'4', Progress: 50, resources: [4]},
            {TaskID: 7, TaskName: 'List materials', StartDate:
new Date('04/04/2019'),Duration: 3, Predecessor: '6', resources: [4,
8],Progress: 50},
            {TaskID: 8, TaskName: 'Estimation approval',
StartDate: new Date('04/04/2019'),Duration: 0, Predecessor: '7', resources:
[12, 5]}
        ]
    }
],
taskFields: {
    id: 'TaskID',
    name: 'TaskName',
    startDate: 'StartDate',
    endDate: 'EndDate',
    duration: 'Duration',
    progress: 'Progress',
    dependency: 'Predecessor',
    child: 'subtasks',
    notes: 'info',
    resourceInfo: 'resources'
},
editDialogFields: [
    { type: 'General', headerText: 'General', fields: ['TaskID',
'TaskName', 'isParent'] },
    { type: 'Dependency' },
    { type: 'Resources' }
],
height: '450px',
resourceFields: {
    id: 'resourceId',
    name: 'resourceName',
},
resources: editingResources,
editSettings: {
    allowAdding: true,
    allowEditing: true,
    mode: 'Dialog',
    allowTaskbarEditing: true
},
toolbar: ['Add'],
labelSettings: {
    leftLabel: 'TaskName',
    rightLabel: 'resources'
}

```

```

    },
    };
  },
  provide: {
    gantt: [ Edit, Toolbar ]
  }
};
</script>

```

{% previewsample "page.domainurl/code-snippet/gantt/managing-tasks-cs9" %}

**Note:** You can also define the custom fields in the add/edit dialog General tab using the [fields](#) property.

### Task dependencies

In the Gantt component, you can update the dependencies between the tasks and link the tasks interactively. The task dependencies can be mapped from the data source using the [dependency](#) property.

You can update the task dependencies using the following ways:

- Mouse interactions: Using connector points in the taskbar, you can perform drag and drop action to create task dependency links.
- Edit dialog: Create or remove the task dependencies using the **Dependency** tab in the edit dialog.
- Cell editing: Create or remove the task links using cell editing.

The following code example demonstrates how to enable task dependency editing in the Gantt chart using the [editSettings](#) property.

### APP.VUE

```

<template>
  <div>
    <ejs-gantt ref='gantt' id="GanttContainer" :dataSource="data"
    :taskFields = "taskFields" :height = "height" :columns = "columns"
    :editSettings= "editSettings"></ejs-gantt>
  </div>
</template>
<script>
import Vue from "vue";
import { GanttPlugin, Edit } from "@syncfusion/ej2-vue-gantt";
Vue.use(GanttPlugin);
export default {
  data: function() {
    return{
      data: [
        {
          TaskID: 1,
          TaskName: 'Project Initiation',
          StartDate: new Date('04/02/2019'),
          EndDate: new Date('04/21/2019'),
          isParent: true,
          subtasks: [
            { TaskID: 2, TaskName: 'Identify Site location', StartDate: new
Date('04/02/2019'), Duration: 0, Progress: 50 },

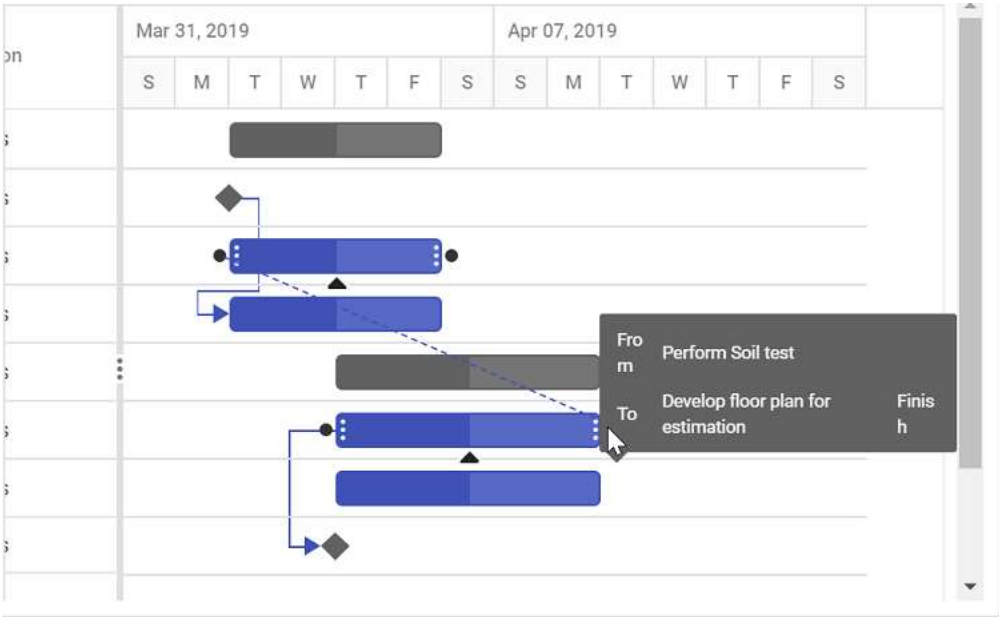
```

```

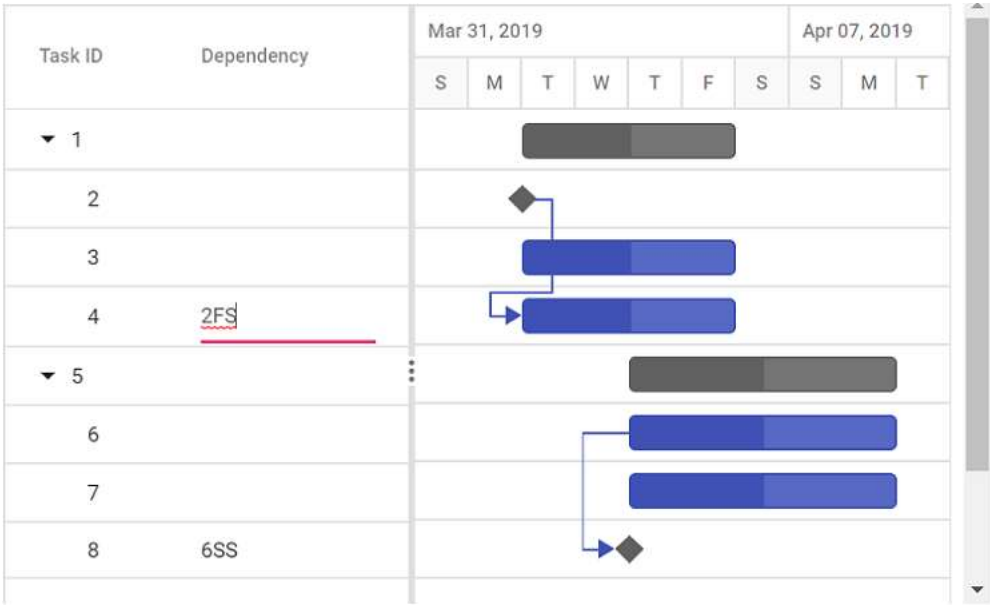
        { TaskID: 3, TaskName: 'Perform Soil test', StartDate: new
Date('04/02/2019'), Duration: 4, Progress: 50 },
        { TaskID: 4, TaskName: 'Soil test approval', StartDate: new
Date('04/02/2019'), Duration: 4, Predecessor: "2FS", Progress: 50 },
    ]
},
{
    TaskID: 5,
    TaskName: 'Project Estimation',
    StartDate: new Date('04/02/2019'),
    EndDate: new Date('04/21/2019'),
    isParent: true,
    subtasks: [
        { TaskID: 6, TaskName: 'Develop floor plan for estimation',
StartDate: new Date('04/04/2019'), Duration: 3, Progress: 50 },
        { TaskID: 7, TaskName: 'List materials', StartDate: new
Date('04/04/2019'), Duration: 3, Progress: 50 },
        { TaskID: 8, TaskName: 'Estimation approval', StartDate: new
Date('04/04/2019'), Duration: 0, Predecessor: "6SS", Progress: 50 }
    ]
},
],
    height: '450px',
    taskFields: {
        id: 'TaskID',
        name: 'TaskName',
        startDate: 'StartDate',
        endDate: 'EndDate',
        duration: 'Duration',
        progress: 'Progress',
        dependency: 'Predecessor',
        child: 'subtasks'
    },
    columns: [
        { field: 'TaskID', headerText: 'Task ID', textAlign: 'Left',
width: '100' },
        { field: 'TaskName', headerText: 'Task Name', width: '250'
},
        { field: 'StartDate', headerText: 'Start Date', width: '150'
},
        { field: 'Duration', headerText: 'Duration', width: '150' },
        { field: 'Progress', headerText: 'Progress', width: '150' },
    ],
    editSettings: {
        allowTaskbarEditing: true,
        allowEditing: true,
        mode: 'Dialog'
    },
};
},
provide: {
    gantt: [ Edit ]
}
};
</script>

```

```
{% previewsample "page.domainurl/code-snippet/gantt/managing-tasks-cs17" %}
```



Updating with mouse interaction action



Updating with cell Edit

Task Information

GENERAL

DEPENDENCY

+

Add

Delete

ID	Name	Type	Offset
2	2-Identify Site location	Finish-Start	0 days

SAVE

CANCEL

### Updating with Dialog

**Note:** When the edit mode is set to **Auto**, on performing double-click action on TreeGrid side, the cells will be changed to editable mode and on performing double-click action on chart side, the edit dialog will appear for editing the task details.

#### Update task values using method

The tasks value can be dynamically updated by using the [updateRecordById](#) method. You can call this method on any custom action. The following code example shows how to use this method to update a task.

### APP.VUE

```

<template>
  <div>
    <ejs-button id="updateRecord" cssClass="e-info" v-
on:click.native="update">update</ejs-button>
    <br><br>
    <ejs-gantt ref='gantt' id="GanttContainer" :dataSource="data"
:taskFields = "taskFields" :height = "height" :editSettings=
"editSettings"></ejs-gantt>
  </div>
</template>
<script>
import Vue from "vue";
import { GanttPlugin, Edit } from "@syncfusion/ej2-vue-gantt";
import { ButtonPlugin } from "@syncfusion/ej2-vue-buttons";
Vue.use(GanttPlugin);
Vue.use(ButtonPlugin);
export default {
  data: function() {

```

```

return{
  data: [
    {
      TaskID: 1,
      TaskName: 'Project Initiation',
      StartDate: new Date('04/02/2019'),
      EndDate: new Date('04/21/2019'),
      subtasks: [
        { TaskID: 2, TaskName: 'Identify Site location',
StartDate: new Date('04/02/2019'), Duration: 4, Progress: 50 },
        { TaskID: 3, TaskName: 'Perform Soil test', StartDate:
new Date('04/02/2019'), Duration: 4, Progress: 50 },
        { TaskID: 4, TaskName: 'Soil test approval', StartDate:
new Date('04/02/2019'), Duration: 4, Progress: 50 },
      ]
    },
    {
      TaskID: 5,
      TaskName: 'Project Estimation',
      StartDate: new Date('04/02/2019'),
      EndDate: new Date('04/21/2019'),
      subtasks: [
        { TaskID: 6, TaskName: 'Develop floor plan for
estimation', StartDate: new Date('04/04/2019'), Duration: 3, Progress: 50 },
        { TaskID: 7, TaskName: 'List materials', StartDate: new
Date('04/04/2019'), Duration: 3, Progress: 50 },
        { TaskID: 8, TaskName: 'Estimation approval', StartDate:
new Date('04/04/2019'), Duration: 3, Progress: 50 }
      ]
    },
  ],
  height: '450px',
  taskFields: {
    id: 'TaskID',
    name: 'TaskName',
    startDate: 'StartDate',
    endDate: 'EndDate',
    duration: 'Duration',
    progress: 'Progress',
    dependency: 'Predecessor',
    child: 'subtasks'
  },
  editSettings: {
    allowEditing: true
  },
};

},
provide: {
  gantt: [ Edit ]
},
methods: {
  update: function(e){
    var ganttObj =
document.getElementById('GanttContainer').ej2_instances[0];
    var data = {
      TaskID: 3,
      TaskName: 'Updated by index value',

```



```

        StartDate: new Date('04/03/2019'),
        Duration: 4,
        Progress: 50,
    };
    ganttObj.updateRecordByID(data);
},
}
};
</script>

```

{% previewsample "page.domainurl/code-snippet/gantt/managing-tasks-cs18" %}

NOTE: Using the [updateRecordById](#) method, you cannot update the task ID value.

### Deleting tasks in Vue Gantt component

#### Deleting tasks

A task delete option in the Gantt component can be enabled by enabling the [editSettings.allowDeleting](#) property. Tasks can be deleted by clicking the delete toolbar item or using the `deleteRow` method. You can call this method dynamically on any custom actions like button click. The following code example shows how to enable the delete option in the Gantt component.

#### APP.VUE

```

<template>
  <div>
    <ejs-button id="deleteRecord" cssClass="e-info" v-
on:click.native="update">delete</ejs-button>
    <br><br>
    <ejs-gantt ref='gantt' id="GanttContainer" :dataSource="data"
:taskFields = "taskFields" :height = "height" :columns = "columns"
:editSettings= "editSettings"></ejs-gantt>
  </div>
</template>
<script>
import Vue from "vue";
import { GanttPlugin, Edit, Selection } from "@syncfusion/ej2-vue-gantt";
import { ButtonPlugin } from "@syncfusion/ej2-vue-buttons";
Vue.use(GanttPlugin);
Vue.use(ButtonPlugin);
export default {
  data: function() {
    return{
      data: [
        {
          TaskID: 1,
          TaskName: 'Project Initiation',
          StartDate: new Date('04/02/2019'),
          EndDate: new Date('04/21/2019'),
          subtasks: [
            { TaskID: 2, TaskName: 'Identify Site location',
StartDate: new Date('04/02/2019'), Duration: 4, Progress: 50 },
            { TaskID: 3, TaskName: 'Perform Soil test', StartDate:
new Date('04/02/2019'), Duration: 4, Progress: 50 },
            { TaskID: 4, TaskName: 'Soil test approval', StartDate:
new Date('04/02/2019'), Duration: 4, Progress: 50 },
          ]
        }
      ]
    }
  }
}

```

```

    },
    {
        TaskID: 5,
        TaskName: 'Project Estimation',
        StartDate: new Date('04/02/2019'),
        EndDate: new Date('04/21/2019'),
        subtasks: [
            { TaskID: 6, TaskName: 'Develop floor plan for
estimation', StartDate: new Date('04/04/2019'), Duration: 3, Progress: 50 },
            { TaskID: 7, TaskName: 'List materials', StartDate: new
Date('04/04/2019'), Duration: 3, Progress: 50 },
            { TaskID: 8, TaskName: 'Estimation approval', StartDate:
new Date('04/04/2019'), Duration: 3, Progress: 50 }
        ]
    },
],
height: '450px',
taskFields: {
    id: 'TaskID',
    name: 'TaskName',
    startDate: 'StartDate',
    endDate: 'EndDate',
    duration: 'Duration',
    progress: 'Progress',
    dependency: 'Predecessor',
    child: 'subtasks'
},
columns: [
    { field: 'TaskID', headerText: 'Task ID', textAlign: 'Left',
width: '100' },
    { field: 'TaskName', headerText: 'Task Name', width: '250'
},
    { field: 'StartDate', headerText: 'Start Date', width: '150'
},
    { field: 'Duration', headerText: 'Duration', width: '150' },
    { field: 'Progress', headerText: 'Progress', width: '150' },
],
editSettings: {
    allowDeleting: true,
    showDeleteConfirmDialog: true
},
};
},
provide: {
    gantt: [ Edit, Selection ]
},
methods: {
    update: function(e){
        var ganttObj =
document.getElementById('GanttContainer').ej2_instances[0];
        ganttObj.editSettings.showDeleteConfirmDialog = true ;
        ganttObj.editModule.deleteRow();
    },
}
};
</script>

```

```
{% previewsample "page.domainurl/code-snippet/gantt/managing-tasks-cs4" %}
```

NOTE: You should select any one of the rows in the Gantt component to perform task delete action.

You should set the [editSettings.allowDeleting](#) value to `true` to delete the record dynamically.

#### *Delete confirmation message*

Delete confirmation message is used to get the confirmation from users before deleting a task. This confirmation message can be enabled by setting the [editSettings.showDeleteConfirmDialog](#) property to `true`.

The following code snippet explains how to enable the delete confirmation message in Gantt.

#### **APP.VUE**

```
<template>
  <div>
    <ejs-gantt ref='gantt' id="GanttContainer" :dataSource="data"
    :taskFields = "taskFields" :height = "height" :toolbar="toolbar" :columns =
    "columns" :editSettings= "editSettings"></ejs-gantt>
  </div>
</template>
<script>
import Vue from "vue";
import { GanttPlugin, Edit, Selection, Toolbar } from "@syncfusion/ej2-vue-gantt";
Vue.use(GanttPlugin);
export default {
  data: function() {
    return {
      data: [
        {
          TaskID: 1,
          TaskName: 'Project Initiation',
          StartDate: new Date('04/02/2019'),
          EndDate: new Date('04/21/2019'),
          subtasks: [
            { TaskID: 2, TaskName: 'Identify Site location',
              StartDate: new Date('04/02/2019'), Duration: 4, Progress: 50 },
            { TaskID: 3, TaskName: 'Perform Soil test', StartDate:
              new Date('04/02/2019'), Duration: 4, Progress: 50 },
            { TaskID: 4, TaskName: 'Soil test approval', StartDate:
              new Date('04/02/2019'), Duration: 4, Progress: 50 },
          ]
        },
        {
          TaskID: 5,
          TaskName: 'Project Estimation',
          StartDate: new Date('04/02/2019'),
          EndDate: new Date('04/21/2019'),
          subtasks: [
            { TaskID: 6, TaskName: 'Develop floor plan for
              estimation', StartDate: new Date('04/04/2019'), Duration: 3, Progress: 50 },
            { TaskID: 7, TaskName: 'List materials', StartDate: new
              Date('04/04/2019'), Duration: 3, Progress: 50 },
          ]
        }
      ]
    }
  }
}
```

```

        { TaskID: 8, TaskName: 'Estimation approval', StartDate:
new Date('04/04/2019'), Duration: 3, Progress: 50 }
    ],
    },
    ],
    height: '450px',
    taskFields: {
        id: 'TaskID',
        name: 'TaskName',
        startDate: 'StartDate',
        endDate: 'EndDate',
        duration: 'Duration',
        progress: 'Progress',
        dependency: 'Predecessor',
        child: 'subtasks'
    },
    columns: [
        { field: 'TaskID', headerText: 'Task ID', textAlign: 'Left',
width: '100' },
        { field: 'TaskName', headerText: 'Task Name', width: '250'
},
        { field: 'StartDate', headerText: 'Start Date', width: '150'
},
        { field: 'Duration', headerText: 'Duration', width: '150' },
        { field: 'Progress', headerText: 'Progress', width: '150' },
    ],
    toolbar: ['Delete'],
    editSettings: {
        allowDeleting: true,
        showDeleteConfirmDialog: true
    },
    };
    },
    provide: {
        gantt: [ Edit, Selection, Toolbar ]
    }
    };
</script>

```

{% previewsample "page.domainurl/code-snippet/gantt/managing-tasks-cs5" %}

### Task bar editing in Vue Gantt component

#### Taskbar editing

Modify the task details through user interaction such as resizing and dragging the taskbar by enabling the [allowTaskbarEditing](#) property.

#### APP.VUE

```

<template>
  <div>
    <ejs-gantt ref='gantt' id="GanttContainer" :dataSource="data"
:taskFields = "taskFields" :height = "height" :columns = "columns"
:editSettings= "editSettings"></ejs-gantt>
  </div>
</template>
<script>

```

```

import Vue from "vue";
import { GanttPlugin, Edit } from "@syncfusion/ej2-vue-gantt";
Vue.use(GanttPlugin);
export default {
  data: function() {
    return {
      data: [
        {
          TaskID: 1,
          TaskName: 'Project Initiation',
          StartDate: new Date('04/02/2019'),
          EndDate: new Date('04/21/2019'),
          subtasks: [
            { TaskID: 2, TaskName: 'Identify Site location',
StartDate: new Date('04/02/2019'), Duration: 4, Progress: 50 },
            { TaskID: 3, TaskName: 'Perform Soil test', StartDate:
new Date('04/02/2019'), Duration: 4, Progress: 50 },
            { TaskID: 4, TaskName: 'Soil test approval', StartDate:
new Date('04/02/2019'), Duration: 4, Progress: 50 },
          ]
        },
        {
          TaskID: 5,
          TaskName: 'Project Estimation',
          StartDate: new Date('04/02/2019'),
          EndDate: new Date('04/21/2019'),
          subtasks: [
            { TaskID: 6, TaskName: 'Develop floor plan for
estimation', StartDate: new Date('04/04/2019'), Duration: 3, Progress: 50 },
            { TaskID: 7, TaskName: 'List materials', StartDate: new
Date('04/04/2019'), Duration: 3, Progress: 50 },
            { TaskID: 8, TaskName: 'Estimation approval', StartDate:
new Date('04/04/2019'), Duration: 3, Progress: 50 }
          ]
        }
      ],
      height: '450px',
      taskFields: {
        id: 'TaskID',
        name: 'TaskName',
        startDate: 'StartDate',
        endDate: 'EndDate',
        duration: 'Duration',
        progress: 'Progress',
        child: 'subtasks'
      },
      columns: [
        { field: 'TaskID', headerText: 'Task ID', textAlign: 'Left',
width: '100' },
        { field: 'TaskName', headerText: 'Task Name', width: '250' },
        { field: 'StartDate', headerText: 'Start Date', width: '150' },
        { field: 'Duration', headerText: 'Duration', width: '150' },
        { field: 'Progress', headerText: 'Progress', width: '150' },
      ],
      editSettings: {

```

```

        allowTaskbarEditing:true
    },
    };
    },
    provide: {
        gantt: [ Edit ]
    }
    };
</script>

```

{% previewsample "page.domainurl/code-snippet/gantt/managing-tasks-cs15" %}

### Prevent editing for specific tasks

On taskbar edit action, the [taskbarEditing](#) event will be triggered. You can prevent the taskbar from editing using the [taskbarEditing](#) event. This can be done by setting cancel property of [taskbarEditing](#) event argument to true. And we can hide the taskbar editing indicators like taskbar resizer, progress resizer and connector points by using [queryTaskbarInfo](#) event. The following code example shows how to achieve this.

### APP.VUE

```

<template>
  <div>
    <ejs-gantt ref='gantt' id="GanttContainer" :dataSource="data"
      :taskFields = "taskFields" :height = "height"
      :taskbarEditing="taskbarEditing" :columns = "columns" :editSettings=
      "editSettings" :queryTaskbarInfo = "queryTaskbarInfo"></ejs-gantt>
    </div>
  </template>
<script>
import Vue from "vue";
import { GanttPlugin, Edit } from "@syncfusion/ej2-vue-gantt";
Vue.use(GanttPlugin);
export default {
  data: function() {
    return{
      data: [
        {
          TaskID: 1,
          TaskName: 'Project Initiation',
          StartDate: new Date('04/02/2019'),
          EndDate: new Date('04/21/2019'),
          subtasks: [
            { TaskID: 2, TaskName: 'Identify Site location',
              StartDate: new Date('04/02/2019'), Duration: 4, Progress: 50 },
            { TaskID: 3, TaskName: 'Perform Soil test', StartDate:
              new Date('04/02/2019'), Duration: 4, Progress: 50 },
            { TaskID: 4, TaskName: 'Soil test approval', StartDate:
              new Date('04/02/2019'), Duration: 4, Progress: 50 },
          ]
        },
        {
          TaskID: 5,
          TaskName: 'Project Estimation',
          StartDate: new Date('04/02/2019'),
          EndDate: new Date('04/21/2019'),

```

```

        subtasks: [
            { TaskID: 6, TaskName: 'Develop floor plan for
estimation', StartDate: new Date('04/04/2019'), Duration: 3, Progress: 50 },
            { TaskID: 7, TaskName: 'List materials', StartDate: new
Date('04/04/2019'), Duration: 3, Progress: 50 },
            { TaskID: 8, TaskName: 'Estimation approval', StartDate:
new Date('04/04/2019'), Duration: 3, Progress: 50 }
        ]
    },
],
height: '450px',
taskFields: {
    id: 'TaskID',
    name: 'TaskName',
    startDate: 'StartDate',
    endDate: 'EndDate',
    duration: 'Duration',
    progress: 'Progress',
    child: 'subtasks'
},
columns: [
    { field: 'TaskID', headerText: 'Task ID', textAlign: 'Left',
width: '100' },
    { field: 'TaskName', headerText: 'Task Name', width: '250'
},
    { field: 'StartDate', headerText: 'Start Date', width: '150'
},
    { field: 'Duration', headerText: 'Duration', width: '150' },
    { field: 'Progress', headerText: 'Progress', width: '150' },
],
editSettings: {
    allowTaskbarEditing: true
},
taskbarEditing: function (args) {
    if (args.data.TaskID == 4) // We can't edit Task Id 4
        args.cancel = true;
},
queryTaskbarInfo: function(args) {
    if (args.data.TaskID == 6) {
        args.taskbarElement.className += ' e-preventEdit' //
Taskbar editing indicators are disabled
    }
}
};
},
provide: {
    gantt: [ Edit ]
}
};
</script>
<style>
    .e-gantt-chart .e-preventEdit .e-right-resize-gripper,
    .e-gantt-chart .e-preventEdit .e-left-resize-gripper,
    .e-gantt-chart .e-preventEdit .e-progress-resize-gripper,
    .e-gantt-chart .e-preventEdit .e-left-connectorpoint-outer-div,
    .e-gantt-chart .e-preventEdit .e-right-connectorpoint-outer-div {
        display: none;
    }

```

```
}
</style>
```

```
{% previewsample "page.domainurl/code-snippet/gantt/managing-tasks-cs16" %}
```

### In dent and out dent in Vue Gantt component

Indent and Outdent of a task are used to update the level of the task in hierarchical order of the task. It can be performed by enabling the ['editSettings.allowEditing'](#) property.

**Indent** - Selected task can be indented to the level of task to the hierarchical order. It can be performed by using in-built context menu or toolbar items. It can also be invoked by using the `indent` method dynamically on any action like external button click. The following code example shows how to enable indent option in the Gantt chart.

**Outdent** - Selected task can be outdented to the level of task from the hierarchical order. It can be performed by using in-built context menu or toolbar items. It can also be invoked by using the `outdent` method dynamically on any action like external button click. The following code example shows how to enable outdent option in the Gantt chart.

### APP.VUE

```
<template>
  <div>
    <ejs-button id="indent" cssClass="e-info" v-
on:click.native="indent">Indent</ejs-button>
    <ejs-button id="outdent" cssClass="e-info" v-
on:click.native="outdent">Outdent</ejs-button>
    <br>
    <ejs-gantt ref='gantt' id="GanttContainer" :dataSource="data"
:taskFields = "taskFields" :height = "height" :columns = "columns"
:editSettings= "editSettings" :toolbar="toolbar"></ejs-gantt>
  </div>
</template>
<script>
import Vue from "vue";
import { GanttPlugin, Edit, Selection, Toolbar } from "@syncfusion/ej2-vue-
gantt";
import { ButtonPlugin } from "@syncfusion/ej2-vue-buttons";
Vue.use(GanttPlugin);
Vue.use(ButtonPlugin);
export default {
  data: function() {
    return{
      data: [
        {
          TaskID: 1,
          TaskName: 'Project Initiation',
          StartDate: new Date('04/02/2019'),
          EndDate: new Date('04/21/2019'),
          subtasks: [
            { TaskID: 2, TaskName: 'Identify Site location',
StartDate: new Date('04/02/2019'), Duration: 4, Progress: 50 },
            { TaskID: 3, TaskName: 'Perform Soil test', StartDate:
new Date('04/02/2019'), Duration: 4, Progress: 50 },
```



```

        { TaskID: 4, TaskName: 'Soil test approval', StartDate:
new Date('04/02/2019'), Duration: 4, Progress: 50 },
    ],
    },
    {
        TaskID: 5,
        TaskName: 'Project Estimation',
        StartDate: new Date('04/02/2019'),
        EndDate: new Date('04/21/2019'),
        subtasks: [
            { TaskID: 6, TaskName: 'Develop floor plan for
estimation', StartDate: new Date('04/04/2019'), Duration: 3, Progress: 50 },
            { TaskID: 7, TaskName: 'List materials', StartDate: new
Date('04/04/2019'), Duration: 3, Progress: 50 },
            { TaskID: 8, TaskName: 'Estimation approval', StartDate:
new Date('04/04/2019'), Duration: 3, Progress: 50 }
        ]
    },
],
height: '450px',
taskFields: {
    id: 'TaskID',
    name: 'TaskName',
    startDate: 'StartDate',
    endDate: 'EndDate',
    duration: 'Duration',
    progress: 'Progress',
    child: 'subtasks'
},
columns: [
    { field: 'TaskID', headerText: 'Task ID', textAlign: 'Left',
width: '100' },
    { field: 'TaskName', headerText: 'Task Name', width: '250'
},
    { field: 'StartDate', headerText: 'Start Date', width: '150'
},
    { field: 'Duration', headerText: 'Duration', width: '150' },
    { field: 'Progress', headerText: 'Progress', width: '150' },
],
editSettings: {
    allowEditing: true
},
toolbar: ['Indent', 'Outdent'],
};
},
provide: {
    gantt: [ Edit, Selection, Toolbar ]
},
methods: {
    indent: function(e){
        var ganttChart =
document.getElementById('GanttContainer').ej2_instances[0];
        ganttChart.indent();
    },
    outdent: function(e){
        var ganttChart =
document.getElementById('GanttContainer').ej2_instances[0];

```

```

        ganttChart.outdent();
    },
    },
};
</script>

```

{% previewsample "page.domainurl/code-snippet/gantt/managing-tasks-cs10" %}

Splitting and merging tasks in Vue Gantt component

*Splitting task at load time*

To split task at load time, we can define segment details in both hierarchical and self-referential way.

Refer below link for more details.

- [Split task at load time](#)

*Split task dynamically*

The task can be split dynamically, either by using the context menu or dialog.

- **Dialog:** Segments tab is rendered in add/edit dialog, when the [taskFields.segments](#) or [taskFields.segmentId](#) property is mapped. Using this tab, we can split the task based on the original start and end date of a particular task.
- **Context menu:** When the [taskFields.segments](#) or [taskFields.segmentId](#) property is mapped and the [enableContextMenu](#) property is enabled, **Split Task** item will be included in the context menu.

## APP.VUE

```

<template>
  <div>
    <ejs-gantt ref='gantt' id="GanttContainer" :dataSource="data"
    :taskFields = "taskFields" :height = "height" :editSettings= "editSettings"
    :toolbar="toolbar" :enableContextMenu="true"></ejs-gantt>
  </div>
</template>
<script>
import Vue from "vue";
import { GanttPlugin, ContextMenu, Edit, Selection, Toolbar } from
"@syncfusion/ej2-vue-gantt";
Vue.use(GanttPlugin);
export default {
  data: function() {
    return{
      data: [
        {
          TaskID: 1,
          TaskName: 'Project Initiation',
          StartDate: new Date('04/02/2019'),
          EndDate: new Date('04/21/2019'),
          subtasks: [
            { TaskID: 2, TaskName: 'Identify Site location', StartDate:
new Date('04/02/2019'), Duration: 4, Progress: 50,
              Segments: [

```

```

        { StartDate: new Date("04/02/2019"), Duration: 2 },
        { StartDate: new Date("04/04/2019"), Duration: 2 }
    ] },
    { TaskID: 3, TaskName: 'Perform Soil test', StartDate: new
Date('04/02/2019'), Duration: 4, Progress: 50 },
    { TaskID: 4, TaskName: 'Soil test approval', StartDate: new
Date('04/02/2019'), Duration: 4, Progress: 50 },
    ]
},
{
    TaskID: 5,
    TaskName: 'Project Estimation',
    StartDate: new Date('04/02/2019'),
    EndDate: new Date('04/21/2019'),
    subtasks: [
        { TaskID: 6, TaskName: 'Develop floor plan for estimation',
StartDate: new Date('04/04/2019'), Duration: 3, Progress: 50 },
        { TaskID: 7, TaskName: 'List materials', StartDate: new
Date('04/04/2019'), Duration: 3, Progress: 50 },
        { TaskID: 8, TaskName: 'Estimation approval', StartDate: new
Date('04/04/2019'), Duration: 3, Progress: 50 }
    ]
}
],
    height: '450px',
    taskFields: {
        id: 'TaskID',
        name: 'TaskName',
        startDate: 'StartDate',
        endDate: 'EndDate',
        duration: 'Duration',
        progress: 'Progress',
        child: 'subtasks',
        segments: 'Segments'
    },
    editSettings: {
        allowEditing: true,
        allowAdding: true,
        allowDeleting: true,
        allowTaskbarEditing: true
    },
    toolbar: ['Add', 'Edit', 'Delete', 'Cancel', 'ExpandAll',
'CollapseAll', 'Update'],
};
},
provide: {
    gantt: [ ContextMenu, Edit, Selection, Toolbar ]
}
};
</script>

```

{% previewsample "page.domainurl/code-snippet/gantt/managing-tasks-cs14" %}

### Merge tasks

The split tasks can be merged either by using the **Merge Task** item of the Context menu or by using the dialog. We can also merge the tasks, by simply dragging the segments together in the UI.

### Limitations of Split tasks

1. Parent and milestone tasks cannot be split into segments.
2. The task must have a width greater than the timeline unit cell in order to be split.
3. Split task is not supported in the **Resource view**.

### Maintaining data in server in Vue Gantt component

All the modified data in Gantt control can be maintained in the database using RESTful web services.

All the CRUD operations in the gantt are done through DataManager. The DataManager has an option to bind all the CRUD related data in server-side.

In the below section, we have explained how to get the edited data details on the server-side using the **UrlAdaptor**.

### URL Adaptor

In Gantt, we can fetch data from SQL database using **ADO.NET** Entity Data Model and update the changes on CRUD action to the server by using **DataManager** support. To communicate with the remote data we are using **UrlAdaptor** of **DataManager** property to call the server method and get back resultant data in JSON format. We can know more about **UrlAdaptor** from [here](#).

Please refer the [link](#) to create the **ADO.NET** Entity Data Model in Visual studio,

We can define data source for Gantt as instance of **DataManager** using **url** property of **DataManager**. Please Check the below code snippet to assign data source to Gantt.

```
<template>
<div>
<ejs-gantt ref='gantt' :dataSource="data" id="GanttContainer" :taskFields = "taskFields" :height =
"height"></ejs-gantt>
</div>
</template>
<script>
import Vue from "vue";
import { GanttPlugin } from "@syncfusion/ej2-vue-gantt";
import { DataManager, UrlAdaptor } from '@syncfusion/ej2-data';
Vue.use(GanttPlugin);
export default {
data: function() {
return{
data: new DataManager({
url: '/Home/UrlDatasource',
```

```

adaptor: new UrlAdaptor
}},
height: '450px',
taskFields: {
id: 'TaskId',
name: 'TaskName',
startDate: 'StartDate',
progress: 'Progress',
duration: 'Duration',
dependency: 'Predecessor',
child: 'SubTasks'
}
};
},
};
</script>
`ts
GanttDataSourceEntities db = new GanttDataSourceEntities();
public ActionResult UrlDatasource(DataManagerRequest dm)
{
List<GanttData>DataList = db.GanttDatas.ToList();
var count = DataList.Count();
return Json(new { result = DataList, count = count });
}
`

```

We can also do CRUD operations over Gantt data and save the changes to database. By using **BatchUrl** property of **DataManager**, we can communicate with the controller method to update the data source on CRUD operation. In gantt CRUD actions on task are dependent with other tasks. For example on editing the child record on chart side, corresponding parent item also will get affect and predecessor dependency task as well get affected. So in Gantt all the CRUD operations are considered to be batch editing where you will get all the affected records as collection. Please check the below code snippet to assign controller method to this property.

```
<template>
```

```
<div>

<ejs-gantt ref='gantt' :dataSource="data" id="GanttContainer" :taskFields = "taskFields" :height =
"height"></ejs-gantt>

</div>

</template>

<script>
import Vue from "vue";
import { GanttPlugin } from "@syncfusion/ej2-vue-gantt";
import { DataManager, UrlAdaptor } from '@syncfusion/ej2-data';
Vue.use(GanttPlugin);
export default {
  data: function() {
    return{
      data: new DataManager({
        url: '/Home/UrlDatasource',
        adaptor: new UrlAdaptor,
        batchUrl: "Home/BatchSave"
      }),
      height: '450px',
      taskFields: {
        id: 'TaskId',
        name: 'TaskName',
        startDate: 'StartDate',
        progress: 'Progress',
        duration: 'Duration',
        dependency: 'Predecessor',
        child: 'SubTasks'
      }
    };
  },
};
</script>
`
```

```

`ts
public class ICRUDModel<T> where T : class
{
    public object key { get; set; }
    public T value { get; set; }
    public List<T> added { get; set; }
    public List<T> changed { get; set; }
    public List<T> deleted { get; set; }
}

public ActionResult BatchSave([FromBody]ICRUDModel<GanttData> data)
{
    List<GanttData> uAdded = new List<GanttData>();
    List<GanttData> uChanged = new List<GanttData>();
    List<GanttData> uDeleted = new List<GanttData>();
    //...
    return Json(new { addedRecords = uAdded, changedRecords = uChanged, deletedRecords = uDeleted });
}
`

```

This server method will be triggered for all the CRUD operations like adding, editing and deleting actions. We can handle those each operations separately inside this method with corresponding data received in this method argument. Also, when using the `UrlAdaptor`, you need to return the data as JSON from the controller action and the JSON object must contain a property as `result` with `dataSource` as its value and one more property `count` with the `dataSource` total records count as its value.

#### *Insert action*

Using the `added` argument of the `BatchUrl` method we can insert the newly added row to database and return the same to client side. please find the below code example for details.

```

`ts
GanttDataSourceEntities db = new GanttDataSourceEntities();

public ActionResult BatchSave([FromBody]ICRUDModel<GanttData> data)
{
    List<GanttData> uAdded = new List<GanttData>();
    //Performing insert operation
    if (data.added != null && data.added.Count() > 0)
    {
        foreach (var rec in data.added)

```

```

{
    uAdded.Add(this.Create(rec));
}
}

return Json(new { addedRecords = uAdded });
//...
}

public GanttData Create(GanttData value)
{
    db.GanttDatas.Add(value);
    db.SaveChanges();
    return value;
}

```

#### *Editing action*

Using the `changed` argument of the `BatchUrl` method we can update the modified records to database and return the same to client side. please find the below code example for details.

```

`ts

GanttDataSourceEntities db = new GanttDataSourceEntities();

public ActionResult BatchSave([FromBody]ICRUDModel<GanttData> data)
{
    List<GanttData> uChanged = new List<GanttData>();
    //Performing update operation
    if (data.changed != null && data.changed.Count() > 0)
    {
        foreach (var rec in data.changed)
        {
            uChanged.Add(this.Edit(rec));
        }
    }

    return Json(new { changedRecords = uChanged });
    //...
}

public GanttData Edit(GanttData value)

```



```

{
    GanttData result = db.GanttDatas.Where(currentData => currentData.TaskId ==
    value.TaskId).FirstOrDefault();
    if (result != null)
    {
        result.TaskId = value.TaskId;
        result.TaskName = value.TaskName;
        result.StartDate = value.StartDate;
        result.EndDate = value.EndDate;
        result.Duration = value.Duration;
        result.Progress = value.Progress;
        result.Predecessor = value.Predecessor;
        db.SaveChanges();
        return result;
    }
    else
    {
        return null;
    }
}
`ts

```

#### Delete action

Using the `deleted` argument of the `BatchUrl` method we can remove the deleted records from database and return the same to client side. on deleting the record we need to remove its corresponding child records as well if it exist from the data base. please find the below code example for details.

```

`ts
GanttDataSourceEntities db = new GanttDataSourceEntities();
public ActionResult BatchSave([FromBody]ICRUDModel<GanttData> data)
{
    List<GanttData> uDeleted = new List<GanttData>();
    //Performing delete operation
    if (data.deleted != null && data.deleted.Count() > 0)
    {
        foreach (var rec in data.deleted)

```

```

{
    uDeleted.Add(this.Delete(rec.TaskId));
}
}

return Json(new { deletedRecords = uDeleted });
}

public GanttData Delete(string value)
{
    var result = db.GanttDatas.Where(currentData => currentData.TaskId == value).FirstOrDefault();
    db.GanttDatas.Remove(result);
    RemoveChildRecords(value);
    db.SaveChanges();
    return result;
}

public void RemoveChildRecords(string key)
{
    var childList = db.GanttDatas.Where(x => x.ParentId == key).ToList();
    foreach (var item in childList)
    {
        db.GanttDatas.Remove(item);
        RemoveChildRecords(item.TaskId);
    }
}
,

```

## Validation in Vue Gantt control

### Column validation

Column validation validates the editing and adding data and it display errors for invalid fields before saving data. This is effective in both inline and dialog editing.

Gantt uses [Form Validator](#) component for column validation. You can set [validation rules](#) by defining the [columns.validationRules](#). The value cannot be saved unless the validation rule get satisfied.

### APP.VUE

```

<template>
  <div>
    <ejs-gantt ref='gantt' id="GanttContainer" :dataSource="data"
      :taskFields = "taskFields" :height = "height" :toolbar= "toolbar" :columns =
      "columns" :editSettings= "editSettings"></ejs-gantt>
  </div>
</template>

```

```

    </div>
</template>
<script>
import Vue from "vue";
import { GanttPlugin, Edit, Selection, Toolbar } from "@syncfusion/ej2-vue-gantt";
Vue.use(GanttPlugin);
export default {
  data: function() {
    return {
      data: [
        {
          TaskID: 1,
          TaskName: 'Project Initiation',
          StartDate: new Date('04/02/2019'),
          EndDate: new Date('04/21/2019'),
          subtasks: [
            { TaskID: 2, TaskName: 'Identify Site location',
StartDate: new Date('04/02/2019'), Duration: 4, Progress: 50 },
            { TaskID: 3, TaskName: 'Perform Soil test', StartDate:
new Date('04/02/2019'), Duration: 4, Progress: 50 },
            { TaskID: 4, TaskName: 'Soil test approval', StartDate:
new Date('04/02/2019'), Duration: 4, Progress: 50 },
          ]
        },
        {
          TaskID: 5,
          TaskName: 'Project Estimation',
          StartDate: new Date('04/02/2019'),
          EndDate: new Date('04/21/2019'),
          subtasks: [
            { TaskID: 6, TaskName: 'Develop floor plan for
estimation', StartDate: new Date('04/04/2019'), Duration: 3, Progress: 50 },
            { TaskID: 7, TaskName: 'List materials', StartDate: new
Date('04/04/2019'), Duration: 3, Progress: 50 },
            { TaskID: 8, TaskName: 'Estimation approval', StartDate:
new Date('04/04/2019'), Duration: 3, Progress: 50 }
          ]
        },
      ],
      height: '450px',
      taskFields: {
        id: 'TaskID',
        name: 'TaskName',
        startDate: 'StartDate',
        endDate: 'EndDate',
        duration: 'Duration',
        progress: 'Progress',
        child: 'subtasks'
      },
      toolbar: ['Add', 'Edit', 'Update', 'Delete', 'Cancel'],
      columns: [
        { field: 'TaskID', headerText: 'Task ID', textAlign: 'Left',
width: '100' },
        { field: 'TaskName', headerText: 'Task Name', width: '250',
validationRules: { required: true } },
      ],
    }
  },

```

```

        { field: 'StartDate', editType: 'datetimepickeredit', edit: {
params: { format: 'M/d/y hh:mm a' } },
        format: { format: 'M/d/y hh:mm a', type: 'dateTime' },
validationRules: { required: true, date: true } },
        { field: 'Duration', headerText: 'Duration', width: '150',
validationRules: { required: true } },
        { field: 'Progress', headerText: 'Progress', width: '150',
validationRules: { required: true } },
    ],
    editSettings: {
        allowAdding: true,
        allowEditing: true,
        allowDeleting: true,
        allowTaskbarEditing: true,
        showDeleteConfirmDialog: true
    },
    };
},
provide: {
    gantt: [ Edit, Selection, Toolbar ]
}
};
</script>

```

{% previewsample "page.domainurl/code-snippet/gantt/managing-tasks-cs22" %}

#### Custom validation

You can define your own custom validation rules for the specific columns by using callback function to it's [validation rule](#).

In the below demo, custom validation applied for **TaskName** column.

#### APP.VUE

```

<template>
  <div>
    <ejs-gantt ref='gantt' id="GanttContainer" :dataSource="data"
:taskFields = "taskFields" :height = "height" :toolbar= "toolbar" :columns =
"columns" :editSettings= "editSettings"></ejs-gantt>
  </div>
</template>
<script>
import Vue from "vue";
import { GanttPlugin, Edit, Selection, Toolbar } from "@syncfusion/ej2-vue-gantt";
Vue.use(GanttPlugin);
export default {
  data: function() {
    return {
      data: [
        {
          TaskID: 1,
          TaskName: 'Project Initiation',
          StartDate: new Date('04/02/2019'),
          EndDate: new Date('04/21/2019'),
          subtasks: [

```

```

        { TaskID: 2, TaskName: 'Identify Site location',
StartDate: new Date('04/02/2019'), Duration: 4, Progress: 50 },
        { TaskID: 3, TaskName: 'Perform Soil test', StartDate:
new Date('04/02/2019'), Duration: 4, Progress: 50 },
        { TaskID: 4, TaskName: 'Soil test approval', StartDate:
new Date('04/02/2019'), Duration: 4, Progress: 50 },
    ]
},
{
    TaskID: 5,
    TaskName: 'Project Estimation',
    StartDate: new Date('04/02/2019'),
    EndDate: new Date('04/21/2019'),
    subtasks: [
        { TaskID: 6, TaskName: 'Develop floor plan for
estimation', StartDate: new Date('04/04/2019'), Duration: 3, Progress: 50 },
        { TaskID: 7, TaskName: 'List materials', StartDate: new
Date('04/04/2019'), Duration: 3, Progress: 50 },
        { TaskID: 8, TaskName: 'Estimation approval', StartDate:
new Date('04/04/2019'), Duration: 3, Progress: 50 }
    ]
},
],
height: '450px',
taskFields: {
    id: 'TaskID',
    name: 'TaskName',
    startDate: 'StartDate',
    endDate: 'EndDate',
    duration: 'Duration',
    progress: 'Progress',
    child: 'subtasks'
},
columns: [
    { field: 'TaskID', headerText: 'Task ID', textAlign: 'Left',
width: '100' },
    { field: 'TaskName', headerText: 'Task Name', width: '250',
validationRules: { required: true, minLength: [(args) => {return
args['value'].length <= 8;}, 'Value should be within 8 letters'] } },
    { field: 'StartDate', editType: 'datetimepickeredit', edit: {
params: { format: 'M/d/y hh:mm a' } },
format: { format: 'M/d/y hh:mm a', type: 'dateTime' },
validationRules: { required: true, date: true } },
    { field: 'Duration', headerText: 'Duration', width: '150',
validationRules: { required: true } },
    { field: 'Progress', headerText: 'Progress', width: '150',
validationRules: { required: true } },
],
toolbar: ['Add', 'Edit', 'Update', 'Delete', 'Cancel'],
editSettings: {
    allowAdding: true,
    allowEditing: true,
    allowDeleting: true,
    allowTaskbarEditing: true,
    showDeleteConfirmDialog: true
},
};

```

```

    },
    provide: {
      gantt: [ Edit, Selection, Toolbar ]
    }
  };
</script>

```

{% previewsample "page.domainurl/code-snippet/gantt/managing-tasks-cs23" %}

#### Dependency and resource grid validation

Validation rules can also be implemented for the dependency and resource grid in the add or edit dialog by employing the event [actionBegin](#).

Within the actionBegin event, validationRules can be configured for columns in the grid of the dependency and resource tabs using the requestType `beforeOpenEditDialog` or `beforeOpenAddDialog`.

#### APP.VUE

```

<template>
  <div>
    <ejs-gantt ref='gantt' id="GanttContainer" :dataSource="data"
      :taskFields= "taskFields" :height= "height" :toolbar= "toolbar"
      :resourceFields= "resourceFields"
        :resources= "resources" :actionBegin= "actionBegin" :columns =
      "columns" :editSettings= "editSettings"></ejs-gantt>
    </div>
  </template>
<script>
import Vue from "vue";
import { GanttPlugin, Edit, Selection, Toolbar } from "@syncfusion/ej2-vue-gantt";
Vue.use(GanttPlugin);
export default {
  data: function() {
    return {
      data: [
        {
          TaskID: 1,
          TaskName: 'Project Initiation',
          StartDate: new Date('04/02/2019'),
          EndDate: new Date('04/21/2019'),
          subtasks: [
            { TaskID: 2, TaskName: 'Identify Site location',
              StartDate: new Date('04/02/2019'), Duration: 4, Progress: 50 },
            { TaskID: 3, TaskName: 'Perform Soil test', StartDate:
              new Date('04/02/2019'), Duration: 4, Progress: 50 },
            { TaskID: 4, TaskName: 'Soil test approval', StartDate:
              new Date('04/02/2019'), Duration: 4, Progress: 50 },
          ]
        },
        {
          TaskID: 5,
          TaskName: 'Project Estimation',
          StartDate: new Date('04/02/2019'),
          EndDate: new Date('04/21/2019'),

```

```

        subtasks: [
            { TaskID: 6, TaskName: 'Develop floor plan for
estimation', StartDate: new Date('04/04/2019'), Duration: 3, Progress: 50 },
            { TaskID: 7, TaskName: 'List materials', StartDate: new
Date('04/04/2019'), Duration: 3, Progress: 50 },
            { TaskID: 8, TaskName: 'Estimation approval', StartDate:
new Date('04/04/2019'), Duration: 3, Progress: 50 }
        ]
    },
],
height: '450px',
taskFields: {
    id: 'TaskID',
    name: 'TaskName',
    startDate: 'StartDate',
    endDate: 'EndDate',
    duration: 'Duration',
    dependency: 'Predecessor',
    progress: 'Progress',
    child: 'subtasks',
    resourceInfo: 'resources'
},
resourceFields: {
    id: 'resourceId',
    name: 'resourceName'
},
resources: [
    { resourceId: 1, resourceName: 'Martin Tamer' },
    { resourceId: 2, resourceName: 'Rose Fuller' },
    { resourceId: 3, resourceName: 'Margaret Buchanan' },
    { resourceId: 4, resourceName: 'Fuller King' },
    { resourceId: 5, resourceName: 'Davolio Fuller' },
    { resourceId: 6, resourceName: 'Van Jack' },
    { resourceId: 7, resourceName: 'Fuller Buchanan' },
    { resourceId: 8, resourceName: 'Jack Davolio' },
    { resourceId: 9, resourceName: 'Tamer Vinet' },
    { resourceId: 10, resourceName: 'Vinet Fuller' },
    { resourceId: 11, resourceName: 'Bergs Anton' },
    { resourceId: 12, resourceName: 'Construction Supervisor' }
],
toolbar: ['Add', 'Edit', 'Update', 'Delete', 'Cancel'],
columns: [
    { field: 'TaskID', headerText: 'Task ID', textAlign: 'Left',
width: '100' },
    { field: 'TaskName', headerText: 'Task Name', width: '250',
validationRules: { required: true } },
    { field: 'StartDate', editType: 'datetimepickeredit', edit: {
params: { format: 'M/d/y hh:mm a' } },
format: { format: 'M/d/y hh:mm a', type: 'dateTime' },
validationRules: { required: true, date: true } },
    { field: 'Duration', headerText: 'Duration', width: '150',
validationRules: { required: true } },
    { field: 'Progress', headerText: 'Progress', width: '150',
validationRules: { required: true } },
],
editSettings: {
    allowAdding: true,

```

```

        allowEditing: true,
        allowDeleting: true,
        allowTaskbarEditing: true,
        showDeleteConfirmDialog: true
    },
    },
    },
    provide: {
        gantt: [ Edit, Selection, Toolbar ]
    },
    methods: {
        actionBegin(args): void {
            if (args.requestType == "beforeOpenEditDialog" || args.requestType ==
"beforeOpenAddDialog") {
                args.Dependency.columns[3].validationRules = { required: true }
                args.Resources.columns[2].allowEditing = true
                args.Resources.columns[2].validationRules = { required: true }
            }
        }
    }
};
</script>

```

{% previewsample "page.domainurl/code-snippet/gantt/managing-tasks-cs24" %}

### Scrolling in Vue Gantt component

The scrollbar will be displayed in the gantt when content exceeds the element **width** or **height**. The vertical and horizontal scrollbars will be displayed based on the following criteria:

- The vertical scrollbar appears when the total height of rows present in the gantt exceeds its element height.
- The horizontal scrollbar appears when the sum of columns width exceeds the grid pane size.
- The [height](#) and [width](#) are used to set the gantt height and width, respectively.

The default value for **height** and **width** is **auto**.

### Set width and height

We can even set pixel values to width and height of gantt container using [width](#) and [height](#) properties.

### APP.VUE

```

<template>
  <div>
    <ejs-gantt ref='gantt' id="GanttContainer" :dataSource="data"
:taskFields = "taskFields" :height = "height" :width = "width"
:editSettings="editSettings"></ejs-gantt>
  </div>
</template>
<script>
import Vue from "vue";
import { GanttPlugin, Edit } from "@syncfusion/ej2-vue-gantt";
import { projectNewData } from './data-source.js';
Vue.use(GanttPlugin);
export default {

```



```

data: function() {
  return{
    data: projectNewData,
    height: '350px',
    width: '600px',
    taskFields: {
      id: 'TaskID',
      name: 'TaskName',
      startDate: 'StartDate',
      duration: 'Duration',
      progress: 'Progress',
      child: 'subtasks'
    },
    editSettings: {
      allowAdding: true,
      allowEditing: true,
      allowDeleting: true,
      allowTaskbarEditing: true,
      showDeleteConfirmDialog: true
    },
  };
},
provide: {
  gantt: [Edit]
}
};
</script>

```

{% previewsample "page.domainurl/code-snippet/gantt/scrolling-cs1" %}

### Responsive with the parent container

Specify the [width](#) and [height](#) as **100%** to make the gantt element fill its parent container.

Setting the **height** to **100%** requires the gantt parent element to have explicit height. Also, the component will be responsive when the parent container is resized.

### APP.VUE

```

<template>
  <div>
    <p class="e-text"> The parent container can be resizable by dragging the
    bottom-right corner.</p>
    <div class='e-ganttresize'>
      <ejs-gantt ref='gantt' id="GanttContainer" :dataSource="data"
      :taskFields = "taskFields" :height = "height" :width = "width"
      :editSettings="editSettings"></ejs-gantt>
    </div>
  </div>
</template>
<script>
import Vue from "vue";
import { GanttPlugin, Edit } from "@syncfusion/ej2-vue-gantt";
import { projectNewData } from './data-source.js';
Vue.use(GanttPlugin);
export default {
  data: function() {

```

```

    return{
      data: projectNewData,
      height: '100%',
      width: '100%',
      taskFields: {
        id: 'TaskID',
        name: 'TaskName',
        startDate: 'StartDate',
        duration: 'Duration',
        progress: 'Progress',
        child: 'subtasks'
      },
      editSettings: {
        allowAdding: true,
        allowEditing: true,
        allowDeleting: true,
        allowTaskbarEditing: true,
        showDeleteConfirmDialog: true
      },
    };
  },
  provide: {
    gantt: [Edit]
  }
};
</script>
<style>
.e-ganttresize {
  resize: both;
  overflow: auto;
  border: 1px solid red;
  padding: 10px;
  height: 300px;
  min-height: 250px;
  min-width: 250px;
}
</style>

```

{% previewsample "page.domainurl/code-snippet/gantt/scrolling-cs2" %}

### Scroll To Date method

In the Gantt control, When We use the [scrollToDate](#) method, it will scroll the timeline horizontally to the date that we specified in the method's argument.

The following code examples show how the scroll To Date method works in Gantt:

### APP.VUE

```

<template>
  <div>
    <ejs-button id="scrolldate" cssClass="e-info" v-
on:click.native="change">Scroll To Date</ejs-button>
    <ejs-gantt id="GanttContainer" :dataSource="data" :taskFields =
"taskFields" :height = "height" :splitterSettings="splitterSettings"></ejs-
gantt>
    <br><br><br>
  </div>
</template>

```

```

    </div>
  </template>
  <script>
import Vue from "vue";
import { GanttPlugin } from "@syncfusion/ej2-vue-gantt";
import { ButtonPlugin } from "@syncfusion/ej2-vue-buttons";
import { projectNewData } from './data-source.js';
Vue.use(GanttPlugin);
Vue.use(ButtonPlugin);
export default {
  data: function() {
    return {
      data: projectNewData,
      height: '450px',
      taskFields: {
        id: 'TaskID',
        name: 'TaskName',
        startDate: 'StartDate',
        duration: 'Duration',
        progress: 'Progress',
        child: 'subtasks'
      },
      splitterSettings: {
        position: "50%"
      }
    };
  },
  methods: {
    changep: function(e) {
      var ganttChart =
document.getElementById('GanttContainer').ej2_instances[0];
      ganttChart.scrollToDate('04/26/2019');
    }
  }
};
  </script>

```

{% previewsample "page.domainurl/code-snippet/gantt/how-to/scrolltodate-cs1" %}

### Set the vertical scroll position

In the Gantt component, you can set the vertical scroller position dynamically by clicking the custom button using the [setScrollTop](#) method.

### APP.VUE

```

<template>
  <div>
    <ejs-button id="scrolltop" cssClass="e-info" v-
on:click.native="changep">Set Scroll Top</ejs-button>
    <ejs-gantt id="GanttContainer" :dataSource="data" :taskFields =
"taskFields" :height = "height" :splitterSettings="splitterSettings"></ejs-
gantt>
    <br><br><br>
  </div>
</template>
<script>

```

```

import Vue from "vue";
import { GanttPlugin } from "@syncfusion/ej2-vue-gantt";
import { ButtonPlugin } from "@syncfusion/ej2-vue-buttons";
import { editingData } from './data-source.js';
Vue.use(GanttPlugin);
Vue.use(ButtonPlugin);
export default {
  data: function() {
    return {
      data: editingData,
      height: '450px',
      taskFields: {
        id: 'TaskID',
        name: 'TaskName',
        startDate: 'StartDate',
        duration: 'Duration',
        progress: 'Progress',
        child: 'subtasks'
      },
      splitterSettings: {
        position: "50%"
      }
    };
  },
  methods: {
    changep: function(e) {
      var ganttChart =
document.getElementById('GanttContainer').ej2_instances[0];
      ganttChart.ganttChartModule.scrollObject.setScrollTop(500);
    }
  }
};
</script>

```

{% previewsample "page.domainurl/code-snippet/gantt/how-to/setscrolltop-cs1" %}

## Virtual scroll in Vue Gantt component

Virtual Scroll support in Gantt allows you to load large amount of data without performance degradation. To enable Virtual Scrolling, you need to inject `VirtualScroll` module in Gantt.

### Row virtualization

Row virtualization allows you to load and render a large number of tasks in Gantt with effective performance. In this mode, all tasks are fetched initially from the datasource and rendered in the DOM within a compact viewport area.

The number of records displayed in the Gantt is determined by the height.

This mode can be enable by setting the `enableVirtualization` property to `true`.

### APP.VUE

```

<template>
</template>
<script>
import Vue from "vue";

```

```

import { GanttPlugin, Selection, VirtualScroll } from "@syncfusion/ej2-vue-gantt";
import { tempData } from "../datasource.js";
Vue.use(GanttPlugin);
var virtualData = [];
var projId = 1;
for (var i = 0; i < 50; i++) {
    var x = virtualData.length + 1;
    var parent_1 = {};
    parent_1['TaskID'] = x;
    parent_1['TaskName'] = 'Project' + (projId++);
    virtualData.push(parent_1);
    for (var j = 0; j < tempData.length; j++) {
        var subtasks = {};
        subtasks['TaskID'] = tempData[j].TaskID + x;
        subtasks['TaskName'] = tempData[j].TaskName;
        subtasks['StartDate'] = tempData[j].StartDate;
        subtasks['Duration'] = tempData[j].Duration;
        subtasks['Progress'] = tempData[j].Progress;
        subtasks['parentID'] = tempData[j].parentID + x;
        virtualData.push(subtasks);
    }
}
new Vue({
  el: '#app',
  template: `
    <div>
      <ejs-gantt ref='gantt'
        :dataSource= "data"
        :taskFields= "taskFields"
        :allowSelection= "true"
        :enableVirtualization= "true"
        :labelSettings= "labelSettings"
        :height= "height"
        :treeColumnIndex= "1"
        :highlightWeekends= "true"
        :columns= "columns"
        :splitterSettings= "splitterSettings">
      </ejs-gantt>
    </div>
  `,
  data: function() {
    return {
      data : virtualData,
      taskFields: {
        id: 'TaskID',
        name: 'TaskName',
        startDate: 'StartDate',
        endDate: 'EndDate',
        duration: 'Duration',
        progress: 'Progress',
        parentID: 'parentID'
      },
      columns: [
        { field: 'TaskID' },
        { field: 'TaskName' },
        { field: 'StartDate' },

```

```

        { field: 'Duration' },
        { field: 'Progress' }
    ],
    height: '450px',
    labelSettings: {
        taskLabel: 'Progress'
    },
    splitterSettings: {
        columnIndex: 2
    }
}
},
provide: {
    gantt: [ Selection, VirtualScroll ]
}
});
</script>

```

{% previewsample "page.domainurl/code-snippet/gantt/virtual-scroll-cs1" %}

### Timeline virtualization

Timeline virtualization allows you to load a data source having large timespan with high performance. Initially, it renders the timeline with thrice the width of the gantt element, while other timeline cells render on-demand during horizontal scrolling.

This mode can be enable by setting the [enableTimelineVirtualization](#) property to `true`.

### APP.VUE

```

<template>
</template>
<script>
import Vue from "vue";
import { GanttPlugin, Selection, VirtualScroll, Edit } from "@syncfusion/ej2-vue-gantt";
import {tempData} from "../datasource.js";
Vue.use(GanttPlugin);
var virtualData = [];
var projId = 1;
for (var i = 0; i < 50; i++) {
    var x = virtualData.length + 1;
    var parent_1 = {};
    parent_1['TaskID'] = x;
    parent_1['TaskName'] = 'Project' + (projId++);
    virtualData.push(parent_1);
    for (var j = 0; j < tempData.length; j++) {
        var subtasks = {};
        subtasks['TaskID'] = tempData[j].TaskID + x;
        subtasks['TaskName'] = tempData[j].TaskName;
        subtasks['StartDate'] = tempData[j].StartDate;
        subtasks['Duration'] = tempData[j].Duration;
        subtasks['Progress'] = tempData[j].Progress;
        subtasks['parentID'] = tempData[j].parentID + x;
        virtualData.push(subtasks);
    }
}

```

```

}
new Vue({
  el: '#app',
  template: `
    <div>
      <ejs-gantt ref='gantt'
        :dataSource= "data"
        :taskFields= "taskFields"
        :allowSelection= "true"
        :enableVirtualization= "true"
        :labelSettings= "labelSettings"
        :height= "height"
        :treeColumnIndex= "1"
        :highlightWeekends= "true"
        :columns= "columns"
        :autoCalculateDateScheduling='false',
        :editSettings= "editSettings"
        :projectStartDate="projectStartDate"
        :projectEndDate="projectEndDate"
        :enableTimelineVirtualization="true"
        :splitterSettings= "splitterSettings">
      </ejs-gantt>
    </div>
  `,
  data: function() {
    return{
      data : virtualData,
      taskFields: {
        id: 'TaskID',
        name: 'TaskName',
        startDate: 'StartDate',
        endDate: 'EndDate',
        duration: 'Duration',
        progress: 'Progress',
        parentID: 'parentID'
      },
      columns: [
        { field: 'TaskID' },
        { field: 'TaskName' },
        { field: 'StartDate' },
        { field: 'Duration' },
        { field: 'Progress' }
      ],
      height: '450px',
      labelSettings: {
        taskLabel: 'Progress'
      },
      splitterSettings: {
        columnIndex: 2
      },
      editSettings:{
        allowAdding:true
      },
      projectStartDate: new Date('04/01/2019'),
      projectEndDate: new Date('12/31/2025')
    }
  }
})

```

```

,
provide: {
  gantt: [ Selection, VirtualScroll, Edit]
}
});
</script>

```

{% previewsample "page.domainurl/code-snippet/gantt/virtual-scroll-cs3" %}

### Limitations for virtual scroll

- Due to the element height limitation in browsers, the maximum number of records loaded is limited by the browser capacity.
- Cell selection will not be persisted.
- The number of records rendered will be determined by the `height` property.
- It is necessary to mention the height of the Gantt in pixels when enabling Virtual Scrolling.

## Resources in Project View

### Resources in Vue Gantt component

In Gantt, the resources are represented by staff, equipment and materials etc. In Gantt control you can show or allocate the resources (human resources) for each task.

#### Resource collection

The resource collection contains details about resources that are used in the project. Resources are JSON object that contains id, name, unit and group of the resources and this collection is mapped to the Gantt control using the [resources](#) property. These resource fields are mapped to the Gantt control using the [resourceFields](#) property.

#### Resource fields | Description

[id](#) | This field is used to assign resources to the tasks.

[name](#) | This field is used to map the resource names. These names are displayed as one of Gantt columns and also can display as labels using the [labelSettings](#) property.

[unit](#) | It indicates the amount of work that can be done by a resource for the task in a day.

[group](#) | This field is used to group the resources and the tasks assigned to that particular resource into category.

The following code snippets shows resource collection and how it assigned to Gantt control.

```
`ts
```

```

var projectResources: object[] = resources: [
{ resourceid: 1, resourceName: 'Martin Tamer', resourceGroup: 'Planning Team', resourceUnit: 50},
{ resourceid: 2, resourceName: 'Rose Fuller', resourceGroup: 'Testing Team', resourceUnit: 70 },
{ resourceid: 3, resourceName: 'Margaret Buchanan', resourceGroup: 'Approval Team' },
{ resourceid: 4, resourceName: 'Fuller King', resourceGroup: 'Development Team' },
{ resourceid: 5, resourceName: 'Davolio Fuller', resourceGroup: 'Approval Team' },

```



```

{ resourceId: 6, resourceName: 'Van Jack', resourceGroup: 'Development Team', resourceUnit: 40 },
];
export default {
  data: function() {
    return{
      resourceFields: {
        id: 'resourceId', //resource Id Mapping
        name: 'resourceName', //resource Name mapping
        unit: 'resourceUnit', //resource Unit mapping
        group: 'resourceGroup' //resource Group mapping
      },
      resources: projectResources //resource collection dataSource
    };
  },
};

```

#### *Assign resource*

We can assign resources for a task at initial load, using the resource id value of the resources as a collection. This collection is mapped from the dataSource to the Gantt control using the [resourceInfo](#) property.

Resources are assigned to tasks in following ways.

#### *Assign resource alone*

If the unit is not specified for specific resource, the amount of work done will be consider as 100% by default. In such cases, the resource unit will not be displayed in Gantt UI.

```

{ TaskID: 2, TaskName: 'Identify site location', StartDate: new Date('04/02/2019'), Duration: 0, Progress: 50, resources: [1] }

```

#### *Assign resources with unit*

We can assign the quantity of work done by the resources for the specific task as like below code snippet.

```

{ TaskID: 3, TaskName: 'Perform soil test', StartDate: new Date('03/29/2019'), Duration: 4,
  resources: [{resourceId: 2, eUnit: 70}, {resourceId: 1, Unit: 70}] }

```

When resource unit is defined in resource collection, the amount of work done by that particular resource will be same for all the tasks.

The following code snippet shows how to assign the resource for each task and map to Gantt control.

#### APP.VUE

```
<template>
  <div>
    <ejs-gantt ref='gantt' id="GanttContainer" :dataSource="data"
    :taskFields= "taskFields" :resourceFields= "resourceFields" :columns=
    "columns" :resources= "resources" :labelSettings= "labelSettings"></ejs-
    gantt>
  </div>
</template>
<script>
import Vue from "vue";
import { GanttPlugin } from "@syncfusion/ej2-vue-gantt";
Vue.use(GanttPlugin);
export default {
  data: function() {
    return {
      data: [
        {
          TaskID: 1,
          TaskName: 'Project initiation',
          StartDate: new Date('03/29/2019'),
          EndDate: new Date('04/21/2019'),
          subtasks: [
            {
              TaskID: 2, TaskName: 'Identify site location', StartDate:
              new Date('03/29/2019'), Duration: 2,
              Progress: 30, resources: [{ resourceId: 1, Unit: 70 }], 6
            },
            {
              TaskID: 3, TaskName: 'Perform soil test', StartDate: new
              Date('03/29/2019'), Duration: 4,
              resources: [2, 3, 5]
            },
            {
              TaskID: 4, TaskName: 'Soil test approval', StartDate: new
              Date('03/29/2019'), Duration: 1,
              resources: [8, { resourceId: 9, Unit: 50 }], Progress: 30
            },
          ]
        },
        {
          TaskID: 5,
          TaskName: 'Project estimation', StartDate: new Date('03/29/2019'),
          EndDate: new Date('04/21/2019'),
          subtasks: [
            {
              TaskID: 6, TaskName: 'Develop floor plan for estimation',
              StartDate: new Date('03/29/2019'),
              Duration: 3, Progress: 30, resources: [{ resourceId: 4,
              Unit: 50 }],
            },
          ],
        },
      ],
    };
  },
};
```

```

        {
            TaskID: 7, TaskName: 'List materials', StartDate: new
Date('04/01/2019'), Duration: 3,
            resources: [4, 8]
        },
        {
            TaskID: 8, TaskName: 'Estimation approval', StartDate: new
Date('04/01/2019'),
            Duration: 2, resources: [12, { resourceId: 5, Unit: 70 }]
        }
    ]
},
{
    TaskID: 9, TaskName: 'Sign contract', StartDate: new
Date('04/01/2019'), Duration: 1,
    Progress: 30, resources: [12]
}
],
taskFields: {
    id: 'TaskID',
    name: 'TaskName',
    startDate: 'StartDate',
    duration: 'Duration',
    progress: 'Progress',
    resourceInfo: 'resources',
    child: 'subtasks'
},
resourceFields: {
    id: 'resourceId',
    name: 'resourceName',
    unit: 'Unit',
    group: 'resourceGroup'
},
resources: [
    { resourceId: 1, resourceName: 'Martin Tamer' },
    { resourceId: 2, resourceName: 'Rose Fuller' },
    { resourceId: 3, resourceName: 'Margaret Buchanan' },
    { resourceId: 4, resourceName: 'Fuller King' },
    { resourceId: 5, resourceName: 'Davolio Fuller' },
    { resourceId: 6, resourceName: 'Van Jack' },
    { resourceId: 7, resourceName: 'Fuller Buchanan' },
    { resourceId: 8, resourceName: 'Jack Davolio' },
    { resourceId: 9, resourceName: 'Tamer Vinet' },
    { resourceId: 10, resourceName: 'Vinet Fuller' },
    { resourceId: 11, resourceName: 'Bergs Anton' },
    { resourceId: 12, resourceName: 'Construction Supervisor' }
],
columns: [
    { field: 'TaskID', visible: false },
    { field: 'TaskName', headerText: 'Task Name', width: '180' },
    { field: 'resources', headerText: 'Resources', width: '160' },
    { field: 'Duration', width: '100' },
],
labelSettings: {
    rightLabel: 'resources'
}
};

```

```

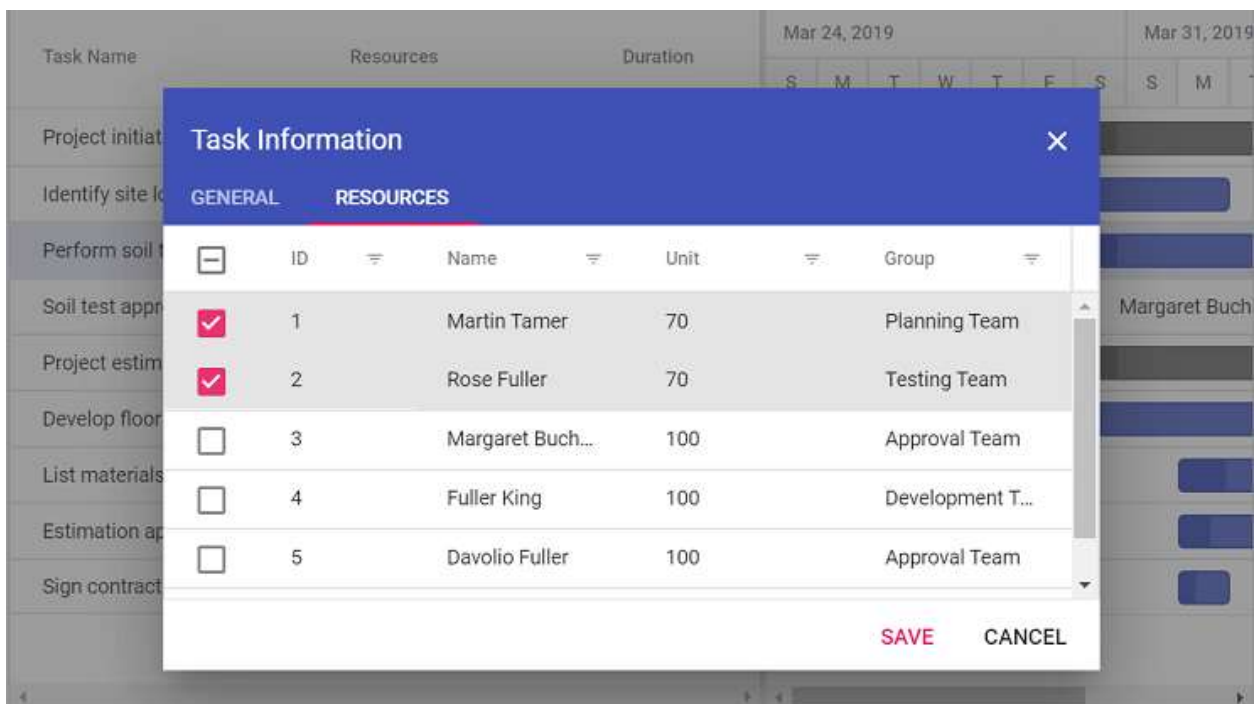
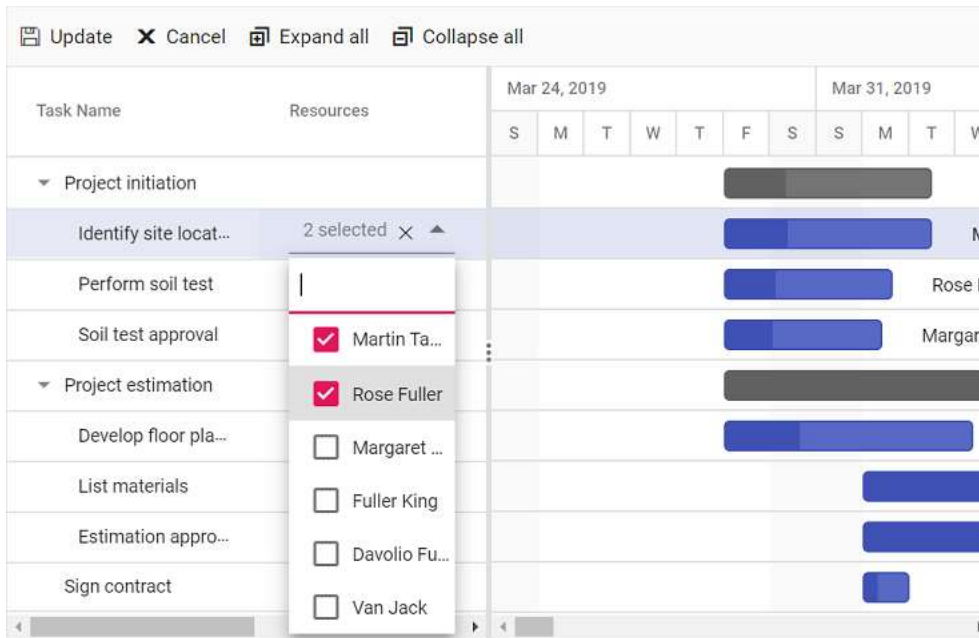
}
}
</script>

```

```
{% previewsample "page.domainurl/code-snippet/gantt/resources-cs1" %}
```

### Add/Edit resource collection

By using cell/ dialog edit option, we can add/remove the multiple resources for a particular task. Resource Unit can be change for a each task on resource tab in edit dialog by double click on the unit cell.



## Work in Vue Gantt component

*Work*

The work is the total hours required to complete a task. Work can be mapped from the data source field using the property [taskFields.work](#). Work can be measured in Hour, Day, Minute. By default, work is measured in Hour and it can be changed, by using the property [workUnit](#).

Note: When the work field is mapped from the data source, the default task type will be FixedWork.

**APP.VUE**

```
<template>
  <div>
    <ejs-gantt ref='gantt' id="GanttContainer" :dataSource="data"
    :editSettings= "editSettings" :taskFields= "taskFields" :treeColumnIndex=
    "1" :columns= "columns" :resources= "resourceResources" :toolbar= "toolbar"
    :allowSelection= "true" ></ejs-gantt>
  </div>
</template>
<script>
import Vue from "vue";
import { GanttPlugin, Toolbar, Edit, Selection } from "@syncfusion/ej2-vue-gantt";
Vue.use(GanttPlugin);
export default {
  data: function() {
    return{
      data: [
        {
          TaskID: 1,
          TaskName: 'Project initiation',
          StartDate: new Date('03/29/2019'),
          EndDate: new Date('04/21/2019'),
          subtasks: [
            {
              TaskID: 2, TaskName: 'Identify site location',
              StartDate: new Date('03/29/2019'), Duration: 2,
              Progress: 30, work: 16, resources: [{
                resourceId: 1, Unit: 70 }, 6]
            },
            {
              TaskID: 3, TaskName: 'Perform soil test',
              StartDate: new Date('03/29/2019'), Duration: 4,
              resources: [2, 3, 5], work: 96
            },
            {
              TaskID: 4, TaskName: 'Soil test approval',
              StartDate: new Date('03/29/2019'), Duration: 1,
              work: 16, resources: [8, { resourceId: 9, Unit:
                50 }], Progress: 30
            },
          ]
        },
        {
          TaskID: 5,
          TaskName: 'Project estimation', StartDate: new
          Date('03/29/2019'), EndDate: new Date('04/21/2019'),
```

```

        subtasks: [
            {
                TaskID: 6, TaskName: 'Develop floor plan for
estimation', StartDate: new Date('03/29/2019'),
                Duration: 3, Progress: 30, resources: [{
resourceId: 4, Unit: 50 }], work: 30
            },
            {
                TaskID: 7, TaskName: 'List materials',
StartDate: new Date('04/01/2019'), Duration: 3,
                work: 48, resources: [4, 8]
            },
            {
                TaskID: 8, TaskName: 'Estimation approval',
StartDate: new Date('04/01/2019'),
                Duration: 2, work: 60, resources: [12, {
resourceId: 5, Unit: 70 }]
            }
        ]
    },
    {
        TaskID: 9, TaskName: 'Sign contract', StartDate: new
Date('04/01/2019'), Duration: 1,
        Progress: 30, resources: [12], work: 24
    }
],
    resourceResources: [
        { resourceId: 1, resourceName: 'Martin Tamer' },
        { resourceId: 2, resourceName: 'Rose Fuller' },
        { resourceId: 3, resourceName: 'Margaret Buchanan' },
        { resourceId: 4, resourceName: 'Fuller King' },
        { resourceId: 5, resourceName: 'Davolio Fuller' },
        { resourceId: 6, resourceName: 'Van Jack' },
        { resourceId: 7, resourceName: 'Fuller Buchanan' },
        { resourceId: 8, resourceName: 'Jack Davolio' },
        { resourceId: 9, resourceName: 'Tamer Vinet' },
        { resourceId: 10, resourceName: 'Vinet Fuller' },
        { resourceId: 11, resourceName: 'Bergs Anton' },
        { resourceId: 12, resourceName: 'Construction Supervisor' }
    ],
    workUnit: 'Hour',
    taskFields: {
        id: 'TaskID',
        name: 'TaskName',
        startDate: 'StartDate',
        duration: 'Duration',
        progress: 'Progress',
        resourceInfo: 'resources',
        work: 'work',
        child: 'subtasks',
    },
    resourceFields: {
        id: 'resourceId',
        name: 'resourceName',
        unit: 'Unit',
    },
    editSettings: {

```

```

        allowAdding: true,
        allowEditing: true,
        allowDeleting: true,
        allowTaskbarEditing: true,
        showDeleteConfirmDialog: true
    },
    toolbar: ['Add', 'Edit', 'Update', 'Delete', 'Cancel', 'ExpandAll',
    'CollapseAll'],
    columns: [
        { field: 'TaskID', visible: false },
        { field: 'TaskName', headerText: 'Name', width: 250 },
        { field: 'work', headerText: 'Work' },
        { field: 'Progress' },
        { field: 'StartDate' },
        { field: 'Duration' },
    ]
    };
},
provide: {
    gantt: [ Toolbar, Edit, Selection ]
}
});
</script>

```

{% previewsample "page.domainurl/code-snippet/gantt/work-cs1" %}

### Task type

The work, duration and resource unit fields of a task depends upon each other and will change automatically on editing any one of these fields. But we can also set these field's values as constant using the [taskType](#) property. `FixedUnit` is the default [taskType](#). The following values can be set to the [taskType](#)

property,

- `FixedDuration` - Duration task field will remain constant while updating resource unit or work field.
- `FixedWork` - Work field will remain constant while updating resource unit or duration fields.
- `FixedUnit` - Resource units will remain constant while updating duration or work field.

### APP.VUE

```

<template>
    <div>
        <ejs-gantt ref='gantt' id="GanttContainer" :dataSource= "data"
        :editSettings= "editSettings" :taskFields= "taskFields" :resourceFields =
        "resourceFields" :treeColumnIndex= "1" :columns= "columns" :resources=
        "resourceResources" :toolbar= "toolbar" :allowSelection= "true" ></ejs-
        gantt>
    </div>
</template>
<script>
import Vue from "vue";
import { GanttPlugin, Toolbar, Edit, Selection } from "@syncfusion/ej2-vue-
gantt";

```

```

Vue.use(GanttPlugin);
export default {
  data: function() {
    return{
      data: [
        {
          TaskID: 1,
          TaskName: 'Project initiation',
          StartDate: new Date('03/29/2019'),
          EndDate: new Date('04/21/2019'),
          subtasks: [
            {
              TaskID: 2, TaskName: 'Identify site location',
              StartDate: new Date('03/29/2019'), Duration: 2,
              Progress: 30, work: 16, resources: [{
                resourceId: 1, Unit: 70 }], 6]
            },
            {
              TaskID: 3, TaskName: 'Perform soil test',
              StartDate: new Date('03/29/2019'), Duration: 4,
              resources: [2, 3, 5], work: 96
            },
            {
              TaskID: 4, TaskName: 'Soil test approval',
              StartDate: new Date('03/29/2019'), Duration: 1,
              work: 16, resources: [8, { resourceId: 9, Unit:
                50 }], Progress: 30
            },
          ]
        },
        {
          TaskID: 5,
          TaskName: 'Project estimation', StartDate: new
            Date('03/29/2019'), EndDate: new Date('04/21/2019'),
          subtasks: [
            {
              TaskID: 6, TaskName: 'Develop floor plan for
                estimation', StartDate: new Date('03/29/2019'),
              Duration: 3, Progress: 30, resources: [{
                resourceId: 4, Unit: 50 }], work: 30
            },
            {
              TaskID: 7, TaskName: 'List materials',
              StartDate: new Date('04/01/2019'), Duration: 3,
              work: 48, resources: [4, 8]
            },
            {
              TaskID: 8, TaskName: 'Estimation approval',
              StartDate: new Date('04/01/2019'),
              Duration: 2, work: 60, resources: [12, {
                resourceId: 5, Unit: 70 }]]
            }
          ]
        },
        {
          TaskID: 9, TaskName: 'Sign contract', StartDate: new
            Date('04/01/2019'), Duration: 1,

```



```

        Progress: 30, resources: [12], work: 24
    },
    ],
    resourceResources: [
    { resourceId: 1, resourceName: 'Martin Tamer' },
    { resourceId: 2, resourceName: 'Rose Fuller' },
    { resourceId: 3, resourceName: 'Margaret Buchanan' },
    { resourceId: 4, resourceName: 'Fuller King' },
    { resourceId: 5, resourceName: 'Davolio Fuller' },
    { resourceId: 6, resourceName: 'Van Jack' },
    { resourceId: 7, resourceName: 'Fuller Buchanan' },
    { resourceId: 8, resourceName: 'Jack Davolio' },
    { resourceId: 9, resourceName: 'Tamer Vinet' },
    { resourceId: 10, resourceName: 'Vinet Fuller' },
    { resourceId: 11, resourceName: 'Bergs Anton' },
    { resourceId: 12, resourceName: 'Construction Supervisor' }
    ],
    workUnit: 'Hour',
    taskType: 'FixedWork',
    taskFields: {
        id: 'TaskID',
        name: 'TaskName',
        startDate: 'StartDate',
        duration: 'Duration',
        progress: 'Progress',
        resourceInfo: 'resources',
        work: 'work',
        child: 'subtasks'
    },
    resourceFields: {
        id: 'resourceId',
        name: 'resourceName',
        unit: 'Unit',
    },
    editSettings: {
        allowAdding: true,
        allowEditing: true,
        allowDeleting: true,
        allowTaskbarEditing: true,
        showDeleteConfirmDialog: true
    },
    toolbar: ['Add', 'Edit', 'Update', 'Delete', 'Cancel',
'ExpandAll', 'CollapseAll'],
    columns: [
        { field: 'TaskID', visible: false },
        { field: 'TaskName', headerText: 'Task Name', width: '180'
},
        { field: 'resources', headerText: 'Resources', width: '160'
},
        { field: 'work', headerText: 'Work', width: '110' },
        { field: 'Duration', width: '100' },
        { field: 'taskType', headerText: 'Task Type', width: '110' }
    ]
    ];
    },
    provide: {
        gantt: [ Toolbar, Edit, Selection ]
    }
}

```

```

    }
  });
</script>

```

{% previewsample "page.domainurl/code-snippet/gantt/work-cs2" %}

Following table explains how the work, duration and resource unit fields will gets updated on changing any of the fields

Task Type | Changes in Duration | Changes in work | Changes in Resource Units

Fixed Duration | Work field updates | Resource unit updates | Work field updates

Fixed Work | Resource unit updates. Note: For manually scheduled task work will update. | Duration field updates. Note: For manually scheduled task resource unit updates. | Duration will update. Note: For manually scheduled task work field updates.

Fixed Unit | Work field updates | Duration field updates. Note: For manually scheduled task resource unit updates. | Duration will update. Note: For manually scheduled task work field updates.

Note: 1. Fixed Unit is the default taskType in Gantt. 2. The above calculations are not applicable for Milestones.

## Resource View

### Resource view in Vue Gantt component

The resource breakdown view is used to visualize the tasks assigned to each resource in hierarchy manner. Resources are displayed as parents and all the tasks assigned to each resource are displayed as its child records. It can be initialized by setting the [viewType](#) property to [Link to the Video](#).

To learn about Gantt Chart Resource view Concepts, you can check on this video:

### Resource task

A task assigned to one or more resources are termed as resource task and it is added as child task to the respective resource. Already assigned task can also be shared or moved with other resources by adding a resource name to the task or removing resource name from the task by cell or dialog editing.

Note: Currently there is no support for unscheduled task in Resource view Gantt.

## APP.VUE

```

<template>
  <div>
    <ejs-gantt ref='gantt' id="GanttContainer" :dataSource="data"
    :viewType= "viewType" :taskFields= "taskFields" :resourceFields=
    "resourceFields" :treeColumnIndex= "1" :columns= "columns"
    :splitterSettings= "splitterSettings" :resources= "resourceCollection"
    :toolbar= "toolbar" :allowResizing= "true" :allowSelection= "true"
    :highlightWeekends= "true" :projectStartDate= "projectStartDate"
    :projectEndDate= "projectEndDate"></ejs-gantt>
  </div>
</template>
<script>
import Vue from "vue";
import { GanttPlugin, Toolbar, Edit, Selection } from "@syncfusion/ej2-vue-gantt";
Vue.use(GanttPlugin);

```

```

export default {
  data: function() {
    return{
      data: [
        {
          TaskID: 1,
          TaskName: 'Project initiation',
          StartDate: new Date('03/29/2019'),
          EndDate: new Date('04/21/2019'),
          subtasks: [
            {
              TaskID: 2, TaskName: 'Identify site location',
              StartDate: new Date('03/29/2019'), Duration: 2,
              Progress: 30, work: 10, resources: [{ resourceId: 1,
              resourceUnit: 50 }]
            },
            {
              TaskID: 3, TaskName: 'Perform soil test', StartDate: new
              Date('03/29/2019'), Duration: 4,
              resources: [{resourceId: 2, resourceUnit: 70}],
              Progress: 30, work: 20
            },
            {
              TaskID: 4, TaskName: 'Soil test approval', StartDate:
              new Date('03/29/2019'), Duration: 1,
              resources: [{resourceId: 3, resourceUnit: 25}, {
              resourceId: 1, resourceUnit: 75 }], Progress: 30, work: 10,
            },
          ]
        },
        {
          TaskID: 5,
          TaskName: 'Project estimation', StartDate: new
          Date('03/29/2019'), EndDate: new Date('04/21/2019'),
          subtasks: [
            {
              TaskID: 6, TaskName: 'Develop floor plan for
              estimation', StartDate: new Date('03/29/2019'),
              Duration: 3, Progress: 30, resources: [{ resourceId: 4,
              resourceUnit: 50 }, {resourceId: 2, resourceUnit: 70}], work: 30
            },
            {
              TaskID: 7, TaskName: 'List materials', StartDate: new
              Date('04/01/2019'), Duration: 3,
              resources: [{resourceId: 6, resourceUnit: 40}],
              Progress: 30, work: 40
            },
            {
              TaskID: 8, TaskName: 'Estimation approval', StartDate:
              new Date('04/01/2019'),
              Duration: 2, resources: [{ resourceId: 5, resourceUnit:
              75 }], Progress: 30, work: 60,
            }
          ]
        },
      ],
    }
  }
}

```

```

        TaskID: 9, TaskName: 'Sign contract', StartDate: new
Date('04/01/2019'), Duration: 1,
        Progress: 30,
    },
],
    resourceCollection: [
        { resourceId: 1, resourceName: 'Martin Tamer', resourceGroup:
'Planning Team' },
        { resourceId: 2, resourceName: 'Rose Fuller', resourceGroup:
'Testing Team' },
        { resourceId: 3, resourceName: 'Margaret Buchanan',
resourceGroup: 'Approval Team' },
        { resourceId: 4, resourceName: 'Fuller King', resourceGroup:
'Development Team' },
        { resourceId: 5, resourceName: 'Davolio Fuller', resourceGroup:
'Approval Team' },
        { resourceId: 6, resourceName: 'Van Jack', resourceGroup:
'Development Team' },
    ],
    viewType: 'ResourceView',
    taskFields: {
        id: 'TaskID',
        name: 'TaskName',
        startDate: 'StartDate',
        endDate: 'EndDate',
        duration: 'Duration',
        progress: 'Progress',
        resourceInfo: 'resources',
        work: 'Work',
        child: 'subtasks',
    },
    resourceFields: {
        id: 'resourceId',
        name: 'resourceName',
        unit: 'Unit',
        group: 'resourceGroup'
    },
    editSettings: {
        allowAdding: true,
        allowEditing: true,
        allowDeleting: true,
        allowTaskbarEditing: true,
        showDeleteConfirmDialog: true
    },
    columns: [
        { field: 'TaskID', visible: false },
        { field: 'TaskName', headerText: 'Name', width: 250 },
        { field: 'work', headerText: 'Work' },
        { field: 'Progress' },
        { field: 'resourceGroup', headerText: 'Group' },
        { field: 'StartDate' },
        { field: 'Duration' }
    ],
    toolbar: ['Add', 'Edit', 'Update', 'Delete', 'Cancel', 'ExpandAll',
'CollapseAll'],
    labelSettings: {
        rightLabel: 'resources'
    }

```

```

    },
    splitterSettings: {
        columnIndex: 3
    },
    projectStartDate: new Date('03/28/2019'),
    projectEndDate: new Date('05/18/2019'),
    };
},
provide: {
    gantt: [ Toolbar, Edit, Selection ]
}
};
</script>

```

{% previewsample "page.domainurl/code-snippet/gantt/resource-view-cs1" %}

### Resource overallocation

When a resource is assigned too much of work to complete within a day of resource's available time then it is called as overallocation.

The available working time of resources for completing the task in a day will be calculated based on the `dayWorkingTime` property and `resource unit`.

The range of overallocation dates can be highlighted by a square bracket. It can be enabled by setting the `showOverallocation` property as `true`. The following code example demonstrates how to hide or show the over allocation by clicking the custom button.

Note: By default, the `showOverAllocation` property value is `false`.

### APP.VUE

```

<template>
  <div>
    <ejs-gantt ref='gantt' id="GanttContainer" :dataSource="data"
    :viewType= "viewType" :taskFields= "taskFields" :resourceFields=
    "resourceFields"
      :treeColumnIndex= "1" :columns= "columns" :resources=
    "resourceCollection" :toolbar= "toolbar" :toolbarClick="toolbarClick"
    :allowSelection= "true"
      :editSettings= "editSettings" :labelSettings="labelSettings"
    :highlightWeekends= "true" :projectStartDate= "projectStartDate"
      :projectEndDate= "projectEndDate" :showOverAllocation =
    "true"></ejs-gantt>
  </div>
</template>
<script>
import Vue from "vue";
import { GanttPlugin, Toolbar, Edit, Selection } from "@syncfusion/ej2-vue-gantt";
Vue.use(GanttPlugin);
export default {
  data: function() {
    return{
      data: [
        {
          TaskID: 1,

```

```

    TaskName: 'Project initiation',
    StartDate: new Date('03/29/2019'),
    EndDate: new Date('04/21/2019'),
    subtasks: [
        {
            TaskID: 2, TaskName: 'Identify site location', StartDate:
new Date('03/29/2019'), Duration: 3,
            Progress: 30, work: 10, resources: [{ resourceId: 1,
resourceUnit: 50 }]
        },
        {
            TaskID: 3, TaskName: 'Perform soil test', StartDate: new
Date('04/03/2019'), Duration: 4,
            resources: [{ resourceId: 1, resourceUnit: 70 }],
            Predecessor: 2, Progress: 30, work: 20
        },
        {
            TaskID: 4, TaskName: 'Soil test approval', StartDate: new
Date('04/09/2019'), Duration: 4,
            resources: [{ resourceId: 1, resourceUnit: 25 }],
            Predecessor: 3, Progress: 30, work: 10,
        },
    ]
},
{
    TaskID: 5,
    TaskName: 'Project estimation', StartDate: new Date('03/29/2019'),
    EndDate: new Date('04/21/2019'),
    subtasks: [
        {
            TaskID: 6, TaskName: 'Develop floor plan for estimation',
StartDate: new Date('04/01/2019'),
            Duration: 5, Progress: 30, resources: [{ resourceId: 2,
resourceUnit: 50 }], work: 30
        },
        {
            TaskID: 7, TaskName: 'List materials', StartDate: new
Date('04/04/2019'), Duration: 4,
            resources: [{ resourceId: 2, resourceUnit: 40 }],
            Predecessor: '6FS-2', Progress: 30, work: 40
        },
        {
            TaskID: 8, TaskName: 'Estimation approval', StartDate: new
Date('04/09/2019'),
            Duration: 4, resources: [{ resourceId: 2, resourceUnit: 75
}], Predecessor: '7FS-1', Progress: 30, work: 60,
        }
    ]
},
{
    TaskID: 9,
    TaskName: 'Site work',
    StartDate: new Date('04/04/2019'),
    EndDate: new Date('04/21/2019'),
    subtasks: [
        {

```

```

        TaskID: 10, TaskName: 'Install temporary power service',
StartDate: new Date('04/01/2019'), Duration: 14,
        Progress: 30, resources: [{ resourceId: 3, resourceUnit: 75
    }]
    },
    {
        TaskID: 11, TaskName: 'Clear the building site', StartDate:
new Date('04/08/2019'),
        Duration: 9, Progress: 30, Predecessor: '10FS-9', resources:
[3]
    },
    {
        TaskID: 12, TaskName: 'Sign contract', StartDate: new
Date('04/12/2019'),
        Duration: 5, resources: [3], Predecessor: '11FS-5'
    },
]
},
],
    resourceCollection: [
        { resourceId: 1, resourceName: 'Martin Tamer', resourceGroup:
'Planning Team' },
        { resourceId: 2, resourceName: 'Rose Fuller', resourceGroup:
'Testing Team' },
        { resourceId: 3, resourceName: 'Margaret Buchanan',
resourceGroup: 'Approval Team' },
    ],
    viewType: 'ResourceView',
    taskFields: {
        id: 'TaskID',
        name: 'TaskName',
        startDate: 'StartDate',
        endDate: 'EndDate',
        duration: 'Duration',
        progress: 'Progress',
        dependency: 'Predecessor',
        resourceInfo: 'resources',
        work: 'Work',
        child: 'subtasks',
    },
    resourceFields: {
        id: 'resourceId',
        name: 'resourceName',
        unit: 'Unit',
        group: 'resourceGroup'
    },
    editSettings: {
        allowAdding: true,
        allowEditing: true,
        allowDeleting: true,
        allowTaskbarEditing: true,
        showDeleteConfirmDialog: true
    },
    columns: [
        { field: 'TaskID', visible: false },
        { field: 'TaskName', headerText: 'Name', width: 250 },
        { field: 'work', headerText: 'Work' },
    ],

```

```

        { field: 'Progress' },
        { field: 'resourceGroup', headerText: 'Group' },
        { field: 'StartDate' },
        { field: 'Duration' }
    ],
    toolbarClick: (args: ClickEventArgs) => {
        if (args.item.id === 'showhidebar') {
            var ganttObj =
document.getElementById('GanttContainer').ej2_instances[0];
            ganttObj.showOverAllocation =
ganttObj.showOverAllocation ? false : true;
        }
    },
    toolbar: ['Add', 'Edit', 'Update', 'Delete', 'Cancel', 'ExpandAll',
'CollapseAll',
        { text: 'Show/Hide Overallocation', tooltipText: 'Show/Hide
Overallocation', id: 'showhidebar' }]];
    labelSettings: {
        rightLabel: 'resources',
        taskLabel: 'Progress'
    },
    projectStartDate: new Date('03/28/2019'),
    projectEndDate: new Date('05/18/2019'),
    };
},
provide: {
    gantt: [ Toolbar, Edit, Selection ]
}
};
</script>

```

{% previewsample "page.domainurl/code-snippet/gantt/resource-view-cs2" %}

### Unassigned task

A task not assigned to any one of the resource are termed as unassigned tasks. The unassigned tasks are grouped with a name as **Unassigned Task** and displayed at the bottom of Gantt data collection . It is validated at load time during Gantt record creation by default based on a task **resourceInfo** mapping property in the Gantt chart data source. If the resource is assigned to the unassigned grouped tasks, the task will be moved as child to the respective resource.

### Enable taskbar drag and drop

In Gantt, you can enable taskbar drag and drop between resources by using the [allowTaskbarDragAndDrop](#) property. This allows you to move a taskbar from one resource to another vertically, making it easier to schedule tasks and manage resources.

## APP.VUE

```

<template>
  <div>
    <ejs-gantt ref='gantt' id="GanttContainer" :dataSource="data"
:viewType= "viewType" :taskFields= "taskFields" :resourceFields=
"resourceFields"
      :treeColumnIndex= "1" :columns= "columns" :resources=
"resourceCollection" :toolbar= "toolbar" :allowSelection= "true"

```



```

      :editSettings= "editSettings" :labelSettings="labelSettings"
    :highlightWeekends= "true" :projectStartDate= "projectStartDate"
      :projectEndDate= "projectEndDate" :showOverAllocation = "true"
    :enableMultiTaskbar= "true" :allowTaskbarDragAndDrop= "true"></ejs-gantt>
  </div>
</template>
<script>
import Vue from "vue";
import { GanttPlugin, Toolbar, Edit, Selection } from "@syncfusion/ej2-vue-gantt";
Vue.use(GanttPlugin);
export default {
  data: function() {
    return{
      data: [
        {
          TaskID: 1,
          TaskName: 'Project initiation',
          StartDate: new Date('03/29/2019'),
          EndDate: new Date('04/21/2019'),
          subtasks: [
            {
              TaskID: 2, TaskName: 'Identify site location', StartDate:
new Date('03/29/2019'), Duration: 3,
              Progress: 30, work: 10, resources: [{ resourceId: 1,
resourceUnit: 50 }]
            },
            {
              TaskID: 3, TaskName: 'Perform soil test', StartDate: new
Date('04/03/2019'), Duration: 4,
              resources: [{ resourceId: 1, resourceUnit: 70 }],
              Predecessor: 2, Progress: 30, work: 20
            },
            {
              TaskID: 4, TaskName: 'Soil test approval', StartDate: new
Date('04/09/2019'), Duration: 4,
              resources: [{ resourceId: 1, resourceUnit: 25 }],
              Predecessor: 3, Progress: 30, work: 10,
            },
          ]
        },
        {
          TaskID: 5,
          TaskName: 'Project estimation', StartDate: new Date('03/29/2019'),
          EndDate: new Date('04/21/2019'),
          subtasks: [
            {
              TaskID: 6, TaskName: 'Develop floor plan for estimation',
              StartDate: new Date('04/01/2019'),
              Duration: 5, Progress: 30, resources: [{ resourceId: 2,
resourceUnit: 50 }], work: 30
            },
            {
              TaskID: 7, TaskName: 'List materials', StartDate: new
Date('04/04/2019'), Duration: 4,
              resources: [{ resourceId: 2, resourceUnit: 40 }],
              Predecessor: '6FS-2', Progress: 30, work: 40
            }
          ]
        }
      ]
    }
  }
}

```

```

    },
    {
        TaskID: 8, TaskName: 'Estimation approval', StartDate: new
Date('04/09/2019'),
        Duration: 4, resources: [{ resourceId: 2, resourceUnit: 75
}], Predecessor: '7FS-1', Progress: 30, work: 60,
    }
]
},
{
    TaskID: 9,
    TaskName: 'Site work',
    StartDate: new Date('04/04/2019'),
    EndDate: new Date('04/21/2019'),
    subtasks: [
        {
            TaskID: 10, TaskName: 'Install temporary power service',
StartDate: new Date('04/01/2019'), Duration: 14,
            Progress: 30, resources: [{ resourceId: 3, resourceUnit: 75
}]
        },
        {
            TaskID: 11, TaskName: 'Clear the building site', StartDate:
new Date('04/08/2019'),
            Duration: 9, Progress: 30, Predecessor: '10FS-9', resources:
[3]
        },
        {
            TaskID: 12, TaskName: 'Sign contract', StartDate: new
Date('04/12/2019'),
            Duration: 5, resources: [3], Predecessor: '11FS-5'
        },
    ]
}
],
resourceCollection: [
    { resourceId: 1, resourceName: 'Martin Tamer', resourceGroup:
'Planning Team', isExpand: false},
    { resourceId: 2, resourceName: 'Rose Fuller', resourceGroup:
'Testing Team', isExpand: true},
    { resourceId: 3, resourceName: 'Margaret Buchanan',
resourceGroup: 'Approval Team', isExpand: false }
],
viewType: 'ResourceView',
taskFields: {
    id: 'TaskID',
    name: 'TaskName',
    startDate: 'StartDate',
    endDate: 'EndDate',
    duration: 'Duration',
    progress: 'Progress',
    dependency: 'Predecessor',
    resourceInfo: 'resources',
    work: 'Work',
    expandState: 'isExpand',
    child: 'subtasks',
},
},

```

```

resourceFields: {
    id: 'resourceId',
    name: 'resourceName',
    unit: 'Unit',
    group: 'resourceGroup'
},
editSettings: {
    allowAdding: true,
    allowEditing: true,
    allowDeleting: true,
    allowTaskbarEditing: true,
    showDeleteConfirmDialog: true
},
columns: [
    { field: 'TaskID' },
    { field: 'TaskName', headerText: 'Name', width: 250 },
    { field: 'work', headerText: 'Work' },
    { field: 'Progress' },
    { field: 'resourceGroup', headerText: 'Group' },
    { field: 'StartDate' },
    { field: 'Duration' }
],
toolbar: ['ExpandAll', 'CollapseAll'];
labelSettings: {
    rightLabel: 'resources',
    taskLabel: 'TaskName'
},
projectStartDate: new Date('03/28/2019'),
projectEndDate: new Date('05/18/2019'),
};
},
provide: {
    gantt: [ Toolbar, Edit, Selection ]
}
};
</script>

```

{% previewsample "page.domainurl/code-snippet/gantt/resource-view-cs4" %}

## Resource Multi Taskbar in Vue Gantt component

### Resource multi taskbar

To visualize multiple tasks assigned to each resource in a row when the records are in the collapsed state. It can be enabled by settings the `enableMultiTaskbar` property value as `true`.

The collapse or expand action of a resource record can be achieved only by using the tree grid side arrow icon. Because it will be disabled on chart side action for this support.

When a resource has multiple tasks scheduled on the same date, then the tasks will be overlapped one another. Taskbar editing is also possible to change the task scheduling on the collapsed state.

Note: By default, the `enableMultiTaskbar` property value is `false`.

### APP.VUE

```

<template>
  <div>

```

```

<ejs-gantt ref='gantt' id="GanttContainer" :dataSource="data"
:viewType= "viewType" :taskFields= "taskFields" :resourceFields=
"resourceFields"
:treeColumnIndex= "1" :columns= "columns" :resources=
"resourceCollection" :toolbar= "toolbar" :allowSelection= "true"
:editSettings= "editSettings" :labelSettings="labelSettings"
:highlightWeekends= "true" :projectStartDate= "projectStartDate"
:projectEndDate= "projectEndDate" :showOverAllocation = "true"
:enableMultiTaskbar= "true" :collapseAllParentTasks= "true"></ejs-gantt>
</div>
</template>
<script>
import Vue from "vue";
import { GanttPlugin, Toolbar, Edit, Selection } from "@syncfusion/ej2-vue-gantt";
Vue.use(GanttPlugin);
export default {
  data: function() {
    return{
      data: [
        {
          TaskID: 1,
          TaskName: 'Project initiation',
          StartDate: new Date('03/29/2019'),
          EndDate: new Date('04/21/2019'),
          subtasks: [
            {
              TaskID: 2, TaskName: 'Identify site location', StartDate:
new Date('03/29/2019'), Duration: 3,
              Progress: 30, work: 10, resources: [{ resourceId: 1,
resourceUnit: 50 }]
            },
            {
              TaskID: 3, TaskName: 'Perform soil test', StartDate: new
Date('04/03/2019'), Duration: 4,
              resources: [{ resourceId: 1, resourceUnit: 70 }],
              Predecessor: 2, Progress: 30, work: 20
            },
            {
              TaskID: 4, TaskName: 'Soil test approval', StartDate: new
Date('04/09/2019'), Duration: 4,
              resources: [{ resourceId: 1, resourceUnit: 25 }],
              Predecessor: 3, Progress: 30, work: 10,
            },
          ]
        },
        {
          TaskID: 5,
          TaskName: 'Project estimation', StartDate: new Date('03/29/2019'),
          EndDate: new Date('04/21/2019'),
          subtasks: [
            {
              TaskID: 6, TaskName: 'Develop floor plan for estimation',
              StartDate: new Date('04/01/2019'),
              Duration: 5, Progress: 30, resources: [{ resourceId: 2,
resourceUnit: 50 }], work: 30
            },
          ]
        }
      ]
    }
  }
}

```

```

        {
            TaskID: 7, TaskName: 'List materials', StartDate: new
Date('04/04/2019'), Duration: 4,
            resources: [{ resourceId: 2, resourceUnit: 40 }],
            Predecessor: '6FS-2', Progress: 30, work: 40
        },
        {
            TaskID: 8, TaskName: 'Estimation approval', StartDate: new
Date('04/09/2019'),
            Duration: 4, resources: [{ resourceId: 2, resourceUnit: 75
}], Predecessor: '7FS-1', Progress: 30, work: 60,
        }
    ]
},
{
    TaskID: 9,
    TaskName: 'Site work',
    StartDate: new Date('04/04/2019'),
    EndDate: new Date('04/21/2019'),
    subtasks: [
        {
            TaskID: 10, TaskName: 'Install temporary power service',
            StartDate: new Date('04/01/2019'), Duration: 14,
            Progress: 30, resources: [{ resourceId: 3, resourceUnit: 75
}]]
        },
        {
            TaskID: 11, TaskName: 'Clear the building site', StartDate:
new Date('04/08/2019'),
            Duration: 9, Progress: 30, Predecessor: '10FS-9', resources:
[3]
        },
        {
            TaskID: 12, TaskName: 'Sign contract', StartDate: new
Date('04/12/2019'),
            Duration: 5, resources: [3], Predecessor: '11FS-5'
        },
    ]
},
],
    resourceCollection: [
        { resourceId: 1, resourceName: 'Martin Tamer', resourceGroup:
'Planning Team', isExpand: false},
        { resourceId: 2, resourceName: 'Rose Fuller', resourceGroup:
'Testing Team', isExpand: true},
        { resourceId: 3, resourceName: 'Margaret Buchanan',
resourceGroup: 'Approval Team', isExpand: false }
    ],
    viewType: 'ResourceView',
    taskFields: {
        id: 'TaskID',
        name: 'TaskName',
        startDate: 'StartDate',
        endDate: 'EndDate',
        duration: 'Duration',
        progress: 'Progress',
        dependency: 'Predecessor',
    }
}

```

```

        resourceInfo: 'resources',
        work: 'Work',
        expandState: 'isExpand',
        child: 'subtasks',
    },
    resourceFields: {
        id: 'resourceId',
        name: 'resourceName',
        unit: 'Unit',
        group: 'resourceGroup'
    },
    editSettings: {
        allowAdding: true,
        allowEditing: true,
        allowDeleting: true,
        allowTaskbarEditing: true,
        showDeleteConfirmDialog: true
    },
    columns: [
        { field: 'TaskID' },
        { field: 'TaskName', headerText: 'Name', width: 250 },
        { field: 'work', headerText: 'Work' },
        { field: 'Progress' },
        { field: 'resourceGroup', headerText: 'Group' },
        { field: 'StartDate' },
        { field: 'Duration' }
    ],
    toolbar: ['Add', 'Edit', 'Update', 'Delete', 'Cancel', 'ExpandAll',
'CollapseAll'];
    labelSettings: {
        rightLabel: 'resources',
        taskLabel: 'TaskName'
    },
    projectStartDate: new Date('03/28/2019'),
    projectEndDate: new Date('05/18/2019'),
    };
},
provide: {
    gantt: [ Toolbar, Edit, Selection ]
}
};
</script>

```

{% previewsample "page.domainurl/code-snippet/gantt/resource-view-cs3" %}

#### Disable taskbar overlap

In Gantt, you can disable taskbar overlap between resource tasks using the [allowTaskbarOverlap](#) property. This prevents the taskbars for different tasks from overlapping on the same row, making it easier to distinguish between the different tasks and manage resources effectively.

When `allowTaskbarOverlap` is set to false, the resources are displayed in a single row and the row height will be extended to occupy the tasks of the resource when it is in a collapsed state. This view allows you to easily identify any overallocation of tasks for a resource in a project.

It's important to note that when `allowTaskbarOverlap` is disabled, task dependencies or relationships cannot be established between tasks that are rendered in multiple lines for the same resource. If you need to establish dependencies between tasks for the same resource, you may want to consider enabling taskbar overlap.

### APP.VUE

```
<template>
  <div>
    <ejs-gantt ref='gantt' id="GanttContainer" :dataSource="data"
    :viewType= "viewType" :taskFields= "taskFields" :resourceFields=
    "resourceFields"
      :treeColumnIndex= "1" :columns= "columns" :resources=
    "resourceCollection" :toolbar= "toolbar" :allowSelection= "true"
      :editSettings= "editSettings" :labelSettings="labelSettings"
    :highlightWeekends= "true" :projectStartDate= "projectStartDate"
      :projectEndDate= "projectEndDate" :showOverAllocation = "true"
    :enableMultiTaskbar= "true" :allowTaskbarOverlap= "false"></ejs-gantt>
  </div>
</template>
<script>
import Vue from "vue";
import { GanttPlugin, Toolbar, Edit, Selection } from "@syncfusion/ej2-vue-gantt";
Vue.use(GanttPlugin);
export default {
  data: function() {
    return{
      data: [
        {
          TaskID: 1,
          TaskName: 'Project initiation',
          StartDate: new Date('03/29/2019'),
          EndDate: new Date('04/21/2019'),
          subtasks: [
            {
              TaskID: 2, TaskName: 'Identify site location', StartDate:
new Date('03/29/2019'), Duration: 3,
              Progress: 30, work: 10, resources: [{ resourceId: 1,
resourceUnit: 50 }]
            },
            {
              TaskID: 3, TaskName: 'Perform soil test', StartDate: new
Date('04/03/2019'), Duration: 4,
              resources: [{ resourceId: 1, resourceUnit: 70 }],
              Predecessor: 2, Progress: 30, work: 20
            },
            {
              TaskID: 4, TaskName: 'Soil test approval', StartDate: new
Date('04/09/2019'), Duration: 4,
              resources: [{ resourceId: 1, resourceUnit: 25 }],
              Predecessor: 3, Progress: 30, work: 10,
            },
          ]
        },
        {
          TaskID: 5,
```

```

    TaskName: 'Project estimation', StartDate: new Date('03/29/2019'),
    EndDate: new Date('04/21/2019'),
    subtasks: [
        {
            TaskID: 6, TaskName: 'Develop floor plan for estimation',
            StartDate: new Date('04/01/2019'),
            Duration: 5, Progress: 30, resources: [{ resourceId: 2,
            resourceUnit: 50 }], work: 30
        },
        {
            TaskID: 7, TaskName: 'List materials', StartDate: new
            Date('04/04/2019'), Duration: 4,
            resources: [{ resourceId: 2, resourceUnit: 40 }],
            Predecessor: '6FS-2', Progress: 30, work: 40
        },
        {
            TaskID: 8, TaskName: 'Estimation approval', StartDate: new
            Date('04/09/2019'),
            Duration: 4, resources: [{ resourceId: 2, resourceUnit: 75
            }], Predecessor: '7FS-1', Progress: 30, work: 60,
        }
    ]
},
{
    TaskID: 9,
    TaskName: 'Site work',
    StartDate: new Date('04/04/2019'),
    EndDate: new Date('04/21/2019'),
    subtasks: [
        {
            TaskID: 10, TaskName: 'Install temporary power service',
            StartDate: new Date('04/01/2019'), Duration: 14,
            Progress: 30, resources: [{ resourceId: 3, resourceUnit: 75
            }]}
        },
        {
            TaskID: 11, TaskName: 'Clear the building site', StartDate:
            new Date('04/08/2019'),
            Duration: 9, Progress: 30, Predecessor: '10FS-9', resources:
            [3]
        },
        {
            TaskID: 12, TaskName: 'Sign contract', StartDate: new
            Date('04/12/2019'),
            Duration: 5, resources: [3], Predecessor: '11FS-5'
        },
    ]
}
],
    resourceCollection: [
        { resourceId: 1, resourceName: 'Martin Tamer', resourceGroup:
        'Planning Team', isExpand: false},
        { resourceId: 2, resourceName: 'Rose Fuller', resourceGroup:
        'Testing Team', isExpand: true},
        { resourceId: 3, resourceName: 'Margaret Buchanan',
        resourceGroup: 'Approval Team', isExpand: false }
    ],

```



```

        viewType: 'ResourceView',
        taskFields: {
            id: 'TaskID',
            name: 'TaskName',
            startDate: 'StartDate',
            endDate: 'EndDate',
            duration: 'Duration',
            progress: 'Progress',
            dependency: 'Predecessor',
            resourceInfo: 'resources',
            work: 'Work',
            expandState: 'isExpand',
            child: 'subtasks',
        },
        resourceFields: {
            id: 'resourceId',
            name: 'resourceName',
            unit: 'Unit',
            group: 'resourceGroup'
        },
        editSettings: {
            allowAdding: true,
            allowEditing: true,
            allowDeleting: true,
            allowTaskbarEditing: true,
            showDeleteConfirmDialog: true
        },
        columns: [
            { field: 'TaskID' },
            { field: 'TaskName', headerText: 'Name', width: 250 },
            { field: 'work', headerText: 'Work' },
            { field: 'Progress' },
            { field: 'resourceGroup', headerText: 'Group' },
            { field: 'StartDate' },
            { field: 'Duration' }
        ],
        toolbar: ['ExpandAll', 'CollapseAll'];
        labelSettings: {
            rightLabel: 'resources',
            taskLabel: 'TaskName'
        },
        projectStartDate: new Date('03/28/2019'),
        projectEndDate: new Date('05/18/2019'),
    };
    },
    provide: {
        gantt: [ Toolbar, Edit, Selection ]
    }
};
</script>

```

{% previewsample "page.domainurl/code-snippet/gantt/resource-view-cs5" %}

## Filtering

### Filtering in Vue Gantt component

Filtering allows you to view specific or related records based on filter criteria. This can be done in the Gantt Component by using the filter menu support and toolbar search support. To enable filtering in the Gantt Component, set the [allowFiltering](#) to `true`. Menu filtering support can be configured using the [filterSettings](#) property and toolbar searching can be configured using the [searchSettings](#) property.

To use the filter, inject the [Filter](#) module in the `provide` section.

### Filter hierarchy modes

The Gantt supports a set of filtering modes with the [filterSettings.hierarchyMode](#) property. The following are the types of filter hierarchy modes available in the Gantt component:

- **Parent:** This is the default filter hierarchy mode in Gantt. The filtered records are displayed with its parent records. If the filtered records do not have any parent record, then only the filtered records will be displayed.
- **Child:** Displays the filtered records with its child record. If the filtered records do not have any child record, then only the filtered records will be displayed.
- **Both:** Displays the filtered records with its both parent and child records. If the filtered records do not have any parent and child records, then only the filtered records will be displayed.
- **None:** Displays only the filtered records.

### APP.VUE

```
<template>
  <div>
    <table>
      <tr>
        <td style="width: 70%">
          <ejs-dropdownlist id="filter-type" value='Parent'
:dataSource="dataSource" :fields = "fields" :change ="change"> Filter
Hierarchy </ejs-dropdownlist>
        </td>
      </tr>
    </table>
    <br> <br>
    <ejs-gantt ref='gantt' id="GanttContainer" :dataSource="data"
:taskFields = "taskFields" :height="height" :columns="columns"
:splitterSettings = "splitterSettings" :allowFiltering= "true"></ejs-gantt>
  </div>
</template>
<script>
import Vue from "vue";
import { GanttPlugin, Filter, Toolbar } from "@syncfusion/ej2-vue-gantt";
import { DropDownListPlugin } from "@syncfusion/ej2-vue-dropdowns";
import { projectNewData } from './data-source.js';
Vue.use(GanttPlugin);
Vue.use(DropDownListPlugin);
export default {
  data: function() {
    return{
      data: projectNewData,
      height: '450px',
```

```

        columns: [
            { field: 'TaskID', headerText: 'Task ID', textAlign: 'Left',
width: '100' },
            { field: 'TaskName', headerText: 'Task Name', width: '250' },
            { field: 'StartDate', headerText: 'Start Date', width: '150' },
            { field: 'Duration', headerText: 'Duration', width: '150' },
            { field: 'Progress', headerText: 'Progress', width: '150' },
        ],
        taskFields: {
            id: 'TaskID',
            name: 'TaskName',
            startDate: 'StartDate',
            duration: 'Duration',
            progress: 'Progress',
            child: 'subtasks'
        },
        splitterSettings: {
            columnIndex: 3
        },
        dataSource: [
            { id: 'Parent', mode: 'Parent' },
            { id: 'Child', mode: 'Child' },
            { id: 'Both', mode: 'Both' },
            { id: 'None', mode: 'None' },
        ],
        fields: { text: 'mode', value: 'id' },
    };
},
provide: {
    gantt: [ Filter ]
},
methods: {
    change: function (e) {
        var ganttChart =
document.getElementById('GanttContainer').ej2_instances[0];
        var mode = e.value;
        ganttChart.filterSettings.hierarchyMode = mode;
        ganttChart.clearFiltering();
    }
},
};
</script>

```

```
{% previewsample "page.domainurl/code-snippet/gantt/filtering-cs3" %}
```

#### Initial filter

To apply the filter at initial rendering, set the filter to predicate object in the [filterSettings.columns](#) property.

#### APP.VUE

```

<template>
  <div>
    <ejs-gantt ref='gantt' id="GanttContainer" :dataSource="data"
:taskFields = "taskFields" :height = "height" :columns="columns"

```

```

:splitterSettings = "splitterSettings" :filterSettings="filterSettings"
:allowFiltering= "true"></ejs-gantt>
</div>
</template>
<script>
import Vue from "vue";
import { GanttPlugin, Filter } from "@syncfusion/ej2-vue-gantt";
Vue.use(GanttPlugin);
export default {
  data: function() {
    return{
      data: [
        {
          TaskID: 1,
          TaskName: 'Project initiation',
          StartDate: new Date('04/02/2019'),
          EndDate: new Date('04/21/2019'),
          subtasks: [
            {TaskID: 2, TaskName: 'Identify site location', StartDate: new
Date('04/02/2019'), Duration: 0,Progress: 50, resources: [1]},
            {TaskID: 3, TaskName: 'Perform soil test', StartDate: new
Date('04/02/2019'), Duration: 4, Predecessor: '2',Progress: 50, resources:
[2, 3, 5]},
            {TaskID: 4, TaskName: 'Soil test approval', StartDate: new
Date('04/02/2019'), Duration: 0, Predecessor: '3', Progress: 50 },
          ]
        },
        {
          TaskID: 5,
          TaskName: 'Project estimation',
          StartDate: new Date('04/02/2019'),
          EndDate: new Date('04/21/2019'),
          subtasks: [
            {TaskID: 6, TaskName: 'Develop floor plan for estimation',
StartDate: new Date('04/04/2019'),Duration: 3, Predecessor: '4', Progress:
50, resources: [4]},
            {TaskID: 7, TaskName: 'List materials', StartDate: new
Date('04/04/2019'),Duration: 3, Predecessor: '6', resources: [4,
8],Progress: 50},
            {TaskID: 8, TaskName: 'Estimation approval', StartDate: new
Date('04/04/2019'),Duration: 0, Predecessor: '7', resources: [12, 5]
          }
        ]
      }
    ];
    height: '450px',
    columns: [
      { field: 'TaskID', headerText: 'Task ID', textAlign: 'Left',
width: '100' },
      { field: 'TaskName', headerText: 'Task Name', width: '250' },
      { field: 'StartDate', headerText: 'Start Date', width: '150' },
      { field: 'Duration', headerText: 'Duration', width: '150' },
      { field: 'Progress', headerText: 'Progress', width: '150' },
    ],
    taskFields: {
      id: 'TaskID',
      name: 'TaskName',
      startDate: 'StartDate',

```

```

        duration: 'Duration',
        progress: 'Progress',
        child: 'subtasks'
    },
    splitterSettings: {
        columnIndex: 3
    },
    filterSettings: {
        columns: [{ field: 'TaskName', matchCase: false, operator:
'startswith', predicate: 'and', value: 'Identify' },
        { field: 'TaskID', matchCase: false, operator: 'equal',
predicate: 'and', value: 2 }]
    },
    };
    },
    provide: {
        gantt: [ Filter ]
    }
    };
</script>

```

{% previewsample "page.domainurl/code-snippet/gantt/filtering-cs4" %}

### Filter operators

The filter operator for a column can be defined in the `filterSettings.columns.operator` property.

The available operators and its supported data types are:

Operator	Description	Supported Types
startswith	Checks whether the value begins with the specified value.	String
endswith	Checks whether the value ends with the specified value.	String
contains	Checks whether the value contains the specified value.	String
equal	Checks whether the value is equal to the specified value.	String &#124; Number &#124; Boolean &#124; Date
notequal	Checks for values not equal to the specified value.	String &#124; Number &#124; Boolean &#124; Date
greaterthan	Checks whether the value is greater than the specified value.	Number &#124; Date
greaterthanorequal	Checks whether a value is greater than or equal to the specified value.	Number &#124; Date
lessthan	Checks whether the value is less than the specified value.	Number &#124; Date
lessthanorequal	Checks whether the value is less than or equal to the specified value.	Number &#124; Date

By default, the `filterSettings.columns.operator` value is `equal`.

### Diacritics

By default, the Gantt component ignores the diacritic characters while filtering. To include diacritic characters, set the `filterSettings.ignoreAccent` to `true`.

In the following sample, type **Project** in the **TaskName** column to filter diacritic characters.

### APP.VUE

```
<template>
  <div>
    <ejs-gantt ref='gantt' id="GanttContainer" :dataSource="data"
    :taskFields = "taskFields" :height = "height" :columns="columns"
    :splitterSettings = "splitterSettings" :filterSettings="filterSettings"
    :allowFiltering= "true"></ejs-gantt>
  </div>
</template>
<script>
import Vue from "vue";
import { GanttPlugin, Filter } from "@syncfusion/ej2-vue-gantt";
import { projectNewData } from './data-source.js';
Vue.use(GanttPlugin);
export default {
  data: function() {
    return{
      data: [
        {
          TaskID: 1,
          TaskName: 'Project initiation',
          StartDate: new Date('04/02/2019'),
          EndDate: new Date('04/21/2019'),
          subtasks: [
            {TaskID: 2, TaskName: 'Identify site location', StartDate: new
            Date('04/02/2019'), Duration: 0,Progress: 50, resources: [1]},
            {TaskID: 3, TaskName: 'Perform soil test', StartDate: new
            Date('04/02/2019'), Duration: 4, Predecessor: '2',Progress: 50, resources:
            [2, 3, 5]},
            {TaskID: 4, TaskName: 'Soil test approval', StartDate: new
            Date('04/02/2019'), Duration: 0, Predecessor: '3', Progress: 50 },
          ]
        },
        {
          TaskID: 5,
          TaskName: 'Project estimation',
          StartDate: new Date('04/02/2019'),
          EndDate: new Date('04/21/2019'),
          subtasks: [
            {TaskID: 6, TaskName: 'Develop floor plan for estimation',
            StartDate: new Date('04/04/2019'),Duration: 3, Predecessor: '4', Progress:
            50, resources: [4]},
            {TaskID: 7, TaskName: 'List materials', StartDate: new
            Date('04/04/2019'),Duration: 3, Predecessor: '6', resources: [4,
            8],Progress: 50},
            {TaskID: 8, TaskName: 'Estimation approval', StartDate: new
            Date('04/04/2019'),Duration: 0, Predecessor: '7', resources: [12, 5]
            }
          ]
        }
      ]
    },
    height: '450px',
    columns: [
      { field: 'TaskID', headerText: 'Task ID', textAlign: 'Left',
        width: '100' },
    ]
  }
}
```

```

        { field: 'TaskName', headerText: 'Task Name', width: '250' },
        { field: 'StartDate', headerText: 'Start Date', width: '150' },
        { field: 'Duration', headerText: 'Duration', width: '150' },
        { field: 'Progress', headerText: 'Progress', width: '150' },
    ],
    taskFields: {
        id: 'TaskID',
        name: 'TaskName',
        startDate: 'StartDate',
        duration: 'Duration',
        progress: 'Progress',
        child: 'subtasks'
    },
    splitterSettings: {
        columnIndex: 3
    },
    filterSettings: {
        ignoreAccent: true
    },
    };
},
provide: {
    gantt: [ Filter ]
}
};
</script>

```

{% previewsample "page.domainurl/code-snippet/gantt/filtering-cs5" %}

#### *Filtering a specific column by method*

You can filter the columns dynamically by using the [filterByColumn](#) method.

#### **APP.VUE**

```

<template>
  <div>
    <ejs-button id="filterRecord" cssClass="e-info" v-
on:click.native="filter">Filter</ejs-button>
    <br><br><br>
    <ejs-gantt ref='gantt' id="GanttContainer" :dataSource="data"
:taskFields = "taskFields" :height = "height" :splitterSettings =
"splitterSettings" :columns="columns" :allowFiltering= "true"></ejs-gantt>
  </div>
</template>
<script>
import Vue from "vue";
import { GanttPlugin, Filter } from "@syncfusion/ej2-vue-gantt";
import { ButtonPlugin } from "@syncfusion/ej2-vue-buttons";
import { projectNewData } from '../data-source.js';
Vue.use(GanttPlugin);
Vue.use(ButtonPlugin);
export default {
  data: function() {
    return {
      data: [
        {

```

```

        TaskID: 1,
        TaskName: 'Project Initiation',
        StartDate: new Date('04/02/2019'),
        EndDate: new Date('04/21/2019'),
        subtasks: [
            { TaskID: 2, TaskName: 'Identify Site location', StartDate:
new Date('04/02/2019'), Duration: 4, Progress: 50 },
            { TaskID: 3, TaskName: 'Perform Soil test', StartDate: new
Date('04/02/2019'), Duration: 4, Progress: 50 },
            { TaskID: 4, TaskName: 'Soil test approval', StartDate: new
Date('04/02/2019'), Duration: 4, Progress: 50 },
        ]
    },
    {
        TaskID: 5,
        TaskName: 'Project Estimation',
        StartDate: new Date('04/02/2019'),
        EndDate: new Date('04/21/2019'),
        subtasks: [
            { TaskID: 6, TaskName: 'Develop floor plan for estimation',
StartDate: new Date('04/04/2019'), Duration: 3, Progress: 50 },
            { TaskID: 7, TaskName: 'List materials', StartDate: new
Date('04/04/2019'), Duration: 3, Progress: 50 },
            { TaskID: 8, TaskName: 'Estimation approval', StartDate: new
Date('04/04/2019'), Duration: 3, Progress: 50 }
        ]
    },
],
    height: '450px',
    columns: [
        { field: 'TaskID', headerText: 'Task ID', textAlign: 'Left',
width: '100' },
        { field: 'TaskName', headerText: 'Task Name', width: '250' },
        { field: 'StartDate', headerText: 'Start Date', width: '150' },
        { field: 'Duration', headerText: 'Duration', width: '150' },
        { field: 'Progress', headerText: 'Progress', width: '150' },
    ],
    taskFields: {
        id: 'TaskID',
        name: 'TaskName',
        startDate: 'StartDate',
        duration: 'Duration',
        progress: 'Progress',
        child: 'subtasks'
    },
    splitterSettings: {
        columnIndex: 3
    },
};

},
provide: {
    gantt: [ Filter ]
},
methods: {
    filter: function(e) {
        var ganttObj =
document.getElementById('GanttContainer').ej2_instances[0];

```



```

        ganttObj.filterByColumn('TaskName','startswith','Iden','and');
    }
}
};
</script>

```

{% previewsample "page.domainurl/code-snippet/gantt/filtering-cs6" %}

### Clear filtered columns

You can clear all the filtering condition done in the Gantt component by using the [clearFiltering](#) method.

The following code snippet explains the above behavior.

### APP.VUE

```

<template>
  <div>
    <ejs-button id="ClearfilterRecord" cssClass="e-info" v-
on:click.native="filter">Clear Filter</ejs-button>
    <br><br><br>
    <ejs-gantt ref='gantt' id="GanttContainer" :dataSource="data"
:taskFields = "taskFields" :height = "height" :columns="columns"
:splitterSettings = "splitterSettings" :filterSettings="filterSettings"
:allowFiltering= "true"></ejs-gantt>
  </div>
</template>
<script>
import Vue from "vue";
import { GanttPlugin, Filter } from "@syncfusion/ej2-vue-gantt";
import { ButtonPlugin } from "@syncfusion/ej2-vue-buttons";
import { projectNewData } from './data-source.js';
Vue.use(GanttPlugin);
Vue.use(ButtonPlugin);
export default {
  data: function() {
    return{
      data: [
        {
          TaskID: 1,
          TaskName: 'Project Initiation',
          StartDate: new Date('04/02/2019'),
          EndDate: new Date('04/21/2019'),
          subtasks: [
            { TaskID: 2, TaskName: 'Identify Site location', StartDate:
new Date('04/02/2019'), Duration: 4, Progress: 50 },
            { TaskID: 3, TaskName: 'Perform Soil test', StartDate: new
Date('04/02/2019'), Duration: 4, Progress: 50 },
            { TaskID: 4, TaskName: 'Soil test approval', StartDate: new
Date('04/02/2019'), Duration: 4, Progress: 50 },
          ]
        },
        {
          TaskID: 5,
          TaskName: 'Project Estimation',
          StartDate: new Date('04/02/2019'),
          EndDate: new Date('04/21/2019'),
          subtasks: [

```

```

        { TaskID: 6, TaskName: 'Develop floor plan for estimation',
        StartDate: new Date('04/04/2019'), Duration: 3, Progress: 50 },
        { TaskID: 7, TaskName: 'List materials', StartDate: new
        Date('04/04/2019'), Duration: 3, Progress: 50 },
        { TaskID: 8, TaskName: 'Estimation approval', StartDate: new
        Date('04/04/2019'), Duration: 3, Progress: 50 }
    ]
},
],
    height: '450px',
    columns: [
        { field: 'TaskID', headerText: 'Task ID', textAlign: 'Left',
width: '100' },
        { field: 'TaskName', headerText: 'Task Name', width: '250' },
        { field: 'StartDate', headerText: 'Start Date', width: '150' },
        { field: 'Duration', headerText: 'Duration', width: '150' },
        { field: 'Progress', headerText: 'Progress', width: '150' },
    ],
    taskFields: {
        id: 'TaskID',
        name: 'TaskName',
        startDate: 'StartDate',
        duration: 'Duration',
        progress: 'Progress',
        child: 'subtasks'
    },
    splitterSettings: {
        columnIndex: 3
    },
    filterSettings: {
        columns: [{ field: 'TaskName', matchCase: false, operator:
'startswith', predicate: 'and', value: 'Identify' },
        { field: 'Progress', matchCase: false, operator: 'equal',
predicate: 'and', value: 50 }],
    }
};
},
provide: {
    gantt: [ Filter ]
},
methods: {
    filter: function(e) {
        var ganttObj =
document.getElementById('GanttContainer').ej2_instances[0];
        ganttObj.clearFiltering();
    }
}
};
</script>

```

{% previewsample "page.domainurl/code-snippet/gantt/filtering-cs7" %}

[Custom component in filter menu](#)

The [column.filter.ui](#) is used to add custom filter components to a particular column.

To implement a custom filter UI, define the following functions:

- **create**: Creates a custom component.
- **write**: Wire events for a custom component.
- **read**: Read the filter value from the custom component.

In the following sample, the dropdown is used as a custom component in the TaskName column.

#### APP.VUE

```
<template>
  <div>
    <ejs-gantt ref='gantt' id="GanttContainer" :dataSource="data"
    :taskFields = "taskFields" :height = "height" :columns="columns"
    :splitterSettings = "splitterSettings" :filterSettings="filterSettings"
    :allowFiltering= true>
      <e-columns>
        <e-column field='TaskID' headerText='Task ID' textAlign='Left'
width='100'></e-column>
        <e-column field='TaskName' :filter= 'filter' headerText='Task
Name' width='150'></e-column>
        <e-column field='StartDate' headerText='Start Date'
width='150'></e-column>
        <e-column field='Duration' headerText='Duration' width='150'></e-
column>
        <e-column field='Progress' headerText='Progress' width='150'></e-
column>
      </e-columns>
    </ejs-gantt>
  </div>
</template>
<script>
import Vue from "vue";
import { GanttPlugin, Filter, } from "@syncfusion/ej2-vue-gantt";
import { projectNewData } from './data-source.js';
Vue.use(GanttPlugin);
import { DropDownListPlugin } from '@syncfusion/ej2-vue-dropdowns';
Vue.use(DropDownListPlugin);
import { DataManager } from "@syncfusion/ej2-data";
import { createElement } from "@syncfusion/ej2-base";
export default {
  data: function() {
    return{
      data: [
        {
          TaskID: 1,
          TaskName: 'Project initiation',
          StartDate: new Date('04/02/2019'),
          EndDate: new Date('04/21/2019'),
          subtasks: [
            {TaskID: 2, TaskName: 'Identify site location', StartDate: new
Date('04/02/2019'), Duration: 0,Progress: 50, resources: [1]},
            {TaskID: 3, TaskName: 'Perform soil test', StartDate: new
Date('04/02/2019'), Duration: 4, Predecessor: '2',Progress: 50, resources:
[2, 3, 5]},
            {TaskID: 4, TaskName: 'Soil test approval', StartDate: new
Date('04/02/2019'), Duration: 0, Predecessor: '3', Progress: 50 },
          ]
        }
      ]
    }
  }
}
```

```

    },
    {
        TaskID: 5,
        TaskName: 'Project estimation',
        StartDate: new Date('04/02/2019'),
        EndDate: new Date('04/21/2019'),
        subtasks: [
            {TaskID: 6, TaskName: 'Develop floor plan for estimation',
            StartDate: new Date('04/04/2019'), Duration: 3, Predecessor: '4', Progress:
            50, resources: [4]},
            {TaskID: 7, TaskName: 'List materials', StartDate: new
            Date('04/04/2019'), Duration: 3, Predecessor: '6', resources: [4,
            8], Progress: 50},
            {TaskID: 8, TaskName: 'Estimation approval', StartDate: new
            Date('04/04/2019'), Duration: 0, Predecessor: '7', resources: [12, 5]}
        ]
    }
  ]],
  height: '450px',
  columns: [
    { field: 'TaskID', headerText: 'Task ID', textAlign: 'Left',
width: '100' },
    { field: 'TaskName', headerText: 'Task Name', width: '250' },
    { field: 'StartDate', headerText: 'Start Date', width: '150' },
    { field: 'Duration', headerText: 'Duration', width: '150' },
    { field: 'Progress', headerText: 'Progress', width: '150' }
  ],
  taskFields: {
    id: 'TaskID',
    name: 'TaskName',
    startDate: 'StartDate',
    duration: 'Duration',
    progress: 'Progress',
    child: 'subtasks'
  },
  filter: {
    ui: {
      create: function (args) {
        let db = new DataManager(data);
        let flValInput = createElement('input', { className: 'flm-
input' });

        args.target.appendChild(flValInput);
        dropInstance = new DropDownList({
          dataSource: new DataManager(data),
          fields: { text: 'TaskName', value: 'TaskName' },
          placeholder: 'Select a value',
          popupHeight: '200px'
        });
        dropInstance.appendTo(flValInput);
      },
      write: function (args) {
        dropInstance.value = args.filteredValue;
      },
      read: function (args) {
        args.fltrObj.filterByColumn(args.column.field,
args.operator, dropInstance.value);
      }
    }
  }
}

```

```

    },
    splitterSettings:{
        columnIndex:3
    },
    filterSettings: {
        type:'Menu'
    },
    };
},
provide: {
    gantt: [ Filter]
}
};
</script>

```

{% previewsample "page.domainurl/code-snippet/gantt/filtering-cs8" %}

### Filter menu in Vue Gantt component

The Gantt Component provides the menu filtering support for each column. You can enable the filter menu by setting the `allowFiltering` to `true`. The filter menu UI will be rendered based on its column type, which allows you to filter data. You can filter the records with different operators.

### APP.VUE

```

<template>
  <div>
    <ejs-gantt ref='gantt' id="GanttContainer" :dataSource="data"
    :taskFields = "taskFields" :height = "height" :columns="columns"
    :splitterSettings = "splitterSettings" :filterSettings="filterSettings"
    :allowFiltering= "true"></ejs-gantt>
  </div>
</template>
<script>
import Vue from "vue";
import { GanttPlugin, Filter, } from "@syncfusion/ej2-vue-gantt";
import { projectNewData } from './data-source.js';
Vue.use(GanttPlugin);
export default {
  data: function() {
    return{
      data: [
        {
          TaskID: 1,
          TaskName: 'Project initiation',
          StartDate: new Date('04/02/2019'),
          EndDate: new Date('04/21/2019'),
          subtasks: [
            {TaskID: 2, TaskName: 'Identify site location', StartDate: new
            Date('04/02/2019'), Duration: 0,Progress: 50, resources: [1]},
            {TaskID: 3, TaskName: 'Perform soil test', StartDate: new
            Date('04/02/2019'), Duration: 4, Predecessor: '2',Progress: 50, resources:
            [2, 3, 5]},
            {TaskID: 4, TaskName: 'Soil test approval', StartDate: new
            Date('04/02/2019'), Duration: 0, Predecessor: '3', Progress: 50 },
          ]
        },
      ],
    },
  },

```

```

{
  TaskID: 5,
  TaskName: 'Project estimation',
  StartDate: new Date('04/02/2019'),
  EndDate: new Date('04/21/2019'),
  subtasks: [
    {TaskID: 6, TaskName: 'Develop floor plan for estimation',
  StartDate: new Date('04/04/2019'),Duration: 3, Predecessor: '4', Progress:
  50, resources: [4]},
    {TaskID: 7, TaskName: 'List materials', StartDate: new
  Date('04/04/2019'),Duration: 3, Predecessor: '6', resources: [4,
  8],Progress: 50},
    {TaskID: 8, TaskName: 'Estimation approval', StartDate: new
  Date('04/04/2019'),Duration: 0, Predecessor: '7', resources: [12, 5]}
  ],
  height: '450px',
  columns: [
    { field: 'TaskID', headerText: 'Task ID', textAlign: 'Left',
width: '100' },
    { field: 'TaskName', headerText: 'Task Name', width: '250' },
    { field: 'StartDate', headerText: 'Start Date', width: '150' },
    { field: 'Duration', headerText: 'Duration', width: '150' },
    { field: 'Progress', headerText: 'Progress', width: '150' }
  ],
  taskFields: {
    id: 'TaskID',
    name: 'TaskName',
    startDate: 'StartDate',
    duration: 'Duration',
    progress: 'Progress',
    child: 'subtasks'
  },
  splitterSettings:{
    columnIndex:3
  },
  filterSettings: {
    type:'Menu'
  },
  };
},
provide: {
  gantt: [ Filter]
}
};
</script>

```

{% previewsample "page.domainurl/code-snippet/gantt/filtering-cs2" %}

[allowFiltering](#) must be set as `true` to enable filter menu.

Setting [columns.allowFiltering](#) as `false` will prevent filter menu rendering for a particular column.

Excel like filter in Vue Gantt component

You can enable Excel like filter by defining [filterSettings.type](#) as `Excel`. The excel menu contains an option such as Sorting, Clear filter, Sub menu for advanced filtering.

**APP.VUE**

```

<template>
  <div>
    <ejs-gantt ref='gantt' id="GanttContainer" :dataSource="data"
    :taskFields = "taskFields" :height = "height" :columns="columns"
    :splitterSettings = "splitterSettings" :filterSettings="filterSettings"
    :allowFiltering= "true"></ejs-gantt>
  </div>
</template>
<script>
import Vue from "vue";
import { GanttPlugin, Filter, } from "@syncfusion/ej2-vue-gantt";
import { projectNewData } from './data-source.js';
Vue.use(GanttPlugin);
export default {
  data: function() {
    return{
      data: [
        {
          TaskID: 1,
          TaskName: 'Project initiation',
          StartDate: new Date('04/02/2019'),
          EndDate: new Date('04/21/2019'),
          subtasks: [
            {TaskID: 2, TaskName: 'Identify site location', StartDate: new
Date('04/02/2019'), Duration: 0,Progress: 50, resources: [1]},
            {TaskID: 3, TaskName: 'Perform soil test', StartDate: new
Date('04/02/2019'), Duration: 4, Predecessor: '2',Progress: 50, resources:
[2, 3, 5]},
            {TaskID: 4, TaskName: 'Soil test approval', StartDate: new
Date('04/02/2019'), Duration: 0, Predecessor: '3', Progress: 50 },
          ]
        },
        {
          TaskID: 5,
          TaskName: 'Project estimation',
          StartDate: new Date('04/02/2019'),
          EndDate: new Date('04/21/2019'),
          subtasks: [
            {TaskID: 6, TaskName: 'Develop floor plan for estimation',
StartDate: new Date('04/04/2019'),Duration: 3, Predecessor: '4', Progress:
50, resources: [4]},
            {TaskID: 7, TaskName: 'List materials', StartDate: new
Date('04/04/2019'),Duration: 3, Predecessor: '6', resources: [4,
8],Progress: 50},
            {TaskID: 8, TaskName: 'Estimation approval', StartDate: new
Date('04/04/2019'),Duration: 0, Predecessor: '7', resources: [12, 5]}
          ]
        }
      ],
      height: '450px',
      columns: [
        { field: 'TaskID', headerText: 'Task ID', textAlign: 'Left',
width: '100' },
        { field: 'TaskName', headerText: 'Task Name', width: '250' },
        { field: 'StartDate', headerText: 'Start Date', width: '150' },
        { field: 'Duration', headerText: 'Duration', width: '150' },
      ]
    }
  }
}

```

```

        { field: 'Progress', headerText: 'Progress', width: '150' },
    ],
    taskFields: {
        id: 'TaskID',
        name: 'TaskName',
        startDate: 'StartDate',
        duration: 'Duration',
        progress: 'Progress',
        child: 'subtasks'
    },
    splitterSettings: {
        columnIndex: 3
    },
    filterSettings: {
        type: 'Excel'
    },
};
},
provide: {
    gantt: [ Filter ]
}
};
</script>

```

{% previewsample "page.domainurl/code-snippet/gantt/filtering-cs1" %}

### Searching in Vue Gantt component

You can search records in the Gantt component by using the [search](#) method with search key as a parameter. The Gantt component provides an option to integrate the search text box in the toolbar by adding the search item to the [toolbar](#) property.

To search records, inject the [Filter](#) module in the `provide` section.

### APP.VUE

```

<template>
  <div>
    <ejs-gantt ref='gantt' id="GanttContainer" :dataSource="data"
    :taskFields = "taskFields" :height = "height" :splitterSettings =
    "splitterSettings" :toolbar="toolbar"></ejs-gantt>
  </div>
</template>
<script>
import Vue from "vue";
import { GanttPlugin, Toolbar, Filter } from "@syncfusion/ej2-vue-gantt";
Vue.use(GanttPlugin);
export default {
  data: function() {
    return {
      data: [
        {
          TaskID: 1,
          TaskName: 'Project initiation',
          StartDate: new Date('04/02/2019'),
          EndDate: new Date('04/21/2019'),
          subtasks: [

```



```

        {TaskID: 2, TaskName: 'Identify site location', StartDate: new
Date('04/02/2019'), Duration: 0,Progress: 50, resources: [1]},
        {TaskID: 3, TaskName: 'Perform soil test', StartDate: new
Date('04/02/2019'), Duration: 4, Predecessor: '2',Progress: 50, resources:
[2, 3, 5]},
        {TaskID: 4, TaskName: 'Soil test approval', StartDate: new
Date('04/02/2019'), Duration: 0, Predecessor: '3', Progress: 50 },
    ]
},
{
    TaskID: 5,
    TaskName: 'Project estimation',
    StartDate: new Date('04/02/2019'),
    EndDate: new Date('04/21/2019'),
    subtasks: [
        {TaskID: 6, TaskName: 'Develop floor plan for estimation',
StartDate: new Date('04/04/2019'),Duration: 3, Predecessor: '4', Progress:
50, resources: [4]},
        {TaskID: 7, TaskName: 'List materials', StartDate: new
Date('04/04/2019'),Duration: 3, Predecessor: '6', resources: [4,
8],Progress: 50},
        {TaskID: 8, TaskName: 'Estimation approval', StartDate: new
Date('04/04/2019'),Duration: 0, Predecessor: '7', resources: [12, 5]}
    ]
}],
    height: '450px',
    taskFields: {
        id: 'TaskID',
        name: 'TaskName',
        startDate: 'StartDate',
        endDate: 'EndDate',
        duration: 'Duration',
        progress: 'Progress',
        dependency: 'Predecessor',
        child: 'subtasks'
    },
    toolbar: ['Search'],
    splitterSettings:{
        columnIndex:3
    },
};
},
provide: {
    gantt: [ Toolbar, Filter ]
}
};
</script>

```

{% previewsample "page.domainurl/code-snippet/gantt/filtering-cs9" %}

### Initial search

In the Gantt component, you can load a task with some search criteria and this can be done by using the [searchSettings](#) property. To apply search at initial rendering, set the value for [fields](#), [operator](#), [key](#), and [ignoreCase](#) in the [searchSettings](#) property.

### APP.VUE

```

<template>
  <div>
    <ejs-gantt ref='gantt' id="GanttContainer" :dataSource="data"
:taskFields = "taskFields" :height = "height" :splitterSettings =
"splitterSettings" :columns="columns" :toolbar="toolbar"
:searchSettings="searchSettings"></ejs-gantt>
  </div>
</template>
<script>
import Vue from "vue";
import { GanttPlugin, Toolbar, Filter } from "@syncfusion/ej2-vue-gantt";
Vue.use(GanttPlugin);
export default {
  data: function() {
    return{
      data: [
        {
          TaskID: 1,
          TaskName: 'Project initiation',
          StartDate: new Date('04/02/2019'),
          EndDate: new Date('04/21/2019'),
          subtasks: [
            {TaskID: 2, TaskName: 'Identify site location', StartDate: new
Date('04/02/2019'), Duration: 0,Progress: 50, resources: [1]},
            {TaskID: 3, TaskName: 'Perform soil test', StartDate: new
Date('04/02/2019'), Duration: 4, Predecessor: '2',Progress: 50, resources:
[2, 3, 5]},
            {TaskID: 4, TaskName: 'Soil test approval', StartDate: new
Date('04/02/2019'), Duration: 0, Predecessor: '3', Progress: 50 },
          ]
        },
        {
          TaskID: 5,
          TaskName: 'Project estimation',
          StartDate: new Date('04/02/2019'),
          EndDate: new Date('04/21/2019'),
          subtasks: [
            {TaskID: 6, TaskName: 'Develop floor plan for estimation',
StartDate: new Date('04/04/2019'),Duration: 3, Predecessor: '4', Progress:
50, resources: [4]},
            {TaskID: 7, TaskName: 'List materials', StartDate: new
Date('04/04/2019'),Duration: 3, Predecessor: '6', resources: [4,
8],Progress: 50},
            {TaskID: 8, TaskName: 'Estimation approval', StartDate: new
Date('04/04/2019'),Duration: 0, Predecessor: '7', resources: [12, 5]}
          ]
        }
      ],
      height: '450px',
      columns: [
        { field: 'TaskID', headerText: 'Task ID', textAlign: 'Left',
width: '100' },
        { field: 'TaskName', headerText: 'Task Name', width: '250'
},
        { field: 'StartDate', headerText: 'Start Date', width: '150'
},
        { field: 'Duration', headerText: 'Duration', width: '150' },
        { field: 'Progress', headerText: 'Progress', width: '150' },
      ]
    }
  }
}

```

```

    ],
    taskFields: {
      id: 'TaskID',
      name: 'TaskName',
      startDate: 'StartDate',
      duration: 'Duration',
      progress: 'Progress',
      child: 'subtasks'
    },
    toolbar: ['Search'],
    splitterSettings: {
      columnIndex: 3
    },
    searchSettings: { fields: ['TaskName'], operator: 'contains',
key: 'List', ignoreCase: true }    };
  },
  provide: {
    gantt: [ Toolbar, Filter ]
  }
};
</script>

```

{% previewsample "page.domainurl/code-snippet/gantt/filtering-cs10" %}

By default, Gantt searches all the bound column values. To customize this behavior, define the [searchSettings.fields](#) property.

#### Search operators

The search operator can be defined in the [searchSettings.operator](#) property to configure specific searching.

The following operators are supported in searching:

#### Operator | Description

startsWith | Checks whether a value begins with the specified value.

endsWith | Checks whether a value ends with the specified value.

contains | Checks whether a value contains the specified value.

equal | Checks whether a value is equal to the specified value.

notEqual | Checks for values not equal to the specified value.

By default, the [searchSettings.operator](#) value is `contains`.

#### Search by external button

To search the Gantt records from an external button, invoke the [search](#) method.

#### APP.VUE

```

<template>
  <div>
    <ejs-button id="searchRecord" cssClass="e-info" v-
on:click.native="search">Search</ejs-button>
    <br><br><br>
  </div>
</template>

```

```

    <ejs-gantt ref='gantt' id="GanttContainer" :dataSource="data"
:taskFields = "taskFields" :height = "height" :toolbar="toolbar"
:splitterSettings = "splitterSettings" :columns="columns"
:allowFiltering='true'></ejs-gantt>
  </div>
</template>
<script>
import Vue from "vue";
import { GanttPlugin, Toolbar, Filter } from "@syncfusion/ej2-vue-gantt";
import { ButtonPlugin } from "@syncfusion/ej2-vue-buttons";
Vue.use(GanttPlugin);
Vue.use(ButtonPlugin);
export default {
  data: function() {
    return{
      data: [
        {
          TaskID: 1,
          TaskName: 'Project initiation',
          StartDate: new Date('04/02/2019'),
          EndDate: new Date('04/21/2019'),
          subtasks: [
            {TaskID: 2, TaskName: 'Identify site location', StartDate: new
Date('04/02/2019'), Duration: 0,Progress: 50, resources: [1]},
            {TaskID: 3, TaskName: 'Perform soil test', StartDate: new
Date('04/02/2019'), Duration: 4, Predecessor: '2',Progress: 50, resources:
[2, 3, 5]},
            {TaskID: 4, TaskName: 'Soil test approval', StartDate: new
Date('04/02/2019'), Duration: 0, Predecessor: '3', Progress: 50 },
          ]
        },
        {
          TaskID: 5,
          TaskName: 'Project estimation',
          StartDate: new Date('04/02/2019'),
          EndDate: new Date('04/21/2019'),
          subtasks: [
            {TaskID: 6, TaskName: 'Develop floor plan for estimation',
StartDate: new Date('04/04/2019'),Duration: 3, Predecessor: '4', Progress:
50, resources: [4]},
            {TaskID: 7, TaskName: 'List materials', StartDate: new
Date('04/04/2019'),Duration: 3, Predecessor: '6', resources: [4,
8],Progress: 50},
            {TaskID: 8, TaskName: 'Estimation approval', StartDate: new
Date('04/04/2019'),Duration: 0, Predecessor: '7', resources: [12, 5]}
          ]
        }
      ]],
      height: '450px',
      columns: [
        { field: 'TaskID', headerText: 'Task ID', textAlign: 'Left',
width: '100' },
        { field: 'TaskName', headerText: 'Task Name', width: '250'
},
        { field: 'StartDate', headerText: 'Start Date', width: '150'
},
        { field: 'Duration', headerText: 'Duration', width: '150' },
        { field: 'Progress', headerText: 'Progress', width: '150' },

```

```

        ],
        taskFields: {
            id: 'TaskID',
            name: 'TaskName',
            startDate: 'StartDate',
            duration: 'Duration',
            progress: 'Progress',
            child: 'subtasks'
        },
        toolbar: ['Search'],
        splitterSettings: {
            columnIndex: 3
        },
    };
},
provide: {
    gantt: [ Toolbar, Filter ]
},
methods: {
    search: function(e) {
        var ganttObj =
document.getElementById('GanttContainer').ej2_instances[0];
        var searchText =
document.getElementById('GanttContainer_searchbar').value;
        ganttObj.search(searchText);
    }
}
};
</script>

```

{% previewsample "page.domainurl/code-snippet/gantt/filtering-cs11" %}

Note: You should set the [allowFiltering](#) property to `true` for searching the content externally.

#### Search specific columns

By default, the Gantt component searches all the columns. You can search specific columns by defining the specific column's field names in the [searchSettings.fields](#) property.

#### APP.VUE

```

<template>
  <div>
    <ejs-gantt ref='gantt' id="GanttContainer" :dataSource="data"
:taskFields = "taskFields" :height = "height" :toolbar="toolbar"
:splitterSettings = "splitterSettings" :columns="columns"
:searchSettings="searchSettings"></ejs-gantt>
  </div>
</template>
<script>
import Vue from "vue";
import { GanttPlugin, Toolbar, Filter } from "@syncfusion/ej2-vue-gantt";
Vue.use(GanttPlugin);
export default {
  data: function() {
    return {
      data: [

```

```

{
    TaskID: 1,
    TaskName: 'Project initiation',
    StartDate: new Date('04/02/2019'),
    EndDate: new Date('04/21/2019'),
    subtasks: [
        {TaskID: 2, TaskName: 'Identify site location', StartDate: new
Date('04/02/2019'), Duration: 0,Progress: 50, resources: [1]},
        {TaskID: 3, TaskName: 'Perform soil test', StartDate: new
Date('04/02/2019'), Duration: 4, Predecessor: '2',Progress: 50, resources:
[2, 3, 5]},
        {TaskID: 4, TaskName: 'Soil test approval', StartDate: new
Date('04/02/2019'), Duration: 0, Predecessor: '3', Progress: 50 },
    ]
},
{
    TaskID: 5,
    TaskName: 'Project estimation',
    StartDate: new Date('04/02/2019'),
    EndDate: new Date('04/21/2019'),
    subtasks: [
        {TaskID: 6, TaskName: 'Develop floor plan for estimation',
StartDate: new Date('04/04/2019'),Duration: 3, Predecessor: '4', Progress:
50, resources: [4]},
        {TaskID: 7, TaskName: 'List materials', StartDate: new
Date('04/04/2019'),Duration: 3, Predecessor: '6', resources: [4,
8],Progress: 50},
        {TaskID: 8, TaskName: 'Estimation approval', StartDate: new
Date('04/04/2019'),Duration: 0, Predecessor: '7', resources: [12, 5]}
    ]
}],
    height: '450px',
    columns: [
        { field: 'TaskID', headerText: 'Task ID', textAlign: 'Left',
width: '100' },
        { field: 'TaskName', headerText: 'Task Name', width: '250'
},
        { field: 'Duration', headerText: 'Duration', width: '150' },
        { field: 'StartDate', headerText: 'Start Date', width: '150'
},
        { field: 'Progress', headerText: 'Progress', width: '150' },
    ],
    taskFields: {
        id: 'TaskID',
        name: 'TaskName',
        startDate: 'StartDate',
        duration: 'Duration',
        progress: 'Progress',
        child: 'subtasks'
    },
    toolbar: ['Search'],
    searchSettings: { fields: ['TaskName', 'Duration'] },
    splitterSettings:{
        columnIndex:3
    },
};
},

```

```

    provide: {
      gantt: [ Toolbar, Filter ]
    },
  };
</script>

```

{% previewsample "page.domainurl/code-snippet/gantt/filtering-cs12" %}

In above sample, you can search only **TaskName** and **Duration** column values.

[Clear search by external button](#)

You can set [searchSettings.key](#) property as **empty** string, to clear the searched Gantt records from external button.

### APP.VUE

```

<template>
  <div>
    <ejs-button id="clearSearch" cssClass="e-info" v-
on:click.native="clearSearch">Clear Search</ejs-button>
    <br><br><br>
    <ejs-gantt ref='gantt' id="GanttContainer" :dataSource="data"
:taskFields = "taskFields" :height = "height" :splitterSettings =
"splitterSettings" :columns="columns" :toolbar="toolbar"
:searchSettings="searchSettings"></ejs-gantt>
  </div>
</template>
<script>
import Vue from "vue";
import { GanttPlugin, Toolbar, Filter } from "@syncfusion/ej2-vue-gantt";
import { ButtonPlugin } from "@syncfusion/ej2-vue-buttons";
Vue.use(GanttPlugin);
Vue.use(ButtonPlugin);
export default {
  data: function() {
    return{
      data: [
        {
          TaskID: 1,
          TaskName: 'Project initiation',
          StartDate: new Date('04/02/2019'),
          EndDate: new Date('04/21/2019'),
          subtasks: [
            {TaskID: 2, TaskName: 'Identify site location', StartDate: new
Date('04/02/2019'), Duration: 0,Progress: 50, resources: [1]},
            {TaskID: 3, TaskName: 'Perform soil test', StartDate: new
Date('04/02/2019'), Duration: 4, Predecessor: '2',Progress: 50, resources:
[2, 3, 5]},
            {TaskID: 4, TaskName: 'Soil test approval', StartDate: new
Date('04/02/2019'), Duration: 0, Predecessor: '3', Progress: 50 },
          ]
        },
        {
          TaskID: 5,
          TaskName: 'Project estimation',
          StartDate: new Date('04/02/2019'),

```

```

        EndDate: new Date('04/21/2019'),
        subtasks: [
            {TaskID: 6, TaskName: 'Develop floor plan for estimation',
StartDate: new Date('04/04/2019'),Duration: 3, Predecessor: '4', Progress:
50, resources: [4]},
            {TaskID: 7, TaskName: 'List materials', StartDate: new
Date('04/04/2019'),Duration: 3, Predecessor: '6', resources: [4,
8],Progress: 50},
            {TaskID: 8, TaskName: 'Estimation approval', StartDate: new
Date('04/04/2019'),Duration: 0, Predecessor: '7', resources: [12, 5]}
        ]
    }],
    height: '450px',
    columns: [
        { field: 'TaskID', headerText: 'Task ID', textAlign: 'Left',
width: '100' },
        { field: 'TaskName', headerText: 'Task Name', width: '250'
},
        { field: 'StartDate', headerText: 'Start Date', width: '150'
},
        { field: 'Duration', headerText: 'Duration', width: '150' },
        { field: 'Progress', headerText: 'Progress', width: '150' },
    ],
    taskFields: {
        id: 'TaskID',
        name: 'TaskName',
        startDate: 'StartDate',
        duration: 'Duration',
        progress: 'Progress',
        child: 'subtasks'
    },
    toolbar: ['Search'],
    splitterSettings:{
        columnIndex:3
    },
    searchSettings: { fields: ['TaskName'], operator: 'contains',
key: 'Perform', ignoreCase: true }    };
    },
    provide: {
        gantt: [ Toolbar, Filter ]
    },
    methods: {
        clearSearch: function(e){
            var ganttObj =
document.getElementById('GanttContainer').ej2_instances[0];
            ganttObj.searchSettings.key='';
        }
    }
};
</script>

```

{% previewsample "page.domainurl/code-snippet/gantt/filtering-cs13" %}



## Sorting in Vue Gantt component

Sorting enables you to sort data in the ascending or descending order. To sort a column, click the column header.

To sort multiple columns, press and hold the CTRL key and click the column header. You can clear sorting of any one of the multi-sorted columns by pressing and holding the SHIFT key and clicking the specific column header.

To enable sorting in the Gantt component, set the [allowSorting](#) property to `true`. Sorting options can be configured through the [sortSettings](#) property.

To sort data, inject the [Sort](#) module in the `provide` section.

### APP.VUE

```
<template>
  <div>
    <ejs-gantt ref='gantt' id="GanttContainer" :dataSource="data"
    :taskFields = "taskFields" :height = "height" :columns="columns"
    :splitterSettings= "splitterSettings" :allowSorting= 'true'></ejs-gantt>
  </div>
</template>
<script>
import Vue from "vue";
import { GanttPlugin, Sort } from "@syncfusion/ej2-vue-gantt";
Vue.use(GanttPlugin);
export default {
  data: function() {
    return{
      data: [
        {
          TaskID: 1,
          TaskName: 'Project Initiation',
          StartDate: new Date('04/02/2019'),
          EndDate: new Date('04/21/2019'),
          subtasks: [
            { TaskID: 2, TaskName: 'Identify Site location',
              StartDate: new Date('04/02/2019'), Duration: 4, Progress: 50 },
            { TaskID: 3, TaskName: 'Perform Soil test', StartDate:
              new Date('04/02/2019'), Duration: 4, Progress: 50 },
            { TaskID: 4, TaskName: 'Soil test approval', StartDate:
              new Date('04/02/2019'), Duration: 4, Progress: 50 },
          ]
        },
        {
          TaskID: 5,
          TaskName: 'Project Estimation',
          StartDate: new Date('04/02/2019'),
          EndDate: new Date('04/21/2019'),
          subtasks: [
            { TaskID: 6, TaskName: 'Develop floor plan for
              estimation', StartDate: new Date('04/04/2019'), Duration: 3, Progress: 50 },
            { TaskID: 7, TaskName: 'List materials', StartDate: new
              Date('04/04/2019'), Duration: 3, Progress: 50 },
            { TaskID: 8, TaskName: 'Estimation approval', StartDate:
              new Date('04/04/2019'), Duration: 3, Progress: 50 }
          ]
        }
      ]
    }
  }
}
```

```

    },
    ],
    height: '450px',
    taskFields: {
      id: 'TaskID',
      name: 'TaskName',
      startDate: 'StartDate',
      duration: 'Duration',
      progress: 'Progress',
      child: 'subtasks'
    },
    splitterSettings: {
      columnIndex: 3
    },
    columns: [
      { field: 'TaskID', headerText: 'Task ID', textAlign: 'Left',
width: '100' },
      { field: 'TaskName', headerText: 'Task Name', width: '250' },
      { field: 'StartDate', headerText: 'Start Date', width: '150' },
      { field: 'Duration', headerText: 'Duration', width: '150' },
      { field: 'Progress', headerText: 'Progress', width: '150' }
    ],
  ],
};
},
provide: {
  gantt: [ Sort ]
}
};
</script>

```

{% previewsample "page.domainurl/code-snippet/gantt/sorting-cs1" %}

\* Gantt columns are sorted in the ascending order. If you click the already sorted column, the sort direction toggles.

\* To disable sorting for a particular column, set the [columns.allowSorting](#) property to `false`.

#### Sorting column on Gantt initialization

The Gantt component can be rendered with sorted columns initially, and this can be achieved by using the [sortSettings](#) property. You can add columns that are sorted initially in the [sortSettings.columns](#) collection defined with [field](#) and [direction](#) properties. The following code example shows how to add the sorted column to Gantt initialization.

#### APP.VUE

```

<template>
  <div>
    <ejs-gantt ref='gantt' id="GanttContainer" :dataSource="data"
:taskFields = "taskFields" :height = "height" :columns="columns"
:splitterSettings= "splitterSettings" :sortSettings="sortSettings"
:allowSorting= 'true'></ejs-gantt>
  </div>
</template>
<script>

```

```

import Vue from "vue";
import { GanttPlugin, Sort } from "@syncfusion/ej2-vue-gantt";
Vue.use(GanttPlugin);
export default {
  data: function() {
    return {
      data: [
        {
          TaskID: 1,
          TaskName: 'Project Initiation',
          StartDate: new Date('04/02/2019'),
          EndDate: new Date('04/21/2019'),
          subtasks: [
            { TaskID: 2, TaskName: 'Identify Site location',
              StartDate: new Date('04/02/2019'), Duration: 4, Progress: 50 },
            { TaskID: 3, TaskName: 'Perform Soil test', StartDate:
              new Date('04/02/2019'), Duration: 4, Progress: 50 },
            { TaskID: 4, TaskName: 'Soil test approval', StartDate:
              new Date('04/02/2019'), Duration: 4, Progress: 50 },
          ]
        },
        {
          TaskID: 5,
          TaskName: 'Project Estimation',
          StartDate: new Date('04/02/2019'),
          EndDate: new Date('04/21/2019'),
          subtasks: [
            { TaskID: 6, TaskName: 'Develop floor plan for
              estimation', StartDate: new Date('04/04/2019'), Duration: 3, Progress: 50 },
            { TaskID: 7, TaskName: 'List materials', StartDate: new
              Date('04/04/2019'), Duration: 3, Progress: 50 },
            { TaskID: 8, TaskName: 'Estimation approval', StartDate:
              new Date('04/04/2019'), Duration: 3, Progress: 50 }
          ]
        }
      ],
      height: '450px',
      taskFields: {
        id: 'TaskID',
        name: 'TaskName',
        startDate: 'StartDate',
        duration: 'Duration',
        progress: 'Progress',
        child: 'subtasks'
      },
      splitterSettings: {
        columnIndex: 3
      },
      columns: [
        { field: 'TaskID', headerText: 'Task ID', textAlign: 'Left',
          width: '100' },
        { field: 'TaskName', headerText: 'Task Name', width: '250' },
        { field: 'StartDate', headerText: 'Start Date', width: '150' },
        { field: 'Duration', headerText: 'Duration', width: '150' },
        { field: 'Progress', headerText: 'Progress', width: '150' },
      ]
    }
  }
}

```

```

    ],
    sortSettings: { columns: [{ field: 'TaskID', direction:
'Descending' }, { field: 'TaskName', direction: 'Ascending' }] },
    };
  },
  provide: {
    gantt: [ Sort ]
  }
};
</script>

```

{% previewsample "page.domainurl/code-snippet/gantt/sorting-cs2" %}

### Sorting column dynamically

Columns in the Gantt component can be sorted dynamically using the [sortColumn](#) method. The following code example demonstrates how to invoke the [sortColumn](#) method by clicking the custom button.

### APP.VUE

```

<template>
  <div>
    <ejs-button id="sortColumn" cssClass="e-info" v-
on:click.native="sort">Sort Column</ejs-button>
    <br>
    <br>
    <ejs-gantt ref='gantt' id="GanttContainer" :dataSource="data"
:taskFields = "taskFields" :height = "height" :columns="columns"
:splitterSettings= "splitterSettings" :allowSorting= 'true'></ejs-gantt>
  </div>
</template>
<script>
import Vue from "vue";
import { GanttPlugin, Sort } from "@syncfusion/ej2-vue-gantt";
import { ButtonPlugin } from "@syncfusion/ej2-vue-buttons";
Vue.use(GanttPlugin);
Vue.use(ButtonPlugin);
export default {
  data: function() {
    return{
      data: [
        {
          TaskID: 1,
          TaskName: 'Project Initiation',
          StartDate: new Date('04/02/2019'),
          EndDate: new Date('04/21/2019'),
          subtasks: [
            { TaskID: 2, TaskName: 'Identify Site location',
StartDate: new Date('04/02/2019'), Duration: 4, Progress: 50 },
            { TaskID: 3, TaskName: 'Perform Soil test', StartDate:
new Date('04/02/2019'), Duration: 4, Progress: 50 },
            { TaskID: 4, TaskName: 'Soil test approval', StartDate:
new Date('04/02/2019'), Duration: 4, Progress: 50 },
          ]
        },
      ],
    }
  }
}

```

```

        TaskID: 5,
        TaskName: 'Project Estimation',
        StartDate: new Date('04/02/2019'),
        EndDate: new Date('04/21/2019'),
        subtasks: [
            { TaskID: 6, TaskName: 'Develop floor plan for
estimation', StartDate: new Date('04/04/2019'), Duration: 3, Progress: 50 },
            { TaskID: 7, TaskName: 'List materials', StartDate: new
Date('04/04/2019'), Duration: 3, Progress: 50 },
            { TaskID: 8, TaskName: 'Estimation approval', StartDate:
new Date('04/04/2019'), Duration: 3, Progress: 50 }
        ]
    },
],
height: '450px',
taskFields: {
    id: 'TaskID',
    name: 'TaskName',
    startDate: 'StartDate',
    duration: 'Duration',
    progress: 'Progress',
    child: 'subtasks'
},
splitterSettings: {
    columnIndex: 3
},
columns: [
    { field: 'TaskID', headerText: 'Task ID', textAlign: 'Left',
width: '100' },
    { field: 'TaskName', headerText: 'Task Name', width: '250'
},
    { field: 'StartDate', headerText: 'Start Date', width: '150'
},
    { field: 'Duration', headerText: 'Duration', width: '150' },
    { field: 'Progress', headerText: 'Progress', width: '150' },
],
];
},
provide: {
    gantt: [ Sort ]
},
methods: {
    sort: function(e){
        var ganttObj =
document.getElementById('GanttContainer').ej2_instances[0];
        ganttObj.sortModule.sortColumn('TaskID', "Descending", false);
    },
}
};
</script>

```

{% previewsample "page.domainurl/code-snippet/gantt/sorting-cs3" %}

### Clear all the sorting dynamically

In the Gantt component, you can clear all the sorted columns and return to previous position using the [clearSorting](#) public method. The following code snippet shows how to clear all the sorted columns by clicking the custom button.

#### APP.VUE

```
<template>
  <div>
    <ejs-button id="Clearsort" cssClass="e-info" v-
on:click.native="sort">Clear Sorting</ejs-button>
    <br>
    <br>
    <ejs-gantt ref='gantt' id="GanttContainer" :dataSource="data"
:taskFields = "taskFields" :height = "height" :columns="columns"
:splitterSettings= "splitterSettings" :sortSettings="sortSettings"
:allowSorting= 'true'></ejs-gantt>
  </div>
</template>
<script>
import Vue from "vue";
import { GanttPlugin, Sort } from "@syncfusion/ej2-vue-gantt";
import { ButtonPlugin } from "@syncfusion/ej2-vue-buttons";
Vue.use(GanttPlugin);
Vue.use(ButtonPlugin);
export default {
  data: function() {
    return {
      data: [
        {
          TaskID: 1,
          TaskName: 'Project Initiation',
          StartDate: new Date('04/02/2019'),
          EndDate: new Date('04/21/2019'),
          subtasks: [
            { TaskID: 2, TaskName: 'Identify Site location',
StartDate: new Date('04/02/2019'), Duration: 4, Progress: 50 },
            { TaskID: 3, TaskName: 'Perform Soil test', StartDate:
new Date('04/02/2019'), Duration: 4, Progress: 50 },
            { TaskID: 4, TaskName: 'Soil test approval', StartDate:
new Date('04/02/2019'), Duration: 4, Progress: 50 },
          ]
        },
        {
          TaskID: 5,
          TaskName: 'Project Estimation',
          StartDate: new Date('04/02/2019'),
          EndDate: new Date('04/21/2019'),
          subtasks: [
            { TaskID: 6, TaskName: 'Develop floor plan for
estimation', StartDate: new Date('04/04/2019'), Duration: 3, Progress: 50 },
            { TaskID: 7, TaskName: 'List materials', StartDate: new
Date('04/04/2019'), Duration: 3, Progress: 50 },
            { TaskID: 8, TaskName: 'Estimation approval', StartDate:
new Date('04/04/2019'), Duration: 3, Progress: 50 }
          ]
        }
      ],
    }
  }
}
```

```

    ],
    height: '450px',
    taskFields: {
      id: 'TaskID',
      name: 'TaskName',
      startDate: 'StartDate',
      duration: 'Duration',
      progress: 'Progress',
      child: 'subtasks'
    },
    columns: [
      { field: 'TaskID', headerText: 'Task ID', textAlign: 'Left',
width: '100' },
      { field: 'TaskName', headerText: 'Task Name', width: '250'
},
      { field: 'StartDate', headerText: 'Start Date', width: '150'
},
      { field: 'Duration', headerText: 'Duration', width: '150' },
      { field: 'Progress', headerText: 'Progress', width: '150' },
    ],
    splitterSettings: {
      columnIndex: 3
    },
    sortSettings: { columns: [{ field: 'TaskID', direction:
'Descending' }, { field: 'TaskName', direction: 'Ascending' }] },
  };
},
provide: {
  gantt: [ Sort ]
},
methods: {
  sort: function(e){
    var ganttObj =
document.getElementById('GanttContainer').ej2_instances[0];
    ganttObj.clearSorting();
  },
}
};
</script>

```

{% previewsample "page.domainurl/code-snippet/gantt/sorting-cs4" %}

### Sorting events

During the sort action, the Gantt component triggers two events. The [actionBegin](#) event triggers before the sort action starts, and the [actionComplete](#) event triggers after the sort action is completed.

### APP.VUE

```

<template>
  <div>
    <ejs-gantt ref='gantt' id="GanttContainer" :dataSource="data"
:taskFields = "taskFields" :height = "height" :columns="columns"
:splitterSettings= "splitterSettings" :actionBegin="actionBegin"
:actionComplete="actionComplete" :allowSorting= 'true'></ejs-gantt>
  </div>
</template>

```

```

<script>
import Vue from "vue";
import { GanttPlugin, Sort } from "@syncfusion/ej2-vue-gantt";
Vue.use(GanttPlugin);
export default {
  data: function() {
    return {
      data: [
        {
          TaskID: 1,
          TaskName: 'Project Initiation',
          StartDate: new Date('04/02/2019'),
          EndDate: new Date('04/21/2019'),
          subtasks: [
            { TaskID: 2, TaskName: 'Identify Site location',
              StartDate: new Date('04/02/2019'), Duration: 4, Progress: 50 },
            { TaskID: 3, TaskName: 'Perform Soil test', StartDate:
              new Date('04/02/2019'), Duration: 4, Progress: 50 },
            { TaskID: 4, TaskName: 'Soil test approval', StartDate:
              new Date('04/02/2019'), Duration: 4, Progress: 50 },
          ]
        },
        {
          TaskID: 5,
          TaskName: 'Project Estimation',
          StartDate: new Date('04/02/2019'),
          EndDate: new Date('04/21/2019'),
          subtasks: [
            { TaskID: 6, TaskName: 'Develop floor plan for
              estimation', StartDate: new Date('04/04/2019'), Duration: 3, Progress: 50 },
            { TaskID: 7, TaskName: 'List materials', StartDate: new
              Date('04/04/2019'), Duration: 3, Progress: 50 },
            { TaskID: 8, TaskName: 'Estimation approval', StartDate:
              new Date('04/04/2019'), Duration: 3, Progress: 50 }
          ]
        }
      ],
      height: '450px',
      taskFields: {
        id: 'TaskID',
        name: 'TaskName',
        startDate: 'StartDate',
        duration: 'Duration',
        progress: 'Progress',
        child: 'subtasks'
      },
      splitterSettings: {
        columnIndex: 3
      },
      columns: [
        { field: 'TaskID', headerText: 'Task ID', textAlign: 'Left',
          width: '100' },
        { field: 'TaskName', headerText: 'Task Name', width: '250' },
        { field: 'StartDate', headerText: 'Start Date', width: '150' },
        { field: 'Duration', headerText: 'Duration', width: '150' },
      ]
    }
  }
}

```



```
        { field: 'Progress', headerText: 'Progress', width: '150' },
      ],
      actionBegin: function(args) {
        alert(args.requestType + ' ' + args.type); //custom Action
      },
      actionComplete: function(args) {
        alert(args.requestType + ' ' + args.type); //custom Action
      },
    };
  },
  provide: {
    gantt: [ Sort ]
  },
};
</script>
```

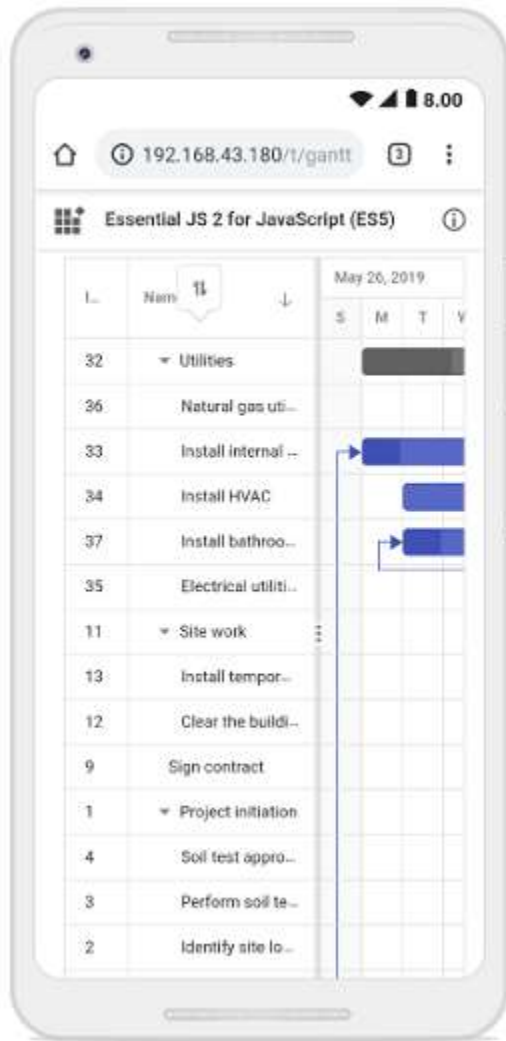
{% previewsample "page.domainurl/code-snippet/gantt/sorting-cs5" %}

The `args.requestType` is the current action name. For example, for sorting the `args.requestType`, value is `sorting`.

#### Touch interaction

To perform **tap** action on a column header, trigger [sorting](#) operation to the selected column. A popup is displayed for multi-column sorting. To sort multiple columns, tap the popup, and then tap the desired column headers.

The following screenshot shows Gantt touch sorting,



## Selection

### Selection in Vue Gantt component

Selection provides an option to highlight a row or a cell. It can be done using arrow keys or by scrolling down the mouse. To disable selection in the Gantt component, set the [allowSelection](#) to `false`.

To select data, inject the [Selection](#) module in the `provide` section.

The Gantt component supports two types of selection that can be set by using the [selectionSettings.type](#) property. They are:

- `Single`: Sets a single value by default and allows only selection of a single row or a cell.
- [Link to the Video](#): Allows you to select multiple rows or cells. To perform the multi-selection, press and hold the CTRL key and click the desired rows or cells.

To learn about Gantt Chart Selection, you can check on this video:

*Selection mode*

The Gantt component supports three types of selection modes that can be set by using the [selectionSettings.mode](#). They are:

- **Row**: Allows you to select only rows, and the row value is set by default.
- **Cell**: Allows you to select only cells.
- **Both**: Allows you to select rows and cells at the same time.

**APP.VUE**

```
<template>
  <div>
    <ejs-gantt ref='gantt' id="GanttContainer" :dataSource="data"
    :taskFields = "taskFields" :height = "height"
    :selectionSettings="selectionSettings"></ejs-gantt>
  </div>
</template>
<script>
import Vue from "vue";
import { GanttPlugin, Selection } from "@syncfusion/ej2-vue-gantt";
Vue.use(GanttPlugin);
export default {
  data: function() {
    return {
      data: [
        {
          TaskID: 1,
          TaskName: 'Project Initiation',
          StartDate: new Date('04/02/2019'),
          EndDate: new Date('04/21/2019'),
          subtasks: [
            { TaskID: 2, TaskName: 'Identify Site location',
              StartDate: new Date('04/02/2019'), Duration: 4, Progress: 50 },
            { TaskID: 3, TaskName: 'Perform Soil test', StartDate:
              new Date('04/02/2019'), Duration: 4, Progress: 50 },
            { TaskID: 4, TaskName: 'Soil test approval', StartDate:
              new Date('04/02/2019'), Duration: 4, Progress: 50 },
          ]
        },
        {
          TaskID: 5,
          TaskName: 'Project Estimation',
          StartDate: new Date('04/02/2019'),
          EndDate: new Date('04/21/2019'),
          subtasks: [
            { TaskID: 6, TaskName: 'Develop floor plan for
              estimation', StartDate: new Date('04/04/2019'), Duration: 3, Progress: 50 },
            { TaskID: 7, TaskName: 'List materials', StartDate: new
              Date('04/04/2019'), Duration: 3, Progress: 50 },
            { TaskID: 8, TaskName: 'Estimation approval', StartDate:
              new Date('04/04/2019'), Duration: 3, Progress: 50 }
          ]
        }
      ],
      height: '450px',
    }
  }
}
```

```

        taskFields: {
            id: 'TaskID',
            name: 'TaskName',
            startDate: 'StartDate',
            endDate: 'EndDate',
            duration: 'Duration',
            progress: 'Progress',
            dependency: 'Predecessor',
            child: 'subtasks'
        },
        selectionSettings: {
            mode: 'Both',
        }
    };
},
provide: {
    gantt: [ Selection ]
}
};
</script>

```

{% previewsample "page.domainurl/code-snippet/gantt/selection-cs11" %}

### [Toggle selection](#)

The toggle selection allows you to select and deselect a specific row or cell. To enable toggle selection, set the `enableToggle` property of the `selectionSettings` to `true`. If you click the selected row or cell, then it will be deselected and vice versa. By default, the `enableToggle` property is set to `false`.

### **APP.VUE**

```

<template>
    <div>
        <ejs-button id="toggle" cssClass="e-info" v-
on:click.native="toggle">Disable Toggle</ejs-button>
        <br>
        <br>
        <ejs-gantt ref='gantt' id="GanttContainer" :dataSource="data"
:taskFields = "taskFields" :height = "height"
:selectionSettings="selectionSettings"></ejs-gantt>
    </div>
</template>
<script>
import Vue from "vue";
import { GanttPlugin, Selection } from "@syncfusion/ej2-vue-gantt";
import { ButtonPlugin } from "@syncfusion/ej2-vue-buttons";
Vue.use(GanttPlugin);
Vue.use(ButtonPlugin);
export default {
    data: function() {
        return{
            data: [
                {
                    TaskID: 1,
                    TaskName: 'Project Initiation',
                    StartDate: new Date('04/02/2019'),
                    EndDate: new Date('04/21/2019'),

```

```

        subtasks: [
            { TaskID: 2, TaskName: 'Identify Site location',
StartDate: new Date('04/02/2019'), Duration: 4, Progress: 50 },
            { TaskID: 3, TaskName: 'Perform Soil test', StartDate:
new Date('04/02/2019'), Duration: 4, Progress: 50 },
            { TaskID: 4, TaskName: 'Soil test approval', StartDate:
new Date('04/02/2019'), Duration: 4, Progress: 50 },
        ]
    },
    {
        TaskID: 5,
        TaskName: 'Project Estimation',
        StartDate: new Date('04/02/2019'),
        EndDate: new Date('04/21/2019'),
        subtasks: [
            { TaskID: 6, TaskName: 'Develop floor plan for
estimation', StartDate: new Date('04/04/2019'), Duration: 3, Progress: 50 },
            { TaskID: 7, TaskName: 'List materials', StartDate: new
Date('04/04/2019'), Duration: 3, Progress: 50 },
            { TaskID: 8, TaskName: 'Estimation approval', StartDate:
new Date('04/04/2019'), Duration: 3, Progress: 50 }
        ]
    },
],
height: '450px',
taskFields: {
    id: 'TaskID',
    name: 'TaskName',
    startDate: 'StartDate',
    endDate: 'EndDate',
    duration: 'Duration',
    progress: 'Progress',
    dependency: 'Predecessor',
    child: 'subtasks'
},
selectionSettings: {
    mode: 'Row',
    type: 'Multiple',
    enableToggle: true
}
};
},
provide: {
    gantt: [ Selection ]
},
methods: {
    toggle: function(e){
        var ganttObj =
document.getElementById('GanttContainer').ej2_instances[0];
        ganttObj.selectionSettings.enableToggle = false;
    }
}
};
</script>

```

{% previewsample "page.domainurl/code-snippet/gantt/selection-cs12" %}

*Clear selection*

You can clear the selected cells and selected rows by using a method called [clearSelection](#). The following code example demonstrates how to clear the selected rows in Gantt Chart.

**APP.VUE**

```
<template>
  <div>
    <ejs-button id="selectRow" cssClass="e-info" v-
on:click.native="select">Select Multiple Rows</ejs-button>
    <ejs-button id="clearSelection" cssClass="e-info" v-
on:click.native="clear">Clear Selection</ejs-button>
    <br>
    <br>
    <ejs-gantt ref='gantt' id="GanttContainer" :dataSource="data"
:taskFields = "taskFields" :height = "height"
:selectionSettings="selectionSettings"></ejs-gantt>
  </div>
</template>
<script>
import Vue from "vue";
import { GanttPlugin, Selection } from "@syncfusion/ej2-vue-gantt";
import { ButtonPlugin } from "@syncfusion/ej2-vue-buttons";
Vue.use(GanttPlugin);
Vue.use(ButtonPlugin);
export default {
  data: function() {
    return{
      data: [
        {
          TaskID: 1,
          TaskName: 'Project Initiation',
          StartDate: new Date('04/02/2019'),
          EndDate: new Date('04/21/2019'),
          subtasks: [
            { TaskID: 2, TaskName: 'Identify Site location',
StartDate: new Date('04/02/2019'), Duration: 4, Progress: 50 },
            { TaskID: 3, TaskName: 'Perform Soil test', StartDate:
new Date('04/02/2019'), Duration: 4, Progress: 50 },
            { TaskID: 4, TaskName: 'Soil test approval', StartDate:
new Date('04/02/2019'), Duration: 4, Progress: 50 },
          ]
        },
        {
          TaskID: 5,
          TaskName: 'Project Estimation',
          StartDate: new Date('04/02/2019'),
          EndDate: new Date('04/21/2019'),
          subtasks: [
            { TaskID: 6, TaskName: 'Develop floor plan for
estimation', StartDate: new Date('04/04/2019'), Duration: 3, Progress: 50 },
            { TaskID: 7, TaskName: 'List materials', StartDate: new
Date('04/04/2019'), Duration: 3, Progress: 50 },
            { TaskID: 8, TaskName: 'Estimation approval', StartDate:
new Date('04/04/2019'), Duration: 3, Progress: 50 }
          ]
        }
      ],
    }
  },
}
```

```

    ],
    height: '450px',
    taskFields: {
        id: 'TaskID',
        name: 'TaskName',
        startDate: 'StartDate',
        endDate: 'EndDate',
        duration: 'Duration',
        progress: 'Progress',
        dependency: 'Predecessor',
        child: 'subtasks'
    },
    selectionSettings: {
        mode: 'Row',
        type: 'Multiple'
    }
};
},
provide: {
    gantt: [ Selection ]
},
methods: {
    select: function(e) {
        var ganttObj =
document.getElementById('GanttContainer').ej2_instances[0];
        ganttObj.selectionModule.selectRows([1, 3, 5]); // passing the
record index to select the rows
    },
    clear: function(e) {
        var ganttObj =
document.getElementById('GanttContainer').ej2_instances[0];
        ganttObj.clearSelection(); // Clear the selected rows
    },
}
};
</script>

```

{% previewsample "page.domainurl/code-snippet/gantt/selection-cs13" %}

#### [Get selected row indexes and records](#)

You can get the selected row indexes by using the [getSelectedRowIndex](#) method. And by using [getSelectedRecords](#) method, you can get the selected record details.

#### **APP.VUE**

```

<template>
  <div>
    <ejs-gantt ref='gantt' id="GanttContainer" :dataSource="data"
:taskFields = "taskFields" :height = "height" :allowSelection='true'
:rowSelected="rowSelected"></ejs-gantt>
  </div>
</template>
<script>
import Vue from "vue";
import { GanttPlugin, Selection } from "@syncfusion/ej2-vue-gantt";
Vue.use(GanttPlugin);

```

```

export default {
  data: function() {
    return{
      data: [
        {
          TaskID: 1,
          TaskName: 'Project Initiation',
          StartDate: new Date('04/02/2019'),
          EndDate: new Date('04/21/2019'),
          subtasks: [
            { TaskID: 2, TaskName: 'Identify Site location',
              StartDate: new Date('04/02/2019'), Duration: 4, Progress: 50 },
            { TaskID: 3, TaskName: 'Perform Soil test', StartDate:
              new Date('04/02/2019'), Duration: 4, Progress: 50 },
            { TaskID: 4, TaskName: 'Soil test approval', StartDate:
              new Date('04/02/2019'), Duration: 4, Progress: 50 },
          ]
        },
        {
          TaskID: 5,
          TaskName: 'Project Estimation',
          StartDate: new Date('04/02/2019'),
          EndDate: new Date('04/21/2019'),
          subtasks: [
            { TaskID: 6, TaskName: 'Develop floor plan for
              estimation', StartDate: new Date('04/04/2019'), Duration: 3, Progress: 50 },
            { TaskID: 7, TaskName: 'List materials', StartDate: new
              Date('04/04/2019'), Duration: 3, Progress: 50 },
            { TaskID: 8, TaskName: 'Estimation approval', StartDate:
              new Date('04/04/2019'), Duration: 3, Progress: 50 }
          ]
        },
      ],
      height: '450px',
      taskFields: {
        id: 'TaskID',
        name: 'TaskName',
        startDate: 'StartDate',
        endDate: 'EndDate',
        duration: 'Duration',
        progress: 'Progress',
        dependency: 'Predecessor',
        child: 'subtasks'
      },
      rowSelected: function(args) {
        var ganttObj =
document.getElementById('GanttContainer').ej2_instances[0];
        let selectedrowindex: number[] =
ganttObj.selectionModule.getSelectedRowIndexes(); // get the selected row
indexes.
        alert(selectedrowindex); // to alert the selected row
indexes.
        let selectedrecords: Object[] =
ganttObj.selectionModule.getSelectedRecords(); // get the selected records.
        console.log(selectedrecords); // to print the selected
records in console window.
      }
    }
  }
}

```



```

    };
  },
  provide: {
    gantt: [ Selection ]
  }
};
</script>

```

{% previewsample "page.domainurl/code-snippet/gantt/selection-cs14" %}

#### Multiple Selection based on condition

You can select multiple rows based on condition by using the [selectRows](#) method.

In the following code, the rows which contains **TaskId** value as 3 and 4 are selected at initial rendering.

#### APP.VUE

```

<template>
  <div>
    <ejs-gantt ref='gantt' id="GanttContainer" :dataSource="data"
    :taskFields = "taskFields" :height = "height"
    :selectionSettings="selectionSettings" :dataBound="dataBound"></ejs-gantt>
  </div>
</template>
<script>
import Vue from "vue";
import { GanttPlugin, Selection } from "@syncfusion/ej2-vue-gantt";
Vue.use(GanttPlugin);
export default {
  data: function() {
    return{
      data: [
        {
          TaskID: 1,
          TaskName: 'Project Initiation',
          StartDate: new Date('04/02/2019'),
          EndDate: new Date('04/21/2019'),
          subtasks: [
            { TaskID: 2, TaskName: 'Identify Site location',
              StartDate: new Date('04/02/2019'), Duration: 4, Progress: 50 },
            { TaskID: 3, TaskName: 'Perform Soil test', StartDate:
              new Date('04/02/2019'), Duration: 4, Progress: 50 },
            { TaskID: 4, TaskName: 'Soil test approval', StartDate:
              new Date('04/02/2019'), Duration: 4, Progress: 50 },
          ]
        },
        {
          TaskID: 5,
          TaskName: 'Project Estimation',
          StartDate: new Date('04/02/2019'),
          EndDate: new Date('04/21/2019'),
          subtasks: [
            { TaskID: 6, TaskName: 'Develop floor plan for
              estimation', StartDate: new Date('04/04/2019'), Duration: 3, Progress: 50 },
            { TaskID: 7, TaskName: 'List materials', StartDate: new
              Date('04/04/2019'), Duration: 3, Progress: 50 },
          ]
        }
      ]
    }
  }
}

```

```

        { TaskID: 8, TaskName: 'Estimation approval', StartDate:
new Date('04/04/2019'), Duration: 3, Progress: 50 }
    ],
    height: '450px',
    taskFields: {
        id: 'TaskID',
        name: 'TaskName',
        startDate: 'StartDate',
        endDate: 'EndDate',
        duration: 'Duration',
        progress: 'Progress',
        dependency: 'Predecessor',
        child: 'subtasks'
    },
    selectionSettings: {
        mode: 'Row',
        type: 'Multiple',
    }
    dataBound: function(args) {
        var ganttObj =
document.getElementById('GanttContainer').ej2_instances[0];
        var rowIndexes = [];
        ganttObj.treeGrid.grid.dataSource.forEach((data, index) => {
            if (data.TaskID === 3 || data.TaskID === 4) {
                rowIndexes.push(index);
            }
        });
        ganttObj.selectRows(rowIndexes);
    }
    };
    },
    provide: {
        gantt: [ Selection ]
    }
    };
</script>

```

{% previewsample "page.domainurl/code-snippet/gantt/selection-cs15" %}

#### *Touch interaction*

When you **tap** gantt row, tapped row will be selected.

**Single selection** : To select a single row or cell, perform **single tap** on it.

**Multiple selection** : To perform multiple selection, **tap** on the multiple selection popup, and then tap the desired rows or cells.



[See Also](#)

- [Touch interaction](#)

#### Row selection in Vue Gantt component

The row selection in the Gantt component can be enabled or disabled using the [allowSelection](#) property. You can get the selected row object using the [getSelectedRecords](#) method. The following code example shows how to disable the row selection in Gantt.

#### APP.VUE

```
<template>
  <div>
    <ejs-gantt ref='gantt' id="GanttContainer" :dataSource="data"
      :taskFields = "taskFields" :height = "height" :allowSelection='false'></ejs-gantt>
  </div>
</template>
<script>
```

```

import Vue from "vue";
import { GanttPlugin, Selection } from "@syncfusion/ej2-vue-gantt";
Vue.use(GanttPlugin);
export default {
  data: function() {
    return {
      data: [
        {
          TaskID: 1,
          TaskName: 'Project Initiation',
          StartDate: new Date('04/02/2019'),
          EndDate: new Date('04/21/2019'),
          subtasks: [
            { TaskID: 2, TaskName: 'Identify Site location',
              StartDate: new Date('04/02/2019'), Duration: 4, Progress: 50 },
            { TaskID: 3, TaskName: 'Perform Soil test', StartDate:
              new Date('04/02/2019'), Duration: 4, Progress: 50 },
            { TaskID: 4, TaskName: 'Soil test approval', StartDate:
              new Date('04/02/2019'), Duration: 4, Progress: 50 },
          ]
        },
        {
          TaskID: 5,
          TaskName: 'Project Estimation',
          StartDate: new Date('04/02/2019'),
          EndDate: new Date('04/21/2019'),
          subtasks: [
            { TaskID: 6, TaskName: 'Develop floor plan for
              estimation', StartDate: new Date('04/04/2019'), Duration: 3, Progress: 50 },
            { TaskID: 7, TaskName: 'List materials', StartDate: new
              Date('04/04/2019'), Duration: 3, Progress: 50 },
            { TaskID: 8, TaskName: 'Estimation approval', StartDate:
              new Date('04/04/2019'), Duration: 3, Progress: 50 }
          ]
        }
      ],
      height: '450px',
      taskFields: {
        id: 'TaskID',
        name: 'TaskName',
        startDate: 'StartDate',
        endDate: 'EndDate',
        duration: 'Duration',
        progress: 'Progress',
        dependency: 'Predecessor',
        child: 'subtasks'
      }
    };
  },
  provide: {
    gantt: [ Selection ]
  }
};
</script>

```

{% previewsample "page.domainurl/code-snippet/gantt/selection-cs5" %}

**Row** selection is the default type of Gantt selection mode.

#### *Selecting a row on initial load*

You can select a row at the time of loading by setting the index of the row to the [selectedRowIndex](#) property. Find the following code example for details.

#### **APP.VUE**

```
<template>
  <div>
    <ejs-gantt ref='gantt' id="GanttContainer" :dataSource="data"
    :selectedRowIndex = '5' :taskFields = "taskFields" :height = "height"></ejs-
    gantt>
  </div>
</template>
<script>
import Vue from "vue";
import { GanttPlugin, Selection } from "@syncfusion/ej2-vue-gantt";
Vue.use(GanttPlugin);
export default {
  data: function() {
    return {
      data: [
        {
          TaskID: 1,
          TaskName: 'Project Initiation',
          StartDate: new Date('04/02/2019'),
          EndDate: new Date('04/21/2019'),
          subtasks: [
            { TaskID: 2, TaskName: 'Identify Site location',
              StartDate: new Date('04/02/2019'), Duration: 4, Progress: 50 },
            { TaskID: 3, TaskName: 'Perform Soil test', StartDate:
              new Date('04/02/2019'), Duration: 4, Progress: 50 },
            { TaskID: 4, TaskName: 'Soil test approval', StartDate:
              new Date('04/02/2019'), Duration: 4, Progress: 50 },
          ]
        },
        {
          TaskID: 5,
          TaskName: 'Project Estimation',
          StartDate: new Date('04/02/2019'),
          EndDate: new Date('04/21/2019'),
          subtasks: [
            { TaskID: 6, TaskName: 'Develop floor plan for
              estimation', StartDate: new Date('04/04/2019'), Duration: 3, Progress: 50 },
            { TaskID: 7, TaskName: 'List materials', StartDate: new
              Date('04/04/2019'), Duration: 3, Progress: 50 },
            { TaskID: 8, TaskName: 'Estimation approval', StartDate:
              new Date('04/04/2019'), Duration: 3, Progress: 50 }
          ]
        }
      ],
      height: '450px',
      taskFields: {
        id: 'TaskID',
        name: 'TaskName',
        startDate: 'StartDate',
```

```

        endDate: 'EndDate',
        duration: 'Duration',
        progress: 'Progress',
        dependency: 'Predecessor',
        child: 'subtasks'
    },
    };
    },
    provide: {
        gantt: [ Selection ]
    }
};
</script>

```

{% previewsample "page.domainurl/code-snippet/gantt/selection-cs6" %}

### Selecting a row dynamically

You can select a single row dynamically using the [selectRow](#) method. Similarly, you can use the [selectRows](#) method to dynamically select multiple rows. The following code demonstrates how to select a single or multiple rows dynamically by clicking the custom button.

### APP.VUE

```

<template>
  <div>
    <ejs-button id="selectRow" cssClass="e-info" v-
on:click.native="select">Select Row</ejs-button>
    <ejs-button id="selectRows" cssClass="e-info" v-
on:click.native="Multiselect">Select Multiple Rows</ejs-button>
    <br>
    <br>
    <ejs-gantt ref='gantt' id="GanttContainer" :dataSource="data"
:taskFields = "taskFields" :height = "height" ></ejs-gantt>
  </div>
</template>
<script>
import Vue from "vue";
import { GanttPlugin, Selection } from "@syncfusion/ej2-vue-gantt";
import { ButtonPlugin } from "@syncfusion/ej2-vue-buttons";
Vue.use(GanttPlugin);
Vue.use(ButtonPlugin);
export default {
  data: function() {
    return{
      data: [
        {
          TaskID: 1,
          TaskName: 'Project Initiation',
          StartDate: new Date('04/02/2019'),
          EndDate: new Date('04/21/2019'),
          subtasks: [
            { TaskID: 2, TaskName: 'Identify Site location',
StartDate: new Date('04/02/2019'), Duration: 4, Progress: 50 },
            { TaskID: 3, TaskName: 'Perform Soil test', StartDate:
new Date('04/02/2019'), Duration: 4, Progress: 50 },

```

```

        { TaskID: 4, TaskName: 'Soil test approval', StartDate:
new Date('04/02/2019'), Duration: 4, Progress: 50 },
    ],
    },
    {
        TaskID: 5,
        TaskName: 'Project Estimation',
        StartDate: new Date('04/02/2019'),
        EndDate: new Date('04/21/2019'),
        subtasks: [
            { TaskID: 6, TaskName: 'Develop floor plan for
estimation', StartDate: new Date('04/04/2019'), Duration: 3, Progress: 50 },
            { TaskID: 7, TaskName: 'List materials', StartDate: new
Date('04/04/2019'), Duration: 3, Progress: 50 },
            { TaskID: 8, TaskName: 'Estimation approval', StartDate:
new Date('04/04/2019'), Duration: 3, Progress: 50 }
        ]
    },
],
height: '450px',
taskFields: {
    id: 'TaskID',
    name: 'TaskName',
    startDate: 'StartDate',
    endDate: 'EndDate',
    duration: 'Duration',
    progress: 'Progress',
    dependency: 'Predecessor',
    child: 'subtasks'
},
selectionSettings: {
    mode: 'Row',
    type: 'Multiple'
}
};
},
provide: {
    gantt: [ Selection ]
},
methods: {
    select: function(e){
        var ganttObj =
document.getElementById('GanttContainer').ej2_instances[0];
        ganttObj.selectionModule.selectRow(2); // passing the record index
to select the row
    },
    Multiselect: function(e){
        var ganttObj =
document.getElementById('GanttContainer').ej2_instances[0];
        ganttObj.selectionModule.selectRows([1, 2, 3]); // passing the
record index to select the rows
    },
}
};
</script>

```

```
{% previewsample "page.domainurl/code-snippet/gantt/selection-cs7" %}
```

### Multiple row selection

You can select multiple rows by setting the [selectionSettings.type](#) property to multiple. You can select more than one row by holding down the CTRL key while selecting multiple rows. The following code example explains how to enable multiple selection in Gantt.

### APP.VUE

```
<template>
  <div>
    <ejs-gantt ref='gantt' id="GanttContainer" :dataSource="data"
:taskFields = "taskFields" :height = "height"
:selectionSettings="selectionSettings"></ejs-gantt>
  </div>
</template>
<script>
import Vue from "vue";
import { GanttPlugin, Selection } from "@syncfusion/ej2-vue-gantt";
Vue.use(GanttPlugin);
export default {
  data: function() {
    return {
      data: [
        {
          TaskID: 1,
          TaskName: 'Project Initiation',
          StartDate: new Date('04/02/2019'),
          EndDate: new Date('04/21/2019'),
          subtasks: [
            { TaskID: 2, TaskName: 'Identify Site location',
StartDate: new Date('04/02/2019'), Duration: 4, Progress: 50 },
            { TaskID: 3, TaskName: 'Perform Soil test', StartDate:
new Date('04/02/2019'), Duration: 4, Progress: 50 },
            { TaskID: 4, TaskName: 'Soil test approval', StartDate:
new Date('04/02/2019'), Duration: 4, Progress: 50 },
          ]
        },
        {
          TaskID: 5,
          TaskName: 'Project Estimation',
          StartDate: new Date('04/02/2019'),
          EndDate: new Date('04/21/2019'),
          subtasks: [
            { TaskID: 6, TaskName: 'Develop floor plan for
estimation', StartDate: new Date('04/04/2019'), Duration: 3, Progress: 50 },
            { TaskID: 7, TaskName: 'List materials', StartDate: new
Date('04/04/2019'), Duration: 3, Progress: 50 },
            { TaskID: 8, TaskName: 'Estimation approval', StartDate:
new Date('04/04/2019'), Duration: 3, Progress: 50 }
          ]
        }
      ],
      height: '450px',
      taskFields: {
        id: 'TaskID',
        name: 'TaskName',

```



```

        startDate: 'StartDate',
        endDate: 'EndDate',
        duration: 'Duration',
        progress: 'Progress',
        dependency: 'Predecessor',
        child: 'subtasks'
    },
    selectionSettings: {
        mode: 'Row',
        type: 'Multiple',
    }
};
},
provide: {
    gantt: [ Selection ]
}
};
</script>

```

{% previewsample "page.domainurl/code-snippet/gantt/selection-cs8" %}

#### *Customize row selection action*

While selecting a row in Gantt, the [rowSelecting](#) and [rowSelected](#) events will be triggered. The [rowSelecting](#) event will be triggered on initialization of row selection, and you can get the previously selected row and current selecting row's information, which is used to prevent selection of a particular row. The [rowSelected](#) event will be triggered on completion of row selection action, and you can get the current selected row's information through this event. The following code example demonstrates how to prevent the selection of a row using the [rowSelecting](#) event.

#### **APP.VUE**

```

<template>
  <div>
    <ejs-gantt ref='gantt' id="GanttContainer" :dataSource="data"
    :taskFields = "taskFields" :height = "height" :rowSelecting="rowSelecting"
    :selectionSettings="selectionSettings"></ejs-gantt>
  </div>
</template>
<script>
import Vue from "vue";
import { GanttPlugin, Selection } from "@syncfusion/ej2-vue-gantt";
Vue.use(GanttPlugin);
export default {
  data: function() {
    return {
      data: [
        {
          TaskID: 1,
          TaskName: 'Project Initiation',
          StartDate: new Date('04/02/2019'),
          EndDate: new Date('04/21/2019'),
          subtasks: [
            { TaskID: 2, TaskName: 'Identify Site location',
              StartDate: new Date('04/02/2019'), Duration: 4, Progress: 50 },
            { TaskID: 3, TaskName: 'Perform Soil test', StartDate:
              new Date('04/02/2019'), Duration: 4, Progress: 50 },

```

```

        { TaskID: 4, TaskName: 'Soil test approval', StartDate:
new Date('04/02/2019'), Duration: 4, Progress: 50 },
    ],
    },
    {
        TaskID: 5,
        TaskName: 'Project Estimation',
        StartDate: new Date('04/02/2019'),
        EndDate: new Date('04/21/2019'),
        subtasks: [
            { TaskID: 6, TaskName: 'Develop floor plan for
estimation', StartDate: new Date('04/04/2019'), Duration: 3, Progress: 50 },
            { TaskID: 7, TaskName: 'List materials', StartDate: new
Date('04/04/2019'), Duration: 3, Progress: 50 },
            { TaskID: 8, TaskName: 'Estimation approval', StartDate:
new Date('04/04/2019'), Duration: 3, Progress: 50 }
        ]
    },
],
height: '450px',
taskFields: {
    id: 'TaskID',
    name: 'TaskName',
    startDate: 'StartDate',
    endDate: 'EndDate',
    duration: 'Duration',
    progress: 'Progress',
    dependency: 'Predecessor',
    child: 'subtasks'
},
rowSelecting: function (args) {
    if (args.data.TaskID == 4) {
        args.cancel = true;
    }
},
selectionSettings: {
    mode: 'Row'
}
};
},
provide: {
    gantt: [ Selection ]
}
};
</script>

```

{% previewsample "page.domainurl/code-snippet/gantt/selection-cs10" %}

In the Gantt component, when you click an already selected row, selection will be cleared. While deselecting a row in Gantt, the [rowDeselecting](#) and [rowDeselected](#) events will occur. The [rowDeselecting](#) event will occur on initialization of deselecting row, and you can get the current deselecting row's information to prevent the deselection of particular row. The [rowDeselected](#) event will occur on completion of row deselection action, and you can get the current deselected row's information.

### Cell selection in Vue Gantt component

You can select a cell in the Gantt component by setting the [selectionSettings.mode](#) property to cell. You can get the selected cell information using the [getSelectedRowCellIndexes](#) method. This method returns the result as an object collection, which has `cellIndexes` and `rowIndex` information of the selected cells.

Refer to the following code example to enable the cell selection in Gantt.

#### APP.VUE

```
<template>
  <div>
    <ejs-gantt ref='gantt' id="GanttContainer" :dataSource="data"
      :taskFields = "taskFields" :height = "height"
      :selectionSettings="selectionSettings"></ejs-gantt>
  </div>
</template>
<script>
import Vue from "vue";
import { GanttPlugin, Selection } from "@syncfusion/ej2-vue-gantt";
Vue.use(GanttPlugin);
export default {
  data: function() {
    return{
      data: [
        {
          TaskID: 1,
          TaskName: 'Project Initiation',
          StartDate: new Date('04/02/2019'),
          EndDate: new Date('04/21/2019'),
          subtasks: [
            { TaskID: 2, TaskName: 'Identify Site location',
              StartDate: new Date('04/02/2019'), Duration: 4, Progress: 50 },
            { TaskID: 3, TaskName: 'Perform Soil test', StartDate:
              new Date('04/02/2019'), Duration: 4, Progress: 50 },
            { TaskID: 4, TaskName: 'Soil test approval', StartDate:
              new Date('04/02/2019'), Duration: 4, Progress: 50 },
          ]
        },
        {
          TaskID: 5,
          TaskName: 'Project Estimation',
          StartDate: new Date('04/02/2019'),
          EndDate: new Date('04/21/2019'),
          subtasks: [
            { TaskID: 6, TaskName: 'Develop floor plan for
              estimation', StartDate: new Date('04/04/2019'), Duration: 3, Progress: 50 },
            { TaskID: 7, TaskName: 'List materials', StartDate: new
              Date('04/04/2019'), Duration: 3, Progress: 50 },
            { TaskID: 8, TaskName: 'Estimation approval', StartDate:
              new Date('04/04/2019'), Duration: 3, Progress: 50 }
          ]
        }
      ],
      height: '450px',
      taskFields: {
        id: 'TaskID',
```

```

        name: 'TaskName',
        startDate: 'StartDate',
        endDate: 'EndDate',
        duration: 'Duration',
        progress: 'Progress',
        dependency: 'Predecessor',
        child: 'subtasks'
    },
    selectionSettings: {
        mode: 'Cell'
    }
};
},
provide: {
    gantt: [ Selection ]
}
};
</script>

```

{% previewsample "page.domainurl/code-snippet/gantt/selection-cs1" %}

### Selecting multiple cells

You can select multiple cells by setting the [selectionSettings.type](#) property to multiple and the [selectionSettings.mode](#) property to cell. Multiple cells can be selected by holding the CTRL key and selecting the cells. The following code example demonstrates how to select multiple cells.

### APP.VUE

```

<template>
  <div>
    <ejs-gantt ref='gantt' id="GanttContainer" :dataSource="data"
    :taskFields = "taskFields" :height = "height"
    :selectionSettings="selectionSettings"></ejs-gantt>
  </div>
</template>
<script>
import Vue from "vue";
import { GanttPlugin, Selection } from "@syncfusion/ej2-vue-gantt";
Vue.use(GanttPlugin);
export default {
  data: function() {
    return {
      data: [
        {
          TaskID: 1,
          TaskName: 'Project Initiation',
          StartDate: new Date('04/02/2019'),
          EndDate: new Date('04/21/2019'),
          subtasks: [
            { TaskID: 2, TaskName: 'Identify Site location',
              StartDate: new Date('04/02/2019'), Duration: 4, Progress: 50 },
            { TaskID: 3, TaskName: 'Perform Soil test', StartDate:
              new Date('04/02/2019'), Duration: 4, Progress: 50 },
            { TaskID: 4, TaskName: 'Soil test approval', StartDate:
              new Date('04/02/2019'), Duration: 4, Progress: 50 },
          ]
        }
      ]
    }
  }
}

```

```

    },
    {
      TaskID: 5,
      TaskName: 'Project Estimation',
      StartDate: new Date('04/02/2019'),
      EndDate: new Date('04/21/2019'),
      subtasks: [
        { TaskID: 6, TaskName: 'Develop floor plan for estimation', StartDate: new Date('04/04/2019'), Duration: 3, Progress: 50 },
        { TaskID: 7, TaskName: 'List materials', StartDate: new Date('04/04/2019'), Duration: 3, Progress: 50 },
        { TaskID: 8, TaskName: 'Estimation approval', StartDate: new Date('04/04/2019'), Duration: 3, Progress: 50 }
      ]
    },
  ],
  height: '450px',
  taskFields: {
    id: 'TaskID',
    name: 'TaskName',
    startDate: 'StartDate',
    endDate: 'EndDate',
    duration: 'Duration',
    progress: 'Progress',
    dependency: 'Predecessor',
    child: 'subtasks'
  },
  selectionSettings: {
    mode: 'Cell',
    type: 'Multiple'
  }
};
},
provide: {
  gantt: [ Selection ]
}
};
</script>

```

{% previewsample "page.domainurl/code-snippet/gantt/selection-cs2" %}

### Selecting a cell dynamically

You can select a cell dynamically using the [selectCell](#) method. Refer to the following code example for details.

### APP.VUE

```

<template>
  <div>
    <ejs-button id="selectCell" cssClass="e-info" v-on:click.native="select">Select Cell</ejs-button>
    <br>
    <br>
    <ejs-gantt ref='gantt' id="GanttContainer" :dataSource="data" :taskFields = "taskFields" :height = "height" :selectionSettings="selectionSettings"></ejs-gantt>
  </div>

```

```

    </div>
</template>
<script>
import Vue from "vue";
import { GanttPlugin, Selection } from "@syncfusion/ej2-vue-gantt";
import { ButtonPlugin } from "@syncfusion/ej2-vue-buttons";
Vue.use(GanttPlugin);
Vue.use(ButtonPlugin);
export default {
  data: function() {
    return {
      data: [
        {
          TaskID: 1,
          TaskName: 'Project Initiation',
          StartDate: new Date('04/02/2019'),
          EndDate: new Date('04/21/2019'),
          subtasks: [
            { TaskID: 2, TaskName: 'Identify Site location',
StartDate: new Date('04/02/2019'), Duration: 4, Progress: 50 },
            { TaskID: 3, TaskName: 'Perform Soil test', StartDate:
new Date('04/02/2019'), Duration: 4, Progress: 50 },
            { TaskID: 4, TaskName: 'Soil test approval', StartDate:
new Date('04/02/2019'), Duration: 4, Progress: 50 },
          ]
        },
        {
          TaskID: 5,
          TaskName: 'Project Estimation',
          StartDate: new Date('04/02/2019'),
          EndDate: new Date('04/21/2019'),
          subtasks: [
            { TaskID: 6, TaskName: 'Develop floor plan for
estimation', StartDate: new Date('04/04/2019'), Duration: 3, Progress: 50 },
            { TaskID: 7, TaskName: 'List materials', StartDate: new
Date('04/04/2019'), Duration: 3, Progress: 50 },
            { TaskID: 8, TaskName: 'Estimation approval', StartDate:
new Date('04/04/2019'), Duration: 3, Progress: 50 }
          ]
        },
      ],
      height: '450px',
      taskFields: {
        id: 'TaskID',
        name: 'TaskName',
        startDate: 'StartDate',
        endDate: 'EndDate',
        duration: 'Duration',
        progress: 'Progress',
        dependency: 'Predecessor',
        child: 'subtasks'
      },
      selectionSettings: {
        mode: 'Cell'
      }
    };
  },
};

```

```

provide: {
  gantt: [ Selection ]
},
methods: {
  select: function(e) {
    var ganttObj =
document.getElementById('GanttContainer').ej2_instances[0];
    ganttObj.selectionModule.selectCell({ cellIndex: 1, rowIndex: 1
  });
  },
}
};
</script>

```

{% previewsample "page.domainurl/code-snippet/gantt/selection-cs3" %}

#### Customize cell selection action

While selecting a cell in Gantt, the [cellSelecting](#) and [cellSelected](#) event will be triggered. The [cellSelecting](#) event will be triggered on initialization of cell selection action, and you can get the current selecting cell information to prevent the selection of a particular cell in a particular row. The [cellSelected](#) event will be triggered on completion of cell selection action, and you can get the current selected cell's information. The following code example demonstrates how to prevent the selection of the cell using the [cellSelecting](#) event.

#### APP.VUE

```

<template>
  <div>
    <ejs-gantt ref='gantt' id="GanttContainer" :dataSource="data"
:taskFields = "taskFields" :height = "height" :cellSelecting="cellSelecting"
:selectionSettings="selectionSettings"></ejs-gantt>
  </div>
</template>
<script>
import Vue from "vue";
import { GanttPlugin, Selection } from "@syncfusion/ej2-vue-gantt";
Vue.use(GanttPlugin);
export default {
  data: function() {
    return{
      data: [
        {
          TaskID: 1,
          TaskName: 'Project Initiation',
          StartDate: new Date('04/02/2019'),
          EndDate: new Date('04/21/2019'),
          subtasks: [
            { TaskID: 2, TaskName: 'Identify Site location',
StartDate: new Date('04/02/2019'), Duration: 4, Progress: 50 },
            { TaskID: 3, TaskName: 'Perform Soil test', StartDate:
new Date('04/02/2019'), Duration: 4, Progress: 50 },
            { TaskID: 4, TaskName: 'Soil test approval', StartDate:
new Date('04/02/2019'), Duration: 4, Progress: 50 },
          ]
        },
      ],
    }
  },
}

```

```

        TaskID: 5,
        TaskName: 'Project Estimation',
        StartDate: new Date('04/02/2019'),
        EndDate: new Date('04/21/2019'),
        subtasks: [
            { TaskID: 6, TaskName: 'Develop floor plan for
estimation', StartDate: new Date('04/04/2019'), Duration: 3, Progress: 50 },
            { TaskID: 7, TaskName: 'List materials', StartDate: new
Date('04/04/2019'), Duration: 3, Progress: 50 },
            { TaskID: 8, TaskName: 'Estimation approval', StartDate:
new Date('04/04/2019'), Duration: 3, Progress: 50 }
        ]
    },
],
height: '450px',
taskFields: {
    id: 'TaskID',
    name: 'TaskName',
    startDate: 'StartDate',
    endDate: 'EndDate',
    duration: 'Duration',
    progress: 'Progress',
    dependency: 'Predecessor',
    child: 'subtasks'
},
cellSelecting: function (args) {
    if (args.data.TaskID == 4 && args.cellIndex.cellIndex == 1)
    {
        args.cancel = true;
    }
},
selectionSettings: {
    mode: 'Cell'
}
};
},
provide: {
    gantt: [ Selection ]
}
};
</script>

```

{% previewsample "page.domainurl/code-snippet/gantt/selection-cs4" %}

## Rows

### Rows in Vue Gantt component

Row represents a task information from the data source, and it is possible to perform the following actions in Gantt rows.

#### Row height

It is possible to change the height of the row in Gantt by setting row height in pixels to the [rowHeight](#) property. The following code example explains how to change the row height in the Gantt at load time.

#### APP.VUE

```
<template>
```



```

<div>
  <ejs-gantt ref='gantt' id="GanttContainer" :dataSource="data"
:taskFields = "taskFields" :height = "height" :rowHeight="rowHeight"></ejs-
gantt>
</div>
</template>
<script>
import Vue from "vue";
import { GanttPlugin } from "@syncfusion/ej2-vue-gantt";
import { projectNewData } from './data-source.js';
Vue.use(GanttPlugin);
export default {
  data: function() {
    return{
      data: [
        {
          TaskID: 1,
          TaskName: 'Project Initiation',
          StartDate: new Date('04/02/2019'),
          EndDate: new Date('04/21/2019'),
          isParent:true,
          subtasks: [
            { TaskID: 2, TaskName: 'Identify Site location', StartDate:
new Date('04/02/2019'), Duration: 3, Progress: 50,isParent:false },
            { TaskID: 3, TaskName: 'Perform Soil test', StartDate: new
Date('04/02/2019'), Duration: 3, Progress: 70, resources: [2, 3,
5],isParent:false },
            { TaskID: 4, TaskName: 'Soil test approval', StartDate: new
Date('04/02/2019'), Duration: 3, Predecessor:"2FS", Progress:
80,isParent:false },
          ]
        },
        {
          TaskID: 5,
          TaskName: 'Project Estimation',
          StartDate: new Date('04/02/2019'),
          EndDate: new Date('04/21/2019'),
          isParent:true,
          subtasks: [
            { TaskID: 6, TaskName: 'Develop floor plan for estimation',
StartDate: new Date('04/04/2019'), Duration: 4, Progress: 50, resources:
[4],isParent:false },
            { TaskID: 7, TaskName: 'List materials', StartDate: new
Date('04/04/2019'), Duration: 4, Progress: 50, DurationUnit:'day',
resources: [4, 8],isParent:false },
            { TaskID: 8, TaskName: 'Estimation approval', StartDate: new
Date('04/04/2019'), Duration: 4,Predecessor:"6SS", DurationUnit:'minute',
Progress: 70, resources: [12, 5],isParent:false }
          ]
        },
      ],
      height: '450px',
      taskFields: {
        id: 'TaskID',
        name: 'TaskName',
        startDate: 'StartDate',
        duration: 'Duration',

```

```

        progress: 'Progress',
        child: 'subtasks'
    },
    rowHeight: 60
};
}
};
</script>

```

{% previewsample "page.domainurl/code-snippet/gantt/rows-cs1" %}

#### [Expand/Collapse Row](#)

In Gantt parent tasks are expanded/collapsed by using expand/collapse icons, expand all/collapse all toolbar items and by using public methods. By default all tasks in Gantt was rendered in expanded state but you can change this status in Gantt.

#### [Collapse all tasks at Gantt load](#)

All tasks available in Gantt was rendered in collapsed state by setting [collapseAllParentTasks](#) property as **true**. The following code example shows how to use [collapseAllParentTasks](#) property.

#### **APP.VUE**

```

<template>
  <div>
    <ejs-gantt ref='gantt' id="GanttContainer" :dataSource="data"
    :taskFields = "taskFields" :height = "height"
    :collapseAllParentTasks="collapseAllParentTasks"></ejs-gantt>
  </div>
</template>
<script>
import Vue from "vue";
import { GanttPlugin } from "@syncfusion/ej2-vue-gantt";
import { projectNewData } from './data-source.js';
Vue.use(GanttPlugin);
export default {
  data: function() {
    return{
      data: [
        {
          TaskID: 1,
          TaskName: 'Project Initiation',
          StartDate: new Date('04/02/2019'),
          EndDate: new Date('04/21/2019'),
          isParent:true,
          subtasks: [
            { TaskID: 2, TaskName: 'Identify Site location', StartDate:
new Date('04/02/2019'), Duration: 3, Progress: 50,isParent:false },
            { TaskID: 3, TaskName: 'Perform Soil test', StartDate: new
Date('04/02/2019'), Duration: 3, Progress: 70, resources: [2, 3,
5],isParent:false },
            { TaskID: 4, TaskName: 'Soil test approval', StartDate: new
Date('04/02/2019'), Duration: 3, Predecessor:"2FS", Progress:
80,isParent:false },
          ]
        },
      ],
    }
  }
}

```

```

        TaskID: 5,
        TaskName: 'Project Estimation',
        StartDate: new Date('04/02/2019'),
        EndDate: new Date('04/21/2019'),
        isParent:true,
        subtasks: [
            { TaskID: 6, TaskName: 'Develop floor plan for estimation',
              StartDate: new Date('04/04/2019'), Duration: 4, Progress: 50, resources:
                [4],isParent:false },
            { TaskID: 7, TaskName: 'List materials', StartDate: new
              Date('04/04/2019'), Duration: 4, Progress: 50, DurationUnit:'day',
              resources: [4, 8],isParent:false },
            { TaskID: 8, TaskName: 'Estimation approval', StartDate: new
              Date('04/04/2019'), Duration: 4,Predecessor:"6SS", DurationUnit:'minute',
              Progress: 70, resources: [12, 5],isParent:false }
        ]
    },
    ],
    height: '450px',
    taskFields: {
        id: 'TaskID',
        name: 'TaskName',
        startDate: 'StartDate',
        duration: 'Duration',
        progress: 'Progress',
        child: 'subtasks'
    },
    collapseAllParentTasks: 'true',
};
},
};
};
</script>

```

{% previewsample "page.domainurl/code-snippet/gantt/rows-cs2" %}

#### Define expand/collapse status of tasks

In Gantt, you can render some tasks in collapsed state and some tasks in expanded state, this can done by defining expand status of the task in data source. This value was mapped to Gantt component by using [expandState](#) property. The following code example shows how to use this property.

#### APP.VUE

```

<template>
  <div>
    <ejs-gantt ref='gantt' id="GanttContainer" :dataSource="data"
      :taskFields = "taskFields" :height = "height"></ejs-gantt>
  </div>
</template>
<script>
import Vue from "vue";
import { GanttPlugin } from "@syncfusion/ej2-vue-gantt";
import { projectNewData } from '../data-source.ts';
Vue.use(GanttPlugin);
export default {
  data: function() {
    return{

```

```

        data: [
            {
                TaskID: 1,
                TaskName: 'Project Initiation',
                StartDate: new Date('04/02/2019'),
                EndDate: new Date('04/21/2019'),
                isExpand:true,
                subtasks: [
                    { TaskID: 2, TaskName: 'Identify Site location', StartDate:
new Date('04/02/2019'), Duration: 3, Progress: 50,isParent:false },
                    { TaskID: 3, TaskName: 'Perform Soil test', StartDate: new
Date('04/02/2019'), Duration: 3, Progress: 70, resources: [2, 3,
5],isParent:false },
                    { TaskID: 4, TaskName: 'Soil test approval', StartDate: new
Date('04/02/2019'), Duration: 3, Predecessor:"2FS", Progress:
80,isParent:false },
                ]
            },
            {
                TaskID: 5,
                TaskName: 'Project Estimation',
                StartDate: new Date('04/02/2019'),
                EndDate: new Date('04/21/2019'),
                isExpand:false,
                subtasks: [
                    { TaskID: 6, TaskName: 'Develop floor plan for estimation',
StartDate: new Date('04/04/2019'), Duration: 4, Progress: 50, resources:
[4],isParent:false },
                    { TaskID: 7, TaskName: 'List materials', StartDate: new
Date('04/04/2019'), Duration: 4, Progress: 50, DurationUnit:'day',
resources: [4, 8],isParent:false },
                    { TaskID: 8, TaskName: 'Estimation approval', StartDate: new
Date('04/04/2019'), Duration: 4,Predecessor:"6SS", DurationUnit:'minute',
Progress: 70, resources: [12, 5],isParent:false }
                ]
            },
        ],
        height: '450px',
        taskFields: {
            id: 'TaskID',
            name: 'TaskName',
            startDate: 'StartDate',
            duration: 'Duration',
            progress: 'Progress',
            expandState:'isExpand',
            child: 'subtasks'
        },
    };
},
};
</script>

```

{% previewsample "page.domainurl/code-snippet/gantt/rows-cs3" %}

### Customize expand/collapse action

On expand action [expanding](#) and [expanded](#) event will be triggered with current expanding row's information. Similarly on collapse action [collapsing](#) and [collapsed](#) event will be triggered. Using this events and it's arguments you can customize the expand/collapse action. The following code example shows how to prevent the particular row from expand/collapse action using [expanding](#) and [collapsing](#) event.

### APP.VUE

```
<template>
  <div>
    <ejs-gantt ref='gantt' id="GanttContainer" :dataSource="data"
    :taskFields = "taskFields" :height = "height" :collapsing="collapsing"
    :expanding="expanding"></ejs-gantt>
  </div>
</template>
<script>
import Vue from "vue";
import { GanttPlugin } from "@syncfusion/ej2-vue-gantt";
import { projectNewData } from '../data-source.ts';
Vue.use(GanttPlugin);
export default {
  data: function() {
    return {
      data: [
        {
          TaskID: 1,
          TaskName: 'Project Initiation',
          StartDate: new Date('04/02/2019'),
          EndDate: new Date('04/21/2019'),
          isParent:true,
          subtasks: [
            { TaskID: 2, TaskName: 'Identify Site location', StartDate:
new Date('04/02/2019'), Duration: 3, Progress: 50,isParent:false },
            { TaskID: 3, TaskName: 'Perform Soil test', StartDate: new
Date('04/02/2019'), Duration: 3, Progress: 70, resources: [2, 3,
5],isParent:false },
            { TaskID: 4, TaskName: 'Soil test approval', StartDate: new
Date('04/02/2019'), Duration: 3, Predecessor:"2FS", Progress:
80,isParent:false },
          ]
        },
        {
          TaskID: 5,
          TaskName: 'Project Estimation',
          StartDate: new Date('04/02/2019'),
          EndDate: new Date('04/21/2019'),
          isParent:true,
          subtasks: [
            { TaskID: 6, TaskName: 'Develop floor plan for estimation',
StartDate: new Date('04/04/2019'), Duration: 4, Progress: 50, resources:
[4],isParent:false },
            { TaskID: 7, TaskName: 'List materials', StartDate: new
Date('04/04/2019'), Duration: 4, Progress: 50, DurationUnit:'day',
resources: [4, 8],isParent:false },
          ]
        }
      ]
    }
  }
}
```

```

        { TaskID: 8, TaskName: 'Estimation approval', StartDate: new
Date('04/04/2019'), Duration: 4, Predecessor: "6SS", DurationUnit: 'minute',
Progress: 70, resources: [12, 5], isParent: false }
    ]
},
],
    height: '450px',
    taskFields: {
        id: 'TaskID',
        name: 'TaskName',
        startDate: 'StartDate',
        duration: 'Duration',
        progress: 'Progress',
        child: 'subtasks'
    },
    collapsing: function(args){
        if(args.data.TaskID==1)
            args.cancel=true;
    },
    expanding: function(args){
        if(args.data.TaskID==5)
            args.cancel=true;
    }
};
},
};
</script>

```

{% previewsample "page.domainurl/code-snippet/gantt/rows-cs4" %}

#### Customize rows

You can customize the appearance of a row in grid side, by using the [rowDataBound](#) event and in chart side by using [queryTaskbarInfo](#) event

#### APP.VUE

```

<template>
  <div>
    <ejs-gantt ref='gantt' id="GanttContainer" :dataSource="data"
:taskFields = "taskFields" :height = "height" :queryTaskbarInfo =
"queryTaskbarInfo" :rowBound = "rowBound"></ejs-gantt>
  </div>
</template>
<script>
import Vue from "vue";
import { GanttPlugin } from "@syncfusion/ej2-vue-gantt";
Vue.use(GanttPlugin);
export default {
  data: function() {
    return {
      data: [
        {
          TaskID: 1,
          TaskName: 'Project Initiation',
          StartDate: new Date('04/02/2019'),
          EndDate: new Date('04/21/2019'),

```

```

        isParent:true,
        subtasks: [
            { TaskID: 2, TaskName: 'Identify Site location', StartDate:
new Date('04/02/2019'), Duration: 0, Progress: 50,isParent:false },
            { TaskID: 3, TaskName: 'Perform Soil test', StartDate: new
Date('04/02/2019'), Duration: 4, Progress: 70, resources: [2, 3,
5],isParent:false },
            { TaskID: 4, TaskName: 'Soil test approval', StartDate: new
Date('04/02/2019'), Duration: 4,Predecessor:"2FS", Progress:
50,isParent:false },
        ]
    },
    {
        TaskID: 5,
        TaskName: 'Project Estimation',
        StartDate: new Date('04/02/2019'),
        EndDate: new Date('04/21/2019'),
        isParent:true,
        subtasks: [
            { TaskID: 6, TaskName: 'Develop floor plan for estimation',
StartDate: new Date('04/04/2019'), Duration: 3, Progress: 50, resources:
[4],isParent:false },
            { TaskID: 7, TaskName: 'List materials', StartDate: new
Date('04/04/2019'), Duration: 3, Progress: 80, resources: [4,
8],isParent:false },
            { TaskID: 8, TaskName: 'Estimation approval', StartDate: new
Date('04/04/2019'), Duration: 0,Predecessor:"6SS", Progress: 70, resources:
[12, 5],isParent:false }
        ]
    },
],
    height: '450px',
    taskFields: {
        id: 'TaskID',
        name: 'TaskName',
        startDate: 'StartDate',
        duration: 'Duration',
        progress: 'Progress',
        child: 'subtasks'
    },
    rowBound: function(args) {
        if (args.data['TaskID'] == 4) {
            args.rowElement.style.background = 'cyan';
        }
    },
    queryTaskbarInfo: function(args) {
        if (args.data['TaskID'] == 4) {
            args.rowElement.style.background = 'cyan';
        }
    }
};
},
};
</script>

```

{% previewsample "page.domainurl/code-snippet/gantt/rows-cs11" %}

*Styling alternate rows*

You can change the background colour of alternative rows in Gantt chart, by overriding the class as shown below.

```
.e-altrow, tr.e-chart-row:nth-child(even) {
background-color: #f2f2f2;
}
```

**APP.VUE**

```
<template>
  <div>
    <ejs-gantt ref='gantt' id="GanttContainer" :dataSource="data"
:taskFields = "taskFields" :height = "height"></ejs-gantt>
  </div>
</template>
<script>
import Vue from "vue";
import { GanttPlugin } from "@syncfusion/ej2-vue-gantt";
Vue.use(GanttPlugin);
export default {
  data: function() {
    return{
      data: [
        {
          TaskID: 1,
          TaskName: 'Project Initiation',
          StartDate: new Date('04/02/2019'),
          EndDate: new Date('04/21/2019'),
          isParent:true,
          subtasks: [
            { TaskID: 2, TaskName: 'Identify Site location', StartDate:
new Date('04/02/2019'), Duration: 0, Progress: 50,isParent:false },
            { TaskID: 3, TaskName: 'Perform Soil test', StartDate: new
Date('04/02/2019'), Duration: 4, Progress: 70, resources: [2, 3,
5],isParent:false },
            { TaskID: 4, TaskName: 'Soil test approval', StartDate: new
Date('04/02/2019'), Duration: 4,Predecessor:"2FS", Progress:
50,isParent:false },
          ]
        },
        {
          TaskID: 5,
          TaskName: 'Project Estimation',
          StartDate: new Date('04/02/2019'),
          EndDate: new Date('04/21/2019'),
          isParent:true,
          subtasks: [
            { TaskID: 6, TaskName: 'Develop floor plan for estimation',
StartDate: new Date('04/04/2019'), Duration: 3, Progress: 50, resources:
[4],isParent:false },
```



```

        { TaskID: 7, TaskName: 'List materials', StartDate: new
Date('04/04/2019'), Duration: 3, Progress: 80, resources: [4,
8],isParent:false },
        { TaskID: 8, TaskName: 'Estimation approval', StartDate: new
Date('04/04/2019'), Duration: 0,Predecessor:"6SS", Progress: 70, resources:
[12, 5],isParent:false }
    ]
},
],
    height: '450px',
    taskFields: {
        id: 'TaskID',
        name: 'TaskName',
        startDate: 'StartDate',
        duration: 'Duration',
        progress: 'Progress',
        child: 'subtasks'
    }
};
},
};
</script>
<style>
    .e-altrow, tr.e-chart-row:nth-child(even) {
        background-color: #f2f2f2;
    }
</style>

```

{% previewsample "page.domainurl/code-snippet/gantt/rows-cs12" %}

### Row spanning

Gantt chart has an option to span row cells. You can achieve this using [rowSpan](#) attribute to span cells in the [QueryCellInfo](#) event.

In the following demo, **Soil test approval** cell is spanned to two rows in the **TaskName** column.

### APP.VUE

```

<template>
  <div>
    <ejs-gantt ref='gantt' id="GanttContainer" :dataSource="data"
:taskFields = "taskFields" :height = "height" :queryCellInfo =
"queryCellInfo"></ejs-gantt>
  </div>
</template>
<script>
import Vue from "vue";
import { GanttPlugin } from "@syncfusion/ej2-vue-gantt";
Vue.use(GanttPlugin);
export default {
  data: function() {
    return{
      data: [
        {
          TaskID: 1,
          TaskName: 'Project Initiation',
          StartDate: new Date('04/02/2019'),

```

```

        EndDate: new Date('04/21/2019'),
        isParent:true,
        subtasks: [
            { TaskID: 2, TaskName: 'Identify Site location', StartDate:
new Date('04/02/2019'), Duration: 0, Progress: 50,isParent:false },
            { TaskID: 3, TaskName: 'Perform Soil test', StartDate: new
Date('04/02/2019'), Duration: 4, Progress: 70, resources: [2, 3,
5],isParent:false },
            { TaskID: 4, TaskName: 'Soil test approval', StartDate: new
Date('04/02/2019'), Duration: 4,Predecessor:"2FS", Progress:
50,isParent:false },
        ]
    },
    {
        TaskID: 5,
        TaskName: 'Project Estimation',
        StartDate: new Date('04/02/2019'),
        EndDate: new Date('04/21/2019'),
        isParent:true,
        subtasks: [
            { TaskID: 6, TaskName: 'Develop floor plan for estimation',
StartDate: new Date('04/04/2019'), Duration: 3, Progress: 50, resources:
[4],isParent:false },
            { TaskID: 7, TaskName: 'List materials', StartDate: new
Date('04/04/2019'), Duration: 3, Progress: 80, resources: [4,
8],isParent:false },
            { TaskID: 8, TaskName: 'Estimation approval', StartDate: new
Date('04/04/2019'), Duration: 0,Predecessor:"6SS", Progress: 70, resources:
[12, 5],isParent:false }
        ]
    },
],
    height: '450px',
    taskFields: {
        id: 'TaskID',
        name: 'TaskName',
        startDate: 'StartDate',
        duration: 'Duration',
        progress: 'Progress',
        child: 'subtasks'
    }
    queryCellInfo: function(args) {
        if (args.data['TaskID'] == 4 && args.column.field ===
'TaskName') {
            args.rowSpan = 2;
        }
    }
};
},
};
};
</script>

```

{% previewsample "page.domainurl/code-snippet/gantt/rows-cs13" %}

*Customize rows and cells*

While rendering the TreeGrid part in Gantt, the [rowDataBound](#) and [queryCellInfo](#) events trigger for every row and cell. Using these events, you can customize the rows and cells. The following code example shows how to customize the cell and row elements using these events.

**APP.VUE**

```
<template>
  <div>
    <ejs-gantt ref='gantt' id="GanttContainer" :dataSource="data"
      :taskFields = "taskFields" :height = "height" :columns = "columns"
      :splitterSettings = "splitterSettings" :rowDataBound = "rowDataBound"
      :queryCellInfo = 'queryCellInfo'></ejs-gantt>
  </div>
</template>
<script>
import Vue from "vue";
import { GanttPlugin } from "@syncfusion/ej2-vue-gantt";
import { editingData } from './data-source.js';
Vue.use(GanttPlugin);
export default {
  data: function() {
    return{
      data: editingData,
      height: '450px',
      taskFields: {
        id: 'TaskID',
        name: 'TaskName',
        startDate: 'StartDate',
        duration: 'Duration',
        dependency: 'Predecessor',
        progress: 'Progress',
        child: 'subtasks'
      },
      columns: [
        { field: 'TaskID', headerText: 'Task ID', textAlign: 'Left',
width: '100' },
        { field: 'TaskName', headerText: 'Task Name', width: '150' },
        { field: 'Progress', headerText: 'Progress', width: '150' },
        { field: 'StartDate', headerText: 'Start Date', width: '150' },
        { field: 'Duration', headerText: 'Duration', width: '150' },
      ],
      splitterSettings:{
        columnIndex:3
      },
      queryCellInfo: function (args) {
        if (args.column.field == "Progress") {
          if (args.data.Progress < 25)
            args.cell.style.backgroundColor="lightgreen"
          else
            args.cell.style.backgroundColor="yellow"
        }
      },
      rowDataBound: function (args) {
        if(args.data.TaskID==4)
          args.row.style.backgroundColor="red"
      }
    }
  }
}
```

```

    };
  },
};
</script>

```

{% previewsample "page.domainurl/code-snippet/gantt/appearance-customization-cs6" %}

### Clip mode

The clip mode provides options to display its overflow cell content and it can be defined by the [columns.clipMode](#) property.

The following are three types of `clipMode`:

- **Clip**: Truncates the cell content when it overflows its area.
- **Ellipsis**: Displays ellipsis when content of the cell overflows its area.
- **EllipsisWithTooltip**: Displays ellipsis when content of the cell overflows its area; it displays the tooltip content when hover over ellipsis.

### NOTE

By default, all the column's [clipMode](#) property is defined as `EllipsisWithTooltip`.

### Cell tooltip

You can enable or disable the Grid cell tooltip using the [columns.clipMode](#) property.

### APP.VUE

```

<template>
  <div>
    <ejs-gantt ref='gantt' :dataSource='data' id='GanttContainer'
    :taskFields = "taskFields" :height = "height" :columns="columns"></ejs-
    gantt>
  </div>
</template>
<script>
import Vue from "vue";
import { GanttPlugin } from "@syncfusion/ej2-vue-gantt";
import { editingData } from './data-source.js';
Vue.use(GanttPlugin);
export default {
  data: function() {
    return{
      data: editingData,
      height: '450px',
      taskFields: {
        id: 'TaskID',
        name: 'TaskName',
        startDate: 'StartDate',
        duration: 'Duration',
        progress: 'Progress',
        child: 'subtasks'
      },
      columns: [
        { field: 'TaskID', headerText: 'Task ID', textAlign: 'Left',
width: '100' },

```

```

        { field: 'TaskName', headerText: 'Task Name', width: '150',
clipMode: 'EllipsisWithTooltip' },
        { field: 'StartDate', headerText: 'Start Date', width: '150' },
        { field: 'Duration', headerText: 'Duration', width:
'150', clipMode: 'Clip' },
        { field: 'Progress', headerText: 'Progress', width: '150' },
    ]
    };
},
};
</script>

```

{% previewsample "page.domainurl/code-snippet/gantt/tooltip-cs3" %}

### Drag and Drop in Vue Gantt component

#### Drag and drop

You can dynamically rearrange the rows in the Gantt control by using the `allowRowDragAndDrop` property. Using this property, row drag and drop can be enabled or disabled in Gantt. Using this feature, rows can be dropped at above and below as a sibling or child to the existing rows

To use row drag and drop feature, inject the `RowDD` and `Edit` modules in the `provide` section.

#### APP.VUE

```

<template>
  <div>
    <ejs-gantt :dataSource="data" :allowRowDragAndDrop='true'
:taskFields = "taskFields" :height = "height"></ejs-gantt>
  </div>
</template>
<script>
import Vue from "vue";
import { GanttPlugin, RowDD, Edit, Selection } from "@syncfusion/ej2-vue-
gantt";
import { ganttData } from "../data-source.js";
Vue.use(GanttPlugin);
export default {
  data: function() {
    return{
      data: ganttData,
      height: '450px',
      taskFields: {
        id: 'TaskID',
        name: 'TaskName',
        startDate: 'StartDate',
        duration: 'Duration',
        progress: 'Progress',
        dependency: 'Predecessor',
        child: 'subtasks'
      }
    };
  },
  provide: {
    gantt: [ RowDD, Edit, Selection ]
  }
}

```

```
};
</script>
```

{% previewsample "page.domainurl/code-snippet/gantt/rows-cs5" %}

### Multiple row drag and drop

Gantt also supports dragging multiple rows at a time and drop them on any rows above, below, or at child positions. In Gantt, you can enable the multiple drag and drop by setting the `selectionSettings.type` to `Multiple` and you should enable the `allowRowDragAndDrop` property.

### APP.VUE

```
<template>
  <div>
    <ejs-gantt :dataSource="data" :allowRowDragAndDrop='true'
    :selectionSettings="selectionSettings" :taskFields = "taskFields" :height =
    "height"></ejs-gantt>
  </div>
</template>
<script>
import Vue from "vue";
import { GanttPlugin, RowDD, Edit, Selection } from "@syncfusion/ej2-vue-gantt";
import { ganttData } from "../data-source.js";
Vue.use(GanttPlugin);
export default {
  data: function() {
    return{
      data: ganttData,
      height: '450px',
      taskFields: {
        id: 'TaskID',
        name: 'TaskName',
        startDate: 'StartDate',
        duration: 'Duration',
        progress: 'Progress',
        dependency: 'Predecessor',
        child: 'subtasks'
      },
      selectionSettings: {
        type: 'Multiple'
      }
    };
  },
  provide: {
    gantt: [ RowDD, Edit, Selection ]
  }
};
</script>
```

{% previewsample "page.domainurl/code-snippet/gantt/rows-cs6" %}

### Taskbar drag and drop between rows

The Gantt feature empowers users to efficiently reorganize records by seamlessly moving taskbar and rearranging their positions through a simple drag-and-drop action. Using this feature, rows can be dropped at above and below as a sibling or child to the existing rows.

This mode can be enable by setting the [allowTaskbarDragAndDrop](#) property to `true`.

To use row drag and drop feature, inject the `RowDD` and `Edit` module in Gantt.

### APP.VUE

```
<template>
  <div>
    <ejs-gantt ref='gantt' id="GanttContainer" :dataSource="data"
    :taskFields = "taskFields" :height = "height" :allowRowDragAndDrop="true"
    :allowTaskbarDragAndDrop="true" :editSettings="editSettings"></ejs-gantt>
  </div>
</template>
<script>
import Vue from "vue";
import { GanttPlugin, Edit, RowDD } from "@syncfusion/ej2-vue-gantt";
Vue.use(GanttPlugin);
export default {
  data: function() {
    return {
      data: [
        {
          TaskID: 1,
          TaskName: 'Project Initiation',
          StartDate: new Date('04/02/2019'),
          EndDate: new Date('04/21/2019'),
          isParent:true,
          subtasks: [
            { TaskID: 2, TaskName: 'Identify Site location', StartDate:
new Date('04/02/2019'), Duration: 0, Progress: 50,isParent:false },
            { TaskID: 3, TaskName: 'Perform Soil test', StartDate: new
Date('04/02/2019'), Duration: 4, Progress: 70, resources: [2, 3,
5],isParent:false },
            { TaskID: 4, TaskName: 'Soil test approval', StartDate: new
Date('04/02/2019'), Duration: 4,Predecessor:"2FS", Progress:
50,isParent:false },
          ]
        },
        {
          TaskID: 5,
          TaskName: 'Project Estimation',
          StartDate: new Date('04/02/2019'),
          EndDate: new Date('04/21/2019'),
          isParent:true,
          subtasks: [
            { TaskID: 6, TaskName: 'Develop floor plan for estimation',
StartDate: new Date('04/04/2019'), Duration: 3, Progress: 50, resources:
[4],isParent:false },
            { TaskID: 7, TaskName: 'List materials', StartDate: new
Date('04/04/2019'), Duration: 3, Progress: 80, resources: [4,
8],isParent:false },
          ]
        }
      ]
    }
  }
}
```

```

        { TaskID: 8, TaskName: 'Estimation approval', StartDate: new
Date('04/04/2019'), Duration: 0,Predecessor:"6SS", Progress: 70, resources:
[12, 5],isParent:false }
    ],
    height: '450px',
    taskFields: {
        id: 'TaskID',
        name: 'TaskName',
        startDate: 'StartDate',
        duration: 'Duration',
        progress: 'Progress',
        child: 'subtasks'
    },
    editSettings : {
        allowAdding: true,
        allowEditing: true,
        allowDeleting: true,
        allowTaskbarEditing: true,
        showDeleteConfirmDialog: true
    }
};
},
provide: {
    gantt: [ Edit, RowDD]
}
};
</script>

```

{% previewsample "page.domainurl/code-snippet/gantt/rows-cs14" %}

### Drag and drop events

We provide various events to customize the row drag and drop action, the following table explains about the available events and its details.

#### Event Name | Description

**rowDragStartHelper** | Triggers when clicking the drag icon or Gantt row.

**rowDragStart** | Triggers when drag action starts in Gantt.

**rowDrag** | Triggers while dragging the Gantt row.

**rowDrop** | Triggers when a drag row was dropped on the target row.

### Customize row drag and drop action

In Gantt, the **rowDragStartHelper** and **rowDrop** events are triggered on row drag and drop action. Using this event, you can prevent dragging of particular record, validate the drop position, and cancel the drop action based on the target record and dragged record. The following topics explains about this.

#### Prevent dragging of particular record

You can prevent drag action of the particular record by setting the **cancel** property to **true**, which is available in the **rowDragStartHelper** event argument based on our requirement. In the following sample, drag action was restricted for first parent record and its child records.



**APP.VUE**

```

<template>
  <div>
    <ejs-gantt :dataSource="data" :allowRowDragAndDrop='true'
    :taskFields = "taskFields" :height = "height"
    :rowDragStartHelper="rowDragStartHelper"></ejs-gantt>
  </div>
</template>
<script>
import Vue from "vue";
import { GanttPlugin, RowDD, Edit, Selection } from "@syncfusion/ej2-vue-gantt";
import { ganttData } from "../data-source.js";
Vue.use(GanttPlugin);
export default {
  data: function() {
    return {
      data: ganttData,
      height: '450px',
      taskFields: {
        id: 'TaskID',
        name: 'TaskName',
        startDate: 'StartDate',
        duration: 'Duration',
        progress: 'Progress',
        dependency: 'Predecessor',
        child: 'subtasks'
      },
      rowDragStartHelper: function(args) {
        var record = args.data[0] ? args.data[0] : args.data;
        var taskId = record.ganttProperties.taskId;
        if (taskId <= 4) {
          args.cancel = true;
        }
      }
    };
  },
  provide: {
    gantt: [ RowDD, Edit, Selection ]
  }
};
</script>

```

{% previewsample "page.domainurl/code-snippet/gantt/rows-cs7" %}

**Validating drop position**

You can prevent drop action based on the drop position and target record, by this, you can prevent dropping particular task on a specific task or specific position. This can be achieved by setting the **cancel** property to **true**, which is available in the **rowDrop** event argument.

In the following sample, we have prevented the drop action based on the position. In this sample, you cannot drop row as child in any of the available rows.

**APP.VUE**

```

<template>
  <div>
    <ejs-gantt :dataSource="data" :allowRowDragAndDrop='true'
:taskFields = "taskFields" :height = "height" :rowDrop="rowDrop"></ejs-
gantt>
  </div>
</template>
<script>
import Vue from "vue";
import { GanttPlugin, RowDD, Edit, Selection } from "@syncfusion/ej2-vue-
gantt";
import { ganttData } from "../data-source.js";
Vue.use(GanttPlugin);
export default {
  data: function() {
    return{
      data: ganttData,
      height: '450px',
      taskFields: {
        id: 'TaskID',
        name: 'TaskName',
        startDate: 'StartDate',
        duration: 'Duration',
        progress: 'Progress',
        dependency: 'Predecessor',
        child: 'subtasks'
      },
      rowDrop: function(args) {
        if (args.dropPosition == "middleSegment") {
          args.cancel = true;
        }
      }
    };
  },
  provide: {
    gantt: [ RowDD, Edit, Selection ]
  }
};
</script>

```

{% previewsample "page.domainurl/code-snippet/gantt/rows-cs8" %}

#### Prevent reordering a row as child to another row

You can prevent the default behavior of dropping rows as children to the target by setting the `cancel` property to `true` in `rowDrop` event argument. You can also change the drop position after cancelling using `reorderRows` method.

In the below example drop action is cancelled and dropped above to target row.

#### APP.VUE

```

<template>
  <div>
    <ejs-gantt ref='gantt' :dataSource="data"
:allowRowDragAndDrop='true' :taskFields = "taskFields" :height = "height"
:rowDrop="rowDrop"></ejs-gantt>

```

```

</div>
</template>
<script>
import Vue from "vue";
import { GanttPlugin, RowDD, Edit, Selection } from "@syncfusion/ej2-vue-gantt";
import { ganttData } from "../data-source.js";
Vue.use(GanttPlugin);
export default {
  data: function() {
    return {
      data: ganttData,
      height: '450px',
      taskFields: {
        id: 'TaskID',
        name: 'TaskName',
        startDate: 'StartDate',
        duration: 'Duration',
        progress: 'Progress',
        dependency: 'Predecessor',
        child: 'subtasks'
      }
    };
  },
  provide: {
    gantt: [ RowDD, Edit, Selection ]
  },
  methods: {
    rowDrop: function(args) {
      if (args.dropPosition == 'middleSegment') {
        args.cancel = true;
        this.$refs.gantt.reorderRows([args.fromIndex],
args.dropIndex, 'above');
      }
    }
  }
};
</script>

```

{% previewsample "page.domainurl/code-snippet/gantt/rows-cs9" %}

#### Perform row drag and drop action programmatically

Gantt provides option to perform row drag and drop action programmatically by using the `reorderRows` method, this method can be used for any external actions like button click.

The following arguments are used to specify the positions to drag and drop a row:

- **fromIndexes**: Index value of source(dragging) row.
- **toIndex**: Value of target index.
- **position**: Drop positions such as above, below, or child.

The following code example shows how to drag and drop a row on button click action.

#### APP.VUE

```

<template>
  <div>
    <ejs-button id="dynamicDrag" cssClass="e-info" v-
on:click.native="dynamicDrag">Drop records as child</ejs-button>
    <br><br>
    <ejs-gantt ref='gantt' :dataSource="data"
:allowRowDragAndDrop='true' :taskFields = "taskFields" :height =
"height"></ejs-gantt>
  </div>
</template>
<script>
import Vue from "vue";
import { GanttPlugin, RowDD, Edit, Selection } from "@syncfusion/ej2-vue-
gantt";
import { ButtonPlugin } from "@syncfusion/ej2-vue-buttons";
import { ganttData } from "../data-source.js";
Vue.use(GanttPlugin);
Vue.use(ButtonPlugin);
export default {
  data: function() {
    return{
      data: ganttData,
      height: '450px',
      taskFields: {
        id: 'TaskID',
        name: 'TaskName',
        startDate: 'StartDate',
        duration: 'Duration',
        progress: 'Progress',
        dependency: 'Predecessor',
        child: 'subtasks'
      }
    };
  },
  provide: {
    gantt: [ RowDD, Edit, Selection ]
  },
  methods: {
    dynamicDrag: function(e) {
      this.$refs.gantt.reorderRows([1,2,3], 4, 'child');
    }
  }
};
</script>

```

{% previewsample "page.domainurl/code-snippet/gantt/rows-cs10" %}

## Export

### Excel Export

#### *Excel export in Vue Gantt component*

Gantt supports client-side exporting, which allows you to export its data to the Excel and CSV formats. Use the [excelExport](#) and [csvExport](#) methods for exporting. To enable Excel export in the Gantt, set the [Link to the Video](#) to true.

To get a configured Gantt chart Exporting, you can follow the steps outlined in the video linked below:

To export data to Excel and CSV, inject the `ExcelExport` module in Gantt.

### APP.VUE

```
<template>
  <div>
    <ejs-gantt ref='gantt' id="GanttContainer" :dataSource="data"
    :taskFields="taskFields" :toolbar="toolbar" :toolbarClick="toolbarClick"
    :allowExcelExport='true' :height="height"></ejs-gantt>
  </div>
</template>
<script>
import Vue from "vue";
import { GanttPlugin, Toolbar, ExcelExport, Selection } from
"@syncfusion/ej2-vue-gantt";
import { ClickEventArgs } from '@syncfusion/ej2-navigations';
import { ganttData } from './data-source.js';
Vue.use(GanttPlugin);
export default {
  data: function() {
    return{
      data: ganttData,
      height: '450px',
      taskFields: {
        id: 'TaskID',
        name: 'TaskName',
        startDate: 'StartDate',
        duration: 'Duration',
        progress: 'Progress',
        child: 'subtasks'
      },
      toolbar: ['ExcelExport', 'CsvExport'],
      toolbarClick: (args: ClickEventArgs) => {
        if (args.item.id === 'GanttContainer_excelexport') {
          var ganttObj =
document.getElementById('GanttContainer').ej2_instances[0];
          ganttObj.excelExport();
        } else if (args.item.id === 'GanttContainer_csvexport') {
          var ganttObj =
document.getElementById('GanttContainer').ej2_instances[0];
          ganttObj.csvExport();
        }
      },
    };
  },
  provide: {
    gantt: [Toolbar, ExcelExport]
  }
};
</script>
```

{% previewsample "page.domainurl/code-snippet/gantt/excel-export-cs2" %}

#### Custom data source in Vue Gantt component

The excel export provides an option to define `datasource` dynamically before exporting. To export data dynamically, define the `dataSource` in `exportProperties`.

**APP.VUE**

```

<template>
  <div>
    <ejs-gantt ref='gantt' id="GanttContainer" :dataSource="data"
    :taskFields="taskFields" :toolbar="toolbar" :toolbarClick="toolbarClick"
    :allowExcelExport='true' :height="height" :treeColumnIndex="1"></ejs-gantt>
  </div>
</template>
<script>
import Vue from "vue";
import { GanttPlugin, Toolbar, ExcelExport, Selection } from
"@syncfusion/ej2-vue-gantt";
import { ClickEventArgs } from '@syncfusion/ej2-navigations';
import { ganttData } from './data-source.js';
Vue.use(GanttPlugin);
export default {
  data: function() {
    return{
      data: ganttData,
      height: '450px',
      taskFields: {
        id: 'TaskID',
        name: 'TaskName',
        startDate: 'StartDate',
        duration: 'Duration',
        progress: 'Progress',
        child: 'subtasks'
      },
      toolbar: ['ExcelExport', 'CsvExport'],
      toolbarClick: (args: ClickEventArgs) => {
        if (args.item.id === 'GanttContainer_excelexport') {
          var excelExportProperties: ExcelExportProperties = {
            dataSource: ganttData[1];
          };
          var ganttObj =
document.getElementById('GanttContainer').ej2_instances[0];
          ganttObj.excelExport(excelExportProperties);
        }
      },
    };
  },
  provide: {
    gantt: [Toolbar, ExcelExport]
  }
};
</script>

```

{% previewsample "page.domainurl/code-snippet/gantt/excel-export-cs1" %}

### [Multiple gantt exporting in Vue Gantt component](#)

In Gantt, the Excel export provides support to export multiple Gantt data in new sheet or same sheet.

#### [Same sheet](#)

The Excel export provides support to export multiple Gantt data in the same sheet. To export in same sheet, define `multipleExport.type` as `AppendToSheet` in `ExcelExportProperties`. You can also

provide blank rows between exported multiple Gantt data. These blank rows count can be defined using `multipleExport.blankRows`.

### APP.VUE

```
<template>
  <div>
    <p><b>First Gantt:</b></p>
    <ejs-gantt ref='gantt1' id="GanttContainer1" :dataSource="fData"
:taskFields="taskFields" :toolbar="toolbar" :toolbarClick="toolbarClick"
:allowExcelExport='true' :height="height1" :treeColumnIndex="1"
:projectStartDate="projectStartDate" :projectEndDate="projectEndDate"></ejs-
gantt>
    <p><b>Second Gantt:</b></p>
    <ejs-gantt ref='gantt2' id="GanttContainer2" :dataSource="sData"
:taskFields="taskFields" :allowExcelExport='true' :height="height2"
:treeColumnIndex="1"></ejs-gantt>
  </div>
</template>
<script>
import Vue from "vue";
import { GanttPlugin, Toolbar, ExcelExport, Selection } from
"@syncfusion/ej2-vue-gantt";
import { ClickEventArgs } from '@syncfusion/ej2-navigations';
import { ganttData } from './data-source.js';
Vue.use(GanttPlugin);
export default {
  data: function() {
    return{
      fData: [ganttData[0]],
      sData: [ganttData[1]],
      height1: '280px',
      height2: '250px',
      taskFields: {
        id: 'TaskID',
        name: 'TaskName',
        startDate: 'StartDate',
        duration: 'Duration',
        progress: 'Progress',
        child: 'subtasks'
      },
      toolbar: ['ExcelExport'],
      projectStartDate: new Date('03/31/2019'),
      projectEndDate: new Date('04/14/2019'),
      toolbarClick: (args: ClickEventArgs) => {
        if (args.item.id === 'GanttContainer1_excelexport') {
          var appendExcelExportProperties: ExcelExportProperties =
{
          multipleExport: {type: 'AppendToSheet',blankRows: 2}
        };
          var FirstGantt =
document.getElementById('GanttContainer1').ej2_instances[0];
          var FirstGanttExport =
FirstGantt.excelExport(appendExcelExportProperties,true);
          FirstGanttExport.then((fData) => {
            var SecondGantt =
document.getElementById('GanttContainer2').ej2_instances[0];
```

```

SecondGantt.excelExport(appendExcelExportProperties, false, fData);
    });
    },
    };
},
provide: {
    gantt: [Toolbar, ExcelExport]
}
};
</script>

```

{% previewsample "page.domainurl/code-snippet/gantt/excel-export-cs3" %}

By default, `multipleExport.blankRows` value is 5.

#### [New sheet](#)

The Excel exporting provides support to export multiple Gantt in new sheet. To export in new sheet, define `multipleExport.type` as `NewSheet` in `ExcelExportProperties`.

#### **APP.VUE**

```

<template>
  <div>
    <p><b>First Gantt:</b></p>
    <ejs-gantt ref='ganttl1' id="GanttContainer1" :dataSource="fData"
:taskFields="taskFields" :toolbar="toolbar" :toolbarClick="toolbarClick"
:allowExcelExport='true' :height="height1" :treeColumnIndex="1"
:projectStartDate="projectStartDate" :projectEndDate="projectEndDate"></ejs-
gantt>
    <p><b>Second Gantt:</b></p>
    <ejs-gantt ref='ganttl2' id="GanttContainer2" :dataSource="sData"
:taskFields="taskFields" :allowExcelExport='true' :height="height2"
:treeColumnIndex="1"></ejs-gantt>
  </div>
</template>
<script>
import Vue from "vue";
import { GanttPlugin, Toolbar, ExcelExport, Selection } from
"@syncfusion/ej2-vue-gantt";
import { ClickEventArgs } from '@syncfusion/ej2-navigations';
import { ganttData } from './data-source.js';
Vue.use(GanttPlugin);
export default {
  data: function() {
    return{
      fData: [ganttlData[0]],
      sData: [ganttlData[1]],
      height1: '280px',
      height2: '250px',
      taskFields: {
        id: 'TaskID',
        name: 'TaskName',
        startDate: 'StartDate',
        duration: 'Duration',

```



```

        progress: 'Progress',
        child: 'subtasks'
    },
    toolbar: ['ExcelExport'],
    projectStartDate: new Date('03/31/2019'),
    projectEndDate: new Date('04/14/2019'),
    toolbarClick: (args: ClickEventArgs) => {
        if (args.item.id === 'GanttContainer1_excelexport') {
            var appendExcelExportProperties: ExcelExportProperties =
{
                multipleExport: {type: 'NewSheet'}
            };
            var FirstGantt =
document.getElementById('GanttContainer1').ej2_instances[0];
            var FirstGanttExport =
FirstGantt.excelExport(appendExcelExportProperties, true);
            FirstGanttExport.then((fData) => {
                var SecondGantt =
document.getElementById('GanttContainer2').ej2_instances[0];
                SecondGantt.excelExport(appendExcelExportProperties, false, fData);
            });
        }
    },
    },
    },
    provide: {
        gantt: [Toolbar, ExcelExport]
    }
};
</script>

```

{% previewsample "page.domainurl/code-snippet/gantt/excel-export-cs4" %}

### Customize the Excel export

Gantt Excel export allows the users to customize the exported document based on requirement.

### Export hidden columns

In Gantt, the Excel export provides an option to export hidden columns by defining `includeHiddenColumn` as `true`.

### APP.VUE

```

<template>
  <div>
    <ejs-gantt ref='gantt' id="GanttContainer" :dataSource="data"
:taskFields="taskFields" :columns="columns" :toolbar="toolbar"
:toolbarClick="toolbarClick" :allowExcelExport='true' :height="height"
:treeColumnIndex="1"></ejs-gantt>
  </div>
</template>
<script>
import Vue from "vue";
import { GanttPlugin, Toolbar, ExcelExport, Selection } from
"@syncfusion/ej2-vue-gantt";
import { ClickEventArgs } from '@syncfusion/ej2-navigations';

```

```

import { ganttData } from './data-source.js';
Vue.use(GanttPlugin);
export default {
  data: function() {
    return{
      data: ganttData,
      height: '450px',
      taskFields: {
        id: 'TaskID',
        name: 'TaskName',
        startDate: 'StartDate',
        duration: 'Duration',
        progress: 'Progress',
        child: 'subtasks'
      },
      columns: [
        { field: 'TaskID', headerText: 'Task ID', textAlign: 'Left',
width: '100' },
        { field: 'TaskName', headerText: 'Task Name', width: '150' },
        { field: 'StartDate', headerText: 'Start Date', width: '150',
visible: false },
        { field: 'Duration', headerText: 'Duration', width: '150' },
        { field: 'Progress', headerText: 'Progress', width: '150' },
      ],
      toolbar: ['ExcelExport', 'CsvExport'],
      toolbarClick: (args: ClickEventArgs) => {
        if (args.item.id === 'GanttContainer_excelexport') {
          var excelExportProperties: ExcelExportProperties = {
            includeHiddenColumn: true
          };
          var ganttObj =
document.getElementById('GanttContainer').ej2_instances[0];
          ganttObj.excelExport(excelExportProperties);
        } else if (args.item.id === 'GanttContainer_csvexport') {
          var excelExportProperties: ExcelExportProperties = {
            includeHiddenColumn: true
          };
          var ganttObj =
document.getElementById('GanttContainer').ej2_instances[0];
          ganttObj.csvExport(excelExportProperties);
        }
      },
    };
  },
  provide: {
    gantt: [Toolbar, ExcelExport]
  }
};
</script>

```

{% previewsample "page.domainurl/code-snippet/gantt/excel-export-cs5" %}

Show or hide columns on exported Excel

In Gantt, while exporting, you can show a hidden column or hide a visible column using the [toolbarClick](#) and [excelExportComplete](#) events.

In the `toolbarClick` event, using the `args.item.id` property, you can show or hide columns by setting the `column.visible` property to `true` or `false` respectively.

Similarly, in the `excelExportComplete` event, you can revert the columns visibility back to the previous state.

### APP.VUE

```
<template>
  <div>
    <ejs-gantt ref='gantt' id="GanttContainer" :dataSource="data"
    :taskFields="taskFields" :columns="columns"
    :toolbar="toolbar" :toolbarClick="toolbarClick"
    :excelExportComplete="excelExportComplete" :allowExcelExport='true'
    :height="height" :treeColumnIndex="1"></ejs-gantt>
  </div>
</template>
<script>
import Vue from "vue";
import { GanttPlugin, Toolbar, ExcelExport, Selection } from
"@syncfusion/ej2-vue-gantt";
import { ClickEventArgs } from "@syncfusion/ej2-navigations";
import { ganttData } from './data-source.js';
Vue.use(GanttPlugin);
export default {
  data: function() {
    return{
      data: ganttData,
      height: '450px',
      taskFields: {
        id: 'TaskID',
        name: 'TaskName',
        startDate: 'StartDate',
        duration: 'Duration',
        progress: 'Progress',
        child: 'subtasks'
      },
      columns: [
        { field: 'TaskID', headerText: 'Task ID', textAlign: 'Left',
width: '100' },
        { field: 'TaskName', headerText: 'Task Name', width: '150' },
        { field: 'StartDate', headerText: 'Start Date', width: '150',
visible: false },
        { field: 'Duration', headerText: 'Duration', width: '150' },
        { field: 'Progress', headerText: 'Progress', width: '150' },
      ],
      toolbar: ['ExcelExport', 'CsvExport'],
      toolbarClick: (args: ClickEventArgs) => {
        if (args.item.id === 'GanttContainer_excelexport') {
          var ganttObj =
document.getElementById('GanttContainer').ej2_instances[0];
          ganttObj.treeGrid.grid.columns[0].visible = true;
          ganttObj.treeGrid.grid.columns[3].visible = false;
          ganttObj.excelExport();
        } else if (args.item.id === 'GanttContainer_csvexport') {
          var ganttObj =
document.getElementById('GanttContainer').ej2_instances[0];
```

```

        ganttObj.treeGrid.grid.columns[0].visible = true;
        ganttObj.treeGrid.grid.columns[3].visible = false;
        ganttObj.csvExport();
    },
    },
    excelExportComplete: () => {
        var ganttObj =
document.getElementById('GanttContainer').ej2_instances[0];
        ganttObj.treeGrid.grid.columns[0].visible = false;
        ganttObj.treeGrid.grid.columns[3].visible = true;
    },
    };
},
provide: {
    gantt: [Toolbar, ExcelExport]
}
};
</script>

```

{% previewsample "page.domainurl/code-snippet/gantt/excel-export-cs6" %}

#### Cell formatting during export

In Gantt, you can customize the TreeGrid cells in the exported document using the [excelQueryCellInfo](#) event. In this event, you can format the TreeGrid cells of exported Excel and CSV documents based on the required condition.

In the following sample, the background color has been customized for **TaskID** column in the exported Excel using the **args.style** and **backColor** properties.

#### APP.VUE

```

<template>
  <div>
    <ejs-gantt ref='gantt' id="GanttContainer" :dataSource="data"
:taskFields="taskFields" :toolbar="toolbar" :toolbarClick="toolbarClick"
:excelQueryCellInfo="excelQueryCellInfo" :queryCellInfo="queryCellInfo"
:queryTaskbarInfo = "queryTaskbarInfo" :allowExcelExport='true'
:height="height" :treeColumnIndex="1" :columns = "columns"
:labelSettings="labelSettings" :splitterSettings= "splitterSettings"></ejs-
gantt>
  </div>
</template>
<script>
import Vue from "vue";
import { GanttPlugin, Toolbar, ExcelExport, Selection } from
"@syncfusion/ej2-vue-gantt";
import { ClickEventArgs } from '@syncfusion/ej2-navigations';
import { ganttData } from './data-source.js';
Vue.use(GanttPlugin);
export default {
  data: function() {
    return{
      data: ganttData,
      height: '450px',
      taskFields: {
        id: 'TaskID',

```

```

        name: 'TaskName',
        startDate: 'StartDate',
        duration: 'Duration',
        progress: 'Progress',
        child: 'subtasks'
    },
    toolbar: ['ExcelExport'],
    columns: [
        { field: 'TaskID', headerText: 'Task ID', textAlign: 'Left',
width: '100', visible: false },
        { field: 'TaskName', headerText: 'Task Name', width: '150' },
        { field: 'Progress', headerText: 'Progress', width: '150' },
        { field: 'StartDate', headerText: 'Start Date', width: '150' },
        { field: 'Duration', headerText: 'Duration', width: '150' }
    ],
    labelSettings: {
        taskLabel: '${Progress}%'
    },
    splitterSettings: {
        columnIndex: 3
    },
    toolbarClick: (args: ClickEventArgs) => {
        if (args.item.id === 'GanttContainer_excelexport') {
            var ganttObj =
document.getElementById('GanttContainer').ej2_instances[0];
            ganttObj.excelExport();
        }
    },
    excelQueryCellInfo: (args: any) => {
        if (args.column.field === 'Progress') {
            if (args.value > 80) {
                args.style = { backgroundColor: '#A569BD' };
            }
            else if (args.value < 20) {
                args.style = { backgroundColor: '#F08080' };
            }
        }
    },
    queryTaskbarInfo: function(args) {
        if (args.data.Progress > 80) {
            args.progressBarBgColor = "#6C3483";
            args.taskbarBgColor = args.taskbarBorderColor = "#A569BD";
        } else if (args.data.Progress < 20) {
            args.progressBarBgColor = "#CD5C5C";
            args.taskbarBgColor = args.taskbarBorderColor = "#F08080";
        }
    },
    queryCellInfo: (args: any) => {
        if (args.column.field === 'Progress') {
            if (args.data.Progress > 80) {
                args.cell.style.backgroundColor = '#A569BD';
            }
            else if (args.data.Progress < 20) {
                args.cell.style.backgroundColor = '#F08080';
            }
        }
    },
},

```

```

    };
    },
    provide: {
      gantt: [Toolbar, ExcelExport]
    }
  };
</script>

```

{% previewsample "page.domainurl/code-snippet/gantt/excel-export-cs7" %}

### Theme

The Excel export also provides an option to include custom theme for exported Excel document.

To apply theme in exported Excel, define the **theme** in **ExcelExportProperties**.

### APP.VUE

```

<template>
  <div>
    <ejs-gantt ref='gantt' id="GanttContainer" :dataSource="data"
    :taskFields="taskFields" :toolbar="toolbar" :toolbarClick="toolbarClick"
    :allowExcelExport='true' :height = "height" :treeColumnIndex="1"></ejs-
    gantt>
  </div>
</template>
<script>
import Vue from "vue";
import { GanttPlugin, Toolbar, ExcelExport, Selection } from
"@syncfusion/ej2-vue-gantt";
import { ClickEventArgs } from '@syncfusion/ej2-navigations';
import { ganttData } from './data-source.js';
Vue.use(GanttPlugin);
export default {
  data: function() {
    return{
      data: ganttData,
      height: '450px',
      taskFields: {
        id: 'TaskID',
        name: 'TaskName',
        startDate: 'StartDate',
        duration: 'Duration',
        progress: 'Progress',
        child: 'subtasks'
      },
      toolbar: ['ExcelExport'],
      toolbarClick: (args: ClickEventArgs) => {
        if (args.item.id === 'GanttContainer_excelexport') {
          var excelExportProperties: ExcelExportProperties = {
            theme: {
              header: { fontName: 'Segoe UI', fontColor:
'#666666' },
              record: { fontName: 'Segoe UI', fontColor:
'#666666' }
            }
          };
        }
      }
    };
  }
};

```

```

        var ganttObj =
document.getElementById('GanttContainer').ej2_instances[0];
        ganttObj.excelExport(excelExportProperties);
    },
    },
    };
    },
    provide: {
        gantt: [Toolbar, ExcelExport]
    }
    };
</script>

```

{% previewsample "page.domainurl/code-snippet/gantt/excel-export-cs8" %}

By default, material theme is applied to the exported Excel document.

#### Add header and footer

The Excel export also allows users to include header and footer contents to the exported Excel document.

#### APP.VUE

```

<template>
  <div>
    <ejs-gantt ref='gantt' id="GanttContainer" :dataSource="data"
:taskFields="taskFields" :toolbar="toolbar" :toolbarClick="toolbarClick"
:allowExcelExport='true' :height="height" :treeColumnIndex="1"></ejs-gantt>
  </div>
</template>
<script>
import Vue from "vue";
import { GanttPlugin, Toolbar, ExcelExport, Selection } from
"@syncfusion/ej2-vue-gantt";
import { ClickEventArgs } from '@syncfusion/ej2-navigations';
import { ganttData } from './data-source.js';
Vue.use(GanttPlugin);
export default {
  data: function() {
    return{
      data: ganttData,
      height: '450px',
      taskFields: {
        id: 'TaskID',
        name: 'TaskName',
        startDate: 'StartDate',
        duration: 'Duration',
        progress: 'Progress',
        child: 'subtasks'
      },
      toolbar: ['ExcelExport'],
      toolbarClick: (args: ClickEventArgs) => {
        if (args.item.id === 'GanttContainer_excelexport') {
          var excelExportProperties: ExcelExportProperties = {
            header: {
              headerRows: 7,

```

```

        rows: [
            { cells: [{ colSpan: 4, value: "Northwind
Traders", style: { fontColor: '#C67878', fontSize: 20, hAlign: 'Center',
bold: true, } }] },
            { cells: [{ colSpan: 4, value: "2501 Aerial
Center Parkway", style: { fontColor: '#C67878', fontSize: 15, hAlign:
'Center', bold: true, } }] },
            { cells: [{ colSpan: 4, value: "Suite 200
Morrisville, NC 27560 USA", style: { fontColor: '#C67878', fontSize: 15,
hAlign: 'Center', bold: true, } }] },
            { cells: [{ colSpan: 4, value: "Tel +1
888.936.8638 Fax +1 919.573.0306", style: { fontColor: '#C67878', fontSize:
15, hAlign: 'Center', bold: true, } }] },
            { cells: [{ colSpan: 4, hyperlink: { target:
'https://www.northwind.com/', displayText: 'www.northwind.com' }, style: {
hAlign: 'Center' } }] },
            { cells: [{ colSpan: 4, hyperlink: { target:
'mailto:support@northwind.com' }, style: { hAlign: 'Center' } }] },
        ],
        footer: {
            footerRows: 4,
            rows: [
                { cells: [{ colSpan: 4, value: "Thank you
for your business!", style: { hAlign: 'Center', bold: true } }] },
                { cells: [{ colSpan: 4, value: "!Visit
Again!", style: { hAlign: 'Center', bold: true } }] }
            ]
        },
    };
    var ganttObj =
document.getElementById('GanttContainer').ej2_instances[0];
    ganttObj.excelExport(excelExportProperties);
}
},
};
},
provide: {
    gantt: [Toolbar, ExcelExport]
}
};
</script>

```

{% previewsample "page.domainurl/code-snippet/gantt/excel-export-cs9" %}

File name for exported document

You can set the required file name for the exported document by defining the `fileName` property in `ExcelExportProperties`.

### APP.VUE

```

<template>
  <div>
    <ejs-gantt ref='gantt' id="GanttContainer" :dataSource="data"
:taskFields="taskFields" :toolbar="toolbar" :toolbarClick="toolbarClick"
:allowExcelExport='true' :height="height" :treeColumnIndex="1"></ejs-gantt>

```



```

    </div>
</template>
<script>
import Vue from "vue";
import { GanttPlugin, Toolbar, ExcelExport, Selection } from
"@syncfusion/ej2-vue-gantt";
import { ClickEventArgs } from '@syncfusion/ej2-navigations';
import { ganttData } from './data-source.js';
Vue.use(GanttPlugin);
export default {
  data: function() {
    return{
      data: ganttData,
      height: '450px',
      taskFields: {
        id: 'TaskID',
        name: 'TaskName',
        startDate: 'StartDate',
        duration: 'Duration',
        progress: 'Progress',
        child: 'subtasks'
      },
      toolbar: ['ExcelExport', 'CsvExport'],
      toolbarClick: (args: ClickEventArgs) => {
        if (args.item.id === 'GanttContainer_excelexport') {
          var excelExportProperties: ExcelExportProperties = {
            fileName: "Gantt.xlsx"
          };
          var ganttObj =
document.getElementById('GanttContainer').ej2_instances[0];
          ganttObj.excelExport(excelExportProperties);
        } else if (args.item.id === 'GanttContainer_csvexport') {
          var excelExportProperties: ExcelExportProperties = {
            fileName: "Gantt.csv"
          };
          var ganttObj =
document.getElementById('GanttContainer').ej2_instances[0];
          ganttObj.csvExport(excelExportProperties);
        }
      },
    };
  },
  provide: {
    gantt: [Toolbar, ExcelExport]
  }
};
</script>

```

{% previewsample "page.domainurl/code-snippet/gantt/excel-export-cs10" %}

## Pdf Export

### Export

#### Pdf export in Vue Gantt component

PDF export allows exporting Gantt data to PDF document. You need to use the [pdfExport](#) method for exporting. To enable PDF export in the Gantt, set the [Link to the Video](#) to true.

To get a configured Gantt chart for exporting, you can follow the steps outlined in the video linked below:

To export data to PDF document, inject the PdfExport module in Gantt.

Note: Currently, we do not have support for exporting manually scheduled tasks.

### APP.VUE

```
<template>
  <div>
    <ejs-gantt ref='gantt' id="GanttContainer" :dataSource="data"
    :taskFields="taskFields" :toolbar="toolbar" :toolbarClick="toolbarClick"
    :allowPdfExport='true' :height="height"></ejs-gantt>
  </div>
</template>
<script>
import Vue from "vue";
import { GanttPlugin, Toolbar, PdfExport, Selection } from "@syncfusion/ej2-
vue-gantt";
import { ClickEventArgs } from '@syncfusion/ej2-
navigations/src/toolbar/toolbar';
import { editingData } from './data-source.js';
Vue.use(GanttPlugin);
export default {
  data: function() {
    return{
      data: editingData,
      height: '450px',
      taskFields: {
        id: 'TaskID',
        name: 'TaskName',
        startDate: 'StartDate',
        duration: 'Duration',
        progress: 'Progress',
        child: 'subtasks'
      },
      toolbar: ['PdfExport'],
      toolbarClick: (args) => {
        if (args.item.id === 'GanttContainer_pdfexport') {
          var ganttChart =
document.getElementById('app').ej2_instances[0];
          ganttChart.pdfExport();
        }
      },
    };
  },
  provide: {
    gantt: [Toolbar, PdfExport]
  }
};
</script>
```

{% previewsample "page.domainurl/code-snippet/gantt/pdf-export-cs13" %}

### Indicators in PDF exporting

The PDF export functionality allows users to export Gantt charts enriched with dynamic indicators and accompanying images.

These indicators, represented by images, can be effortlessly defined using the [base64](#) encoding value in the data object of datasource. This data object field should be mapped to indicator property of [task fields](#).

### APP.VUE

```
<template>
  <div>
    <ejs-gantt ref='gantt' id="GanttContainer" :dataSource="data"
    :taskFields="taskFields" :toolbar="toolbar" :toolbarClick="toolbarClick"
    :allowPdfExport='true' :height="height"></ejs-gantt>
  </div>
</template>
<script>
import Vue from "vue";
import { GanttPlugin, Toolbar, PdfExport, Selection } from "@syncfusion/ej2-
vue-gantt";
import { ClickEventArgs } from '@syncfusion/ej2-
navigations/src/toolbar/toolbar';
import { editingData } from './data-source.js';
Vue.use(GanttPlugin);
export default {
  data: function() {
    return{
      data: editingData,
      height: '450px',
      taskFields: {
        id: 'TaskID',
        name: 'TaskName',
        startDate: 'StartDate',
        duration: 'Duration',
        progress: 'Progress',
        child: 'subtasks',
        indicators: 'Indicators'
      },
      toolbar: ['PdfExport'],
      toolbarClick: (args) => {
        if (args.item.id === 'GanttContainer_pdfexport') {
          var ganttChart =
document.getElementById('app').ej2_instances[0];
          ganttChart.pdfExport();
        }
      },
    };
  },
  provide: {
    gantt: [Toolbar, PdfExport]
  }
};
</script>
```

{% previewsample "page.domainurl/code-snippet/gantt/pdf-export-cs13" %}

### Exporting Gantt data as a blob object

In Gantt, you can export the Gantt chart data as a blob object, which allows you to preview or modify the data before exporting it.

To export the Gantt chart data as a blob object, follow these steps:

step 1: pdfExport fourth argument set as **true**.

step 2: Then , pdfExpComplete return as blob object.

### APP.VUE

```
<template>
  <div>
    <ejs-gantt ref='gantt' id="GanttContainer" :dataSource="data"
    :allowExcelExport='true' :excelExportComplete='excelExpComplete'
    :pdfExportComplete='pdfExportComplete' :taskFields="taskFields"
    :toolbar="toolbar" :toolbarClick="toolbarClick" :allowPdfExport='true'
    :height="height"></ejs-gantt>
  </div>
</template>
<script>
import Vue from "vue";
import { GanttPlugin, Toolbar, PdfExport, Selection, ExcelExport } from
"@syncfusion/ej2-vue-gantt";
import { ClickEventArgs } from '@syncfusion/ej2-
navigations/src/toolbar/toolbar';
import { projectNewData } from './data-source.js';
Vue.use(GanttPlugin);
export default {
  data: function() {
    return{
      data: projectNewData,
      height: '450px',
      taskFields: {
        id: 'TaskID',
        name: 'TaskName',
        startDate: 'StartDate',
        duration: 'Duration',
        progress: 'Progress',
        child: 'subtasks'
      },
      toolbar: ['PdfExport', 'ExcelExport'],
    };
  },
  methods: {
    toolbarClick: (args) => {
      var ganttChart =
document.getElementById('GanttContainer').ej2_instances[0];
      if (args.item.id === 'GanttContainer_pdfexport') {
        ganttChart.pdfExport(null, null, null, true);
      }
      if (args.item.id === 'GanttContainer_excelexport') {
        ganttChart.ganttObj.excelExport(excelExportProperties);
      }
    },
  },
}
```

```

        excelExpComplete: function (args) {
            //This event will be triggered when excel exporting.
            args.promise.then((e) => {
                //In this `then` function, we can get blob data through the
                arguments after promise resolved.
                this.exportBlob(e.blobData);
            });
        },
        pdfExportComplete: function (args) {
            //This event will be triggered when pdf exporting.
            args.promise.then((e) => {
                //In this `then` function, we can get blob data through the
                arguments after promise resolved.
                this.exportBlob(e.blobData);
            });
        },
        exportBlob: function (blob) {
            let a = document.createElement('a');
            document.body.appendChild(a);
            a.style.display = 'none';
            let url = window.URL.createObjectURL(blob);
            a.href = url;
            a.download = 'Export';
            a.click();
            window.URL.revokeObjectURL(url);
            document.body.removeChild(a);
        }
    },
    provide: {
        gantt: [Toolbar, PdfExport, Selection, ExcelExport]
    }
};
</script>

```

{% previewsample "page.domainurl/code-snippet/gantt/exporting-gantt-blobdata-cs1" %}

### Single page exporting in gantt

In Gantt, we have provided support to export the Gantt component where each rows are auto-fit to the PDF document page width by setting [isFitToWidth](#) as true in `fitToWidthSettings` of `PdfExportProperties`.

Also, we can customize the chart width and grid width in exported file using [chartWidth](#) and [gridWidth](#) by defining it as percentage in string.

### APP.VUE

```

<template>
  <div>
    <ejs-gantt ref='gantt' id="GanttContainer" :dataSource="data"
    :taskFields="taskFields" :toolbar="toolbar" :toolbarClick="toolbarClick"
    :allowPdfExport='true' :height="height"></ejs-gantt>
  </div>
</template>
<script>
import Vue from "vue";

```

```

import { GanttPlugin, Toolbar, PdfExport, Selection } from "@syncfusion/ej2-vue-gantt";
import { ClickEventArgs } from '@syncfusion/ej2-navigations/src/toolbar/toolbar';
import { editingData } from './data-source.js';
Vue.use(GanttPlugin);
export default {
  data: function() {
    return {
      data: editingData,
      height: '450px',
      taskFields: {
        id: 'TaskID',
        name: 'TaskName',
        startDate: 'StartDate',
        endDate: 'EndDate',
        duration: 'Duration',
        progress: 'Progress',
        dependency: 'Predecessor',
        child: 'subtasks',
      },
      toolbar: ['PdfExport'],
      toolbarClick: (args) => {
        if (args.item.id === 'GanttContainer_pdfexport') {
          var exportProperties = {
            fitToWidthSettings: {
              isFitToWidth: true,
            }
          };
          var ganttChart =
document.getElementById('app').ej2_instances[0];
          ganttChart.pdfExport(exportProperties);
        }
      },
    };
  },
  provide: {
    gantt: [Toolbar, PdfExport]
  }
};
</script>

```

{% previewsample "page.domainurl/code-snippet/gantt/single-page" %}

### Exporting with column template

The PDF export functionality allows to export Grid columns that include images, hyperlinks, and custom text to an PDF document using [pdfQueryCellInfo](#) event.

In the following sample, the hyperlinks and images are exported to PDF using [hyperlink](#) and [image](#) properties in the [pdfQueryCellInfo](#) event.

Note: PDF Export supports base64 string to export the images.

### APP.VUE

```

<template>
  <div>

```

```

<ejs-gantt ref='gantt' id="GanttContainer" :dataSource="data"
:taskFields="taskFields" :toolbar="toolbar" :toolbarClick="toolbarClick"
:allowPdfExport='true' :pdfQueryCellInfo="pdfQueryCellInfo" :height="height"
:rowHeight="rowHeight" :splitterSettings="splitterSettings"
:allowResizing='true' :resourceFields="resourceFields"
:resources="resources">
  <e-columns>
    <e-column field='TaskID' headerText='Task ID' textAlign='Left'
width='100'></e-column>
    <e-column field='TaskName' headerText='Task Name' width='150'></e-
column>
    <e-column field='resources' headerText='Resources' width='250'
:template="'cTemplate'"></e-column>
    <e-column field='EmailId' headerText='Email ID' width='150'></e-
column>
  </e-columns>
  <template v-slot:cTemplate="{data}">
    <div class="columnTemplate" v-
if="data.ganttProperties.resourceNames">
      
      <div style="display:inline-
block;width:100%;position:relative;left:30px">
        {{data.ganttProperties.resourceNames}}</div>
      </div>
    </template>
  </ejs-gantt>
</div>
</template>
<script>
import Vue from "vue";
import { GanttPlugin, Toolbar, PdfExport, Selection
,PdfQueryCellInfoEventArgs } from "@syncfusion/ej2-vue-gantt";
import { editingData , editingResources } from './data-source.js';
import { ClickEventArgs } from '@syncfusion/ej2-navigations';
Vue.use(GanttPlugin);
export default {
  data: function() {
    return{
      data: editingData,
      taskFields: {
        id: 'TaskID',
        name: 'TaskName',
        resourceInfo: 'resources',
        startDate: 'StartDate',
        duration: 'Duration',
        progress: 'Progress',
        child: 'subtasks',
      },
      toolbar: ['PdfExport'],
      toolbarClick: (args) => {
        if (args.item.id === 'GanttContainer_pdfexport') {
          var exportProperties: PdfExportProperties = {
            fileName:"new.pdf"

```

```

        };
        var ganttChart =
document.getElementById('GanttContainer').ej2_instances[0];
        ganttChart.pdfExport(exportProperties);
    },
    pdfQueryCellInfo: (args: PdfQueryCellInfoEventArgs) => {
        if (args.column.headerText === 'Resources') {
            {
                args.image = { height: 40, width: 40, base64: (args as
any).data.taskData.resourcesImage };
            }
        }
        if (args.column.headerText === 'Email ID') {
            args.hyperLink = {
                target: 'mailto:' + (args as any).data.taskData.EmailId,
                displayText: (args as any).data.taskData.EmailId
            };
        }
    },
    rowHeight: 50,
    splitterSettings: {
        columnIndex: 3
    },
    height: '450px',
    resourceFields: {
        id: 'resourceId',
        name: 'resourceName',
    },
    resources: editingResources
    },
    provide: {
        gantt: [Toolbar, PdfExport]
    }
};
</script>

```

{% previewsample "page.domainurl/code-snippet/gantt/pdf-export-column-template-cs1" %}

#### *Multiple gantt exporting in Vue Gantt component*

PDF export provides an option for exporting multiple Gantt to same file. In this exported document, each Gantt will be exported to a new page of the document in same file.

#### **APP.VUE**

```

<template>
  <div>
    <p><b>First Gantt:</b></p>
    <ejs-gantt ref='gantt1' id="GanttContainer1" :dataSource="fData"
:taskFields="taskFields" :toolbar="toolbar" :toolbarClick="toolbarClick"
:allowPdfExport='true' :height="height1" :treeColumnIndex="1"
:projectStartDate="projectStartDate" :projectEndDate="projectEndDate"></ejs-
gantt>
    <p><b>Second Gantt:</b></p>

```



```

    <ejs-gantt ref='gantt2' id="GanttContainer2" :dataSource="sData"
    :taskFields="taskFields" :toolbar="toolbar" :toolbarClick="toolbarClick"
    :allowPdfExport='true' :height="height2" :treeColumnIndex="1"></ejs-gantt>
  </div>
</template>
<script>
import Vue from "vue";
import { GanttPlugin, Toolbar, PdfExport, Selection } from "@syncfusion/ej2-
vue-gantt";
import { ClickEventArgs } from '@syncfusion/ej2-navigations';
import { editingData } from './data-source.js';
Vue.use(GanttPlugin);
export default {
  data: function() {
    return{
      fData: [ganttData[0]],
      sData: [ganttData[1]],
      height1: '280px',
      height2: '250px',
      taskFields: {
        id: 'TaskID',
        name: 'TaskName',
        startDate: 'StartDate',
        duration: 'Duration',
        progress: 'Progress',
        child: 'subtasks'
      },
      toolbar: ['PdfExport'],
      projectStartDate: new Date('03/31/2019'),
      projectEndDate: new Date('04/14/2019'),
      toolbarClick: (args) => {
        if (args.item.id === 'GanttContainer1_pdfexport') {
          var firstGantt =
document.getElementById('GanttContainer1').ej2_instances[0];
          var firstGanttExport = firstGantt.pdfExport({}, true);
          firstGanttExport.then((fData) => {
            var secondGantt =
document.getElementById('GanttContainer2').ej2_instances[0];
            secondGantt.pdfExport({}, false, fData);
          });
        }
      },
    };
  },
  provide: {
    gantt: [Toolbar, PdfExport]
  }
};
</script>

```

{% previewsample "page.domainurl/code-snippet/gantt/pdf-multiple-export-cs1" %}

#### [To customize PDF export](#)

PDF export provides an option to customize the mapping of Gantt to exported PDF document.

### File name for exported document

You can assign a file name for the exported document by defining the `fileName` property in `pdfExportProperties`.

### APP.VUE

```
<template>
  <div>
    <ejs-gantt ref='gantt' id="GanttContainer" :dataSource="data"
    :taskFields="taskFields" :toolbar="toolbar" :toolbarClick="toolbarClick"
    :allowPdfExport='true' :height="height"></ejs-gantt>
  </div>
</template>
<script>
import Vue from "vue";
import { GanttPlugin, Toolbar, PdfExport, Selection } from "@syncfusion/ej2-
vue-gantt";
import { ClickEventArgs } from '@syncfusion/ej2-navigations';
import { editingData } from './data-source.js';
Vue.use(GanttPlugin);
export default {
  data: function() {
    return{
      data:editingData,
      height:'450px',
      taskFields: {
        id: 'TaskID',
        name: 'TaskName',
        startDate: 'StartDate',
        duration: 'Duration',
        progress: 'Progress',
        child: 'subtasks'
      },
      toolbar: ['PdfExport'],
      toolbarClick: (args) => {
        if (args.item.id === 'GanttContainer_pdfexport') {
          var exportProperties: PdfExportProperties = {
            fileName:"new.pdf"
          };
          var ganttChart =
document.getElementById('GanttContainer').ej2_instances[0];
          ganttChart.pdfExport(exportProperties);
        }
      },
    };
  },
  provide: {
    gantt: [Toolbar, PdfExport]
  }
};
</script>
```

{% previewsample "page.domainurl/code-snippet/gantt/pdf-export-cs1" %}

### How to change page orientation

Page orientation can be changed to **Portrait** (Default Landscape) for the exported document using the property **pdfExportProperties.pageOrientation**.

### APP.VUE

```
<template>
  <div>
    <ejs-gantt ref='gantt' id="GanttContainer" :dataSource="data"
    :taskFields="taskFields" :toolbar="toolbar" :toolbarClick="toolbarClick"
    :allowPdfExport='true' :height="height"></ejs-gantt>
  </div>
</template>
<script>
import Vue from "vue";
import { GanttPlugin, Toolbar, PdfExport, Selection } from "@syncfusion/ej2-
vue-gantt";
import { ClickEventArgs } from '@syncfusion/ej2-
navigations/src/toolbar/toolbar';
import { editingData } from './data-source.js';
Vue.use(GanttPlugin);
export default {
  data: function() {
    return{
      data: editingData,
      height: '450px',
      taskFields: {
        id: 'TaskID',
        name: 'TaskName',
        startDate: 'StartDate',
        duration: 'Duration',
        progress: 'Progress',
        child: 'subtasks'
      },
      toolbar: ['PdfExport'],
      toolbarClick: (args) => {
        if (args.item.id === 'GanttContainer_pdfexport') {
          var exportProperties: PdfExportProperties = {
            pageOrientation: 'Portrait'
          };
          var ganttChart =
document.getElementById('GanttContainer').ej2_instances[0];
          ganttChart.pdfExport(exportProperties);
        }
      },
    };
  },
  provide: {
    gantt: [Toolbar, PdfExport]
  }
};
</script>
```

{% previewsample "page.domainurl/code-snippet/gantt/pdf-export-cs2" %}

### How to change page size

Page size can be customized for the exported document using the property `pdfExportProperties.pageSize`.

The supported page sizes are:

- Letter
- Note
- Legal
- A0
- A1
- A2
- A3
- A5
- A6
- A7
- A8
- A9
- B0
- B1
- B2
- B3
- B4
- B5
- Archa
- Archb
- Archc
- Archd
- Arche
- Flsa
- HalfLetter
- Letter11x17
- Ledger

### APP.VUE

```
<template>
  <div>
    <ejs-gantt ref='gantt' id="GanttContainer" :dataSource="data"
:taskFields="taskFields" :toolbar="toolbar" :toolbarClick="toolbarClick"
:allowPdfExport='true' :height="height"></ejs-gantt>
  </div>
</template>
<script>
import Vue from "vue";
import { GanttPlugin, Toolbar, PdfExport, Selection, PdfExportProperties }
from "@syncfusion/ej2-vue-gantt";
import { ClickEventArgs } from '@syncfusion/ej2-
navigations/src/toolbar/toolbar';
import { editingData } from './data-source.js';
Vue.use(GanttPlugin);
```

```

export default {
  data: function() {
    return{
      data:editingData,
      height:'450px',
      taskFields: {
        id: 'TaskID',
        name: 'TaskName',
        startDate: 'StartDate',
        duration: 'Duration',
        progress: 'Progress',
        child: 'subtasks'
      },
      toolbar: ['PdfExport'],
      toolbarClick: (args) => {
        if (args.item.id === 'GanttContainer_pdfexport') {
          var exportProperties: PdfExportProperties = {
            pageSize: 'A0'
          };
          var ganttChart =
document.getElementById('GanttContainer').ej2_instances[0];
          ganttChart.pdfExport(exportProperties);
        }
      },
    };
  },
  provide: {
    gantt: [Toolbar, PdfExport]
  }
};
</script>

```

{% previewsample "page.domainurl/code-snippet/gantt/pdf-export-cs3" %}

[Export current view data](#)

PDF export provides an option to export the current view data into PDF. To export current view data alone, define the `exportType` to `CurrentViewData`.

### APP.VUE

```

<template>
  <div>
    <ejs-gantt ref='gantt' id="GanttContainer" :dataSource="data"
:taskFields="taskFields" :toolbar="toolbar" :toolbarClick="toolbarClick"
:allowPdfExport='true' :height="height"></ejs-gantt>
  </div>
</template>
<script>
import Vue from "vue";
import { GanttPlugin, Toolbar, PdfExport, Selection, Filter,
PdfExportProperties } from "@syncfusion/ej2-vue-gantt";
import { ClickEventArgs } from '@syncfusion/ej2-
navigations/src/toolbar/toolbar';
import { editingData } from './data-source.js';
Vue.use(GanttPlugin);
export default {

```

```

data: function() {
  return{
    data:editingData,
    height:'450px',
    taskFields: {
      id: 'TaskID',
      name: 'TaskName',
      startDate: 'StartDate',
      duration: 'Duration',
      progress: 'Progress',
      child: 'subtasks'
    },
    toolbar: ['PdfExport'],
    toolbarClick: (args) => {
      if (args.item.id === 'GanttContainer_pdfexport') {
        var exportProperties: PdfExportProperties = {
          exportType: 'CurrentViewData'
        };
        var ganttChart =
document.getElementById('GanttContainer').ej2_instances[0];
        ganttChart.pdfExport(exportProperties);
      }
    },
  };
},
provide: {
  gantt: [Toolbar, PdfExport]
}
};
</script>

```

{% previewsample "page.domainurl/code-snippet/gantt/pdf-export-cs4" %}

#### Enable footer

By default, we render the default footer for a PDF file, this can be enabled or disabled by using the `enableFooter` property.

#### APP.VUE

```

<template>
  <div>
    <ejs-gantt ref='gantt' id="GanttContainer" :dataSource="data"
:taskFields="taskFields" :toolbar="toolbar" :toolbarClick="toolbarClick"
:allowPdfExport='true' :height="height"></ejs-gantt>
  </div>
</template>
<script>
import Vue from "vue";
import { GanttPlugin, Toolbar, PdfExport, Selection, PdfExportProperties }
from "@syncfusion/ej2-vue-gantt";
import { ClickEventArgs } from '@syncfusion/ej2-
navigations/src/toolbar/toolbar';
import { editingData } from './data-source.js';
Vue.use(GanttPlugin);
export default {
  data: function() {

```

```

return{
  data:editingData,
  height:'450px',
  taskFields: {
    id: 'TaskID',
    name: 'TaskName',
    startDate: 'StartDate',
    duration: 'Duration',
    progress: 'Progress',
    child: 'subtasks'
  },
  toolbar: ['PdfExport'],
  toolbarClick: (args) => {
    if (args.item.id === 'GanttContainer_pdfexport') {
      var exportProperties: PdfExportProperties = {
        enableFooter: false
      };
      var ganttChart =
document.getElementById('GanttContainer').ej2_instances[0];
      ganttChart.pdfExport(exportProperties);
    }
  },
};
},
provide: {
  gantt: [Toolbar, PdfExport]
}
};
</script>

```

{% previewsample "page.domainurl/code-snippet/gantt/pdf-export-cs5" %}

#### [Export hidden columns](#)

PDF export provides an option to export hidden columns of Gantt by defining the `includeHiddenColumn` to `true`.

#### APP.VUE

```

<template>
  <div>
    <ejs-gantt ref='gantt' id="GanttContainer" :dataSource="data"
:taskFields="taskFields" :columns="columns" :toolbar="toolbar"
:toolbarClick="toolbarClick" :allowPdfExport='true' :height="height"></ejs-
gantt>
  </div>
</template>
<script>
import Vue from "vue";
import { Gantt, Toolbar, PdfExport, Selection, PdfExportProperties } from
"@syncfusion/ej2-vue-gantt";
import { ClickEventArgs } from '@syncfusion/ej2-
navigations/src/toolbar/toolbar';
import { ganttData } from './data-source.js';
Vue.use(GanttPlugin);
export default {
  data: function() {

```

```

return{
  data: ganttData,
  height: '450px',
  taskFields: {
    id: 'TaskID',
    name: 'TaskName',
    startDate: 'StartDate',
    duration: 'Duration',
    progress: 'Progress',
    child: 'subtasks'
  },
  columns: [
    { field: 'TaskID', headerText: 'Task ID', textAlign: 'Left',
width: '100' },
    { field: 'TaskName', headerText: 'Task Name', width: '150',
visible: false },
    { field: 'StartDate', headerText: 'Start Date', width: '150' },
    { field: 'Duration', headerText: 'Duration', width: '150' },
    { field: 'Progress', headerText: 'Progress', width: '150' },
  ],
  toolbar: ['PdfExport'],
  toolbarClick: (args) => {
    if (args.item.id === 'GanttContainer_pdfexport') {
      var exportProperties: PdfExportProperties = {
        includeHiddenColumn: true
      };
      var ganttChart =
document.getElementById('GanttContainer').ej2_instances[0];
      ganttChart.pdfExport(exportProperties);
    }
  },
};
},
provide: {
  gantt: [Toolbar, PdfExport]
}
};
</script>

```

{% previewsample "page.domainurl/code-snippet/gantt/pdf-export-cs6" %}

#### Export predecessor lines

By using `showPredecessorLines`, you can hide or show predecessor lines in the exported PDF document.

#### APP.VUE

```

<template>
  <div>
    <ejs-gantt ref='gantt' id="GanttContainer" :dataSource="data"
:taskFields="taskFields" :columns="columns" :toolbar="toolbar"
:toolbarClick="toolbarClick" :allowPdfExport='true' :height="height"></ejs-
gantt>
  </div>
</template>
<script>

```



```

import Vue from "vue";
import { GanttPlugin, Toolbar, PdfExport, Selection, PdfExportProperties }
from "@syncfusion/ej2-vue-gantt";
import { ClickEventArgs } from '@syncfusion/ej2-
navigations/src/toolbar/toolbar';
import { editingData } from './data-source.js';
Vue.use(GanttPlugin);
export default {
  data: function() {
    return{
      data: editingData,
      height: '450px',
      taskFields: {
        id: 'TaskID',
        name: 'TaskName',
        startDate: 'StartDate',
        duration: 'Duration',
        progress: 'Progress',
        child: 'subtasks'
      },
      columns: [
        { field: 'TaskID', headerText: 'Task ID', textAlign: 'Left',
width: '100' },
        { field: 'TaskName', headerText: 'Task Name', width: '150' },
        { field: 'StartDate', headerText: 'Start Date', width: '150',
visible: false },
        { field: 'Duration', headerText: 'Duration', width: '150' },
        { field: 'Progress', headerText: 'Progress', width: '150' },
      ],
      toolbar: ['PdfExport'],
      toolbarClick: (args) => {
        if (args.item.id === 'GanttContainer_pdfexport') {
          var exportProperties: PdfExportProperties = {
            showPredecessorLines: true
          };
          var ganttChart =
document.getElementById('GanttContainer').ej2_instances[0];
          ganttChart.pdfExport(exportProperties);
        }
      },
    };
  },
  provide: {
    gantt: [Toolbar, PdfExport]
  }
};
</script>

```

{% previewsample "page.domainurl/code-snippet/gantt/pdf-export-cs7" %}

[Show or hide columns on exported PDF](#)

You can show a hidden column or hide a visible column while exporting the Gantt using the [toolbarClick](#) and [beforePdfExport](#) events.

You can show or hide columns by setting the `column.visible` property to `true` or `false` respectively.

In the following example, there is a hidden column **Duration** in the Gantt. While exporting, we have changed **Duration** to visible column and **StartDate** to hidden column.

### APP.VUE

```
<template>
  <div>
    <ejs-gantt ref='gantt' id="GanttContainer" :dataSource="data"
    :taskFields="taskFields" :columns="columns" :toolbar="toolbar"
    :toolbarClick="toolbarClick" :allowPdfExport='true'
    :beforePdfExport="beforePdfExport" :height="height"></ejs-gantt>
  </div>
</template>
<script>
import Vue from "vue";
import { GanttPlugin, Toolbar, PdfExport, Selection } from "@syncfusion/ej2-
vue-gantt";
import { ClickEventArgs } from '@syncfusion/ej2-
navigations/src/toolbar/toolbar';
import { editingData } from './data-source.js';
Vue.use(GanttPlugin);
export default {
  data: function() {
    return{
      data: editingData,
      height: '450px',
      taskFields: {
        id: 'TaskID',
        name: 'TaskName',
        startDate: 'StartDate',
        duration: 'Duration',
        progress: 'Progress',
        child: 'subtasks'
      },
      columns: [
        { field: 'TaskID', headerText: 'Task ID', textAlign: 'Left',
width: '100' },
        { field: 'TaskName', headerText: 'Task Name', width: '150' },
        { field: 'StartDate', headerText: 'Start Date', width: '150' },
        { field: 'Duration', headerText: 'Duration', width: '150',
visible: false },
        { field: 'Progress', headerText: 'Progress', width: '150' },
      ],
      toolbar: ['PdfExport'],
      toolbarClick: (args) => {
        if (args.item.id === 'GanttContainer_pdfexport') {
          var ganttChart =
document.getElementById('GanttContainer').ej2_instances[0];
          ganttChart.pdfExport();
        }
      },
      beforePdfExport : (args) => {
        var obj =
(document.getElementById('GanttContainer')).ej2_instances[0]
        obj.treeGrid.columns[3].visible = true;
        obj.treeGrid.columns[2].visible = false;
      }
    }
  }
}
```

```

    };
  },
  provide: {
    gantt: [Toolbar, PdfExport]
  }
};
</script>

```

{% previewsample "page.domainurl/code-snippet/gantt/pdf-export-cs8" %}

### Conditional cell formatting

TreeGrid cells in the exported PDF can be customized or formatted using the [pdfQueryCellInfo](#) event. In this event, you can format the treegrid cells of exported PDF document based on the column cell value.

In the following sample, the background color is set for **Progress** column in the exported document by using the **args.style** and **backgroundColor** properties.

### APP.VUE

```

<template>
  <div>
    <ejs-gantt ref='gantt' id="GanttContainer" :dataSource="data"
    :taskFields="taskFields" :columns="columns" :toolbar="toolbar"
    :toolbarClick="toolbarClick" :allowPdfExport='true'
    :pdfQueryCellInfo="pdfQueryCellInfo" :height="height"></ejs-gantt>
  </div>
</template>
<script>
import Vue from "vue";
import { GanttPlugin, Toolbar, PdfExport, Selection,
PdfQueryTimelineCellInfoEventArgs } from "@syncfusion/ej2-vue-gantt";
import { ClickEventArgs } from '@syncfusion/ej2-
navigations/src/toolbar/toolbar';
import { editingData } from './data-source.js';
Vue.use(GanttPlugin);
export default {
  data: function() {
    return{
      data:editingData,
      height:'450px',
      taskFields: {
        id: 'TaskID',
        name: 'TaskName',
        startDate: 'StartDate',
        duration: 'Duration',
        progress: 'Progress',
        child: 'subtasks'
      },
      columns: [
        { field: 'TaskID', headerText: 'Task ID', textAlign: 'Left',
width: '100' },
        { field: 'TaskName', headerText: 'Task Name', width: '150' },
        { field: 'StartDate', headerText: 'Start Date', width: '150' },
        { field: 'Duration', headerText: 'Duration', width: '150',
visible: false },
        { field: 'Progress', headerText: 'Progress', width: '150' },

```

```

    ],
    toolbar: ['PdfExport'],
    toolbarClick: (args) => {
        if (args.item.id === 'GanttContainer_pdfexport') {
            var ganttChart =
document.getElementById('GanttContainer').ej2_instances[0];
            ganttChart.pdfExport();
        }
    },
    pdfQueryCellInfo: (args) => {
        if (args.column.field === 'Progress') {
            if (args.value < 50) {
                args.style = {backgroundColor: '#F08080'};
            } else {
                args.style = {backgroundColor: '#A569BD'};
            }
        }
    }
};

},
provide: {
    gantt: [Toolbar, PdfExport]
}
};
</script>

```

{% previewsample "page.domainurl/code-snippet/gantt/pdf-export-cs9" %}

#### Timeline cell formatting

Timeline cells in the exported PDF document can be customized or formatted using the [pdfQueryTimelineCellInfo](#) event.

In the following sample, the header background color is set for timeline cells in the exported document by using the `args.headerBackgroundColor` property.

#### APP.VUE

```

<template>
  <div>
    <ejs-gantt ref='gantt' id="GanttContainer" :dataSource="data"
:taskFields="taskFields"
:columns="columns" :toolbar="toolbar" :toolbarClick="toolbarClick"
:allowPdfExport='true' :pdfQueryTimelineCellInfo="pdfQueryTimelineCellInfo"
:height="height"></ejs-gantt>
  </div>
</template>
<script>
import Vue from "vue";
import { GanttPlugin, Toolbar, PdfExport, Selection,
PdfQueryTimelineCellInfoEventArgs } from "@syncfusion/ej2-vue-gantt";
import { PdfColor } from '@syncfusion/ej2-pdf-export';
import { ClickEventArgs } from '@syncfusion/ej2-
navigations/src/toolbar/toolbar';
import { editingData } from './data-source.js';
Vue.use(GanttPlugin);
export default {

```

```

data: function() {
  return{
    data:editingData,
    height:'450px',
    taskFields: {
      id: 'TaskID',
      name: 'TaskName',
      startDate: 'StartDate',
      duration: 'Duration',
      progress: 'Progress',
      child: 'subtasks'
    },
    columns: [
      { field: 'TaskID', headerText: 'Task ID', textAlign: 'Left',
width: '100' },
      { field: 'TaskName', headerText: 'Task Name', width: '150' },
      { field: 'StartDate', headerText: 'Start Date', width: '150' },
      { field: 'Duration', headerText: 'Duration', width: '150',
visible: false },
      { field: 'Progress', headerText: 'Progress', width: '150' },
    ],
    toolbar: ['PdfExport'],
    toolbarClick: (args) => {
      if (args.item.id === 'GanttContainer_pdfexport') {
        var ganttChart =
document.getElementById('GanttContainer').ej2_instances[0];
        ganttChart.treeGrid.grid.columns[2].visible = false;
        ganttChart.pdfExport();
      }
    },
    pdfQueryTimelineCellInfo: (args) => {
      args.timelineCell.backgroundColor = new PdfColor(240, 248, 255);
    };
  };
},
provide: {
  gantt: [Toolbar, PdfExport]
}
};
</script>

```

{% previewsample "page.domainurl/code-snippet/gantt/pdf-export-cs10" %}

#### Taskbar formatting

Taskbars in the exported PDF document can be customized or formatted using the [pdfQueryTaskbarInfo](#) event.

In the following sample, the taskbar background color is customized in the chart side of the exported document by using the `args.taskbar` property.

#### APP.VUE

```

<template>
  <div>
    <ejs-gantt ref='gantt' id="GanttContainer" :dataSource="data"
:taskFields="taskFields"

```

```

:columns="columns" :toolbar="toolbar" :toolbarClick="toolbarClick"
:allowPdfExport='true' :pdfQueryTaskbarInfo="pdfQueryTaskbarInfo"
:height="height"></ejs-gantt>
</div>
</template>
<script>
import Vue from "vue";
import { GanttPlugin, Toolbar, PdfExport, Selection } from "@syncfusion/ej2-
vue-gantt";
import { PdfColor } from '@syncfusion/ej2-pdf-export';
import { ClickEventArgs } from '@syncfusion/ej2-
navigations/src/toolbar/toolbar';
import { editingData } from './data-source.js';
Vue.use(GanttPlugin);
export default {
  data: function() {
    return{
      data:editingData,
      height:'450px',
      taskFields: {
        id: 'TaskID',
        name: 'TaskName',
        startDate: 'StartDate',
        duration: 'Duration',
        progress: 'Progress',
        child: 'subtasks'
      },
      columns: [
        { field: 'TaskID', headerText: 'Task ID', textAlign: 'Left',
width: '100' },
        { field: 'TaskName', headerText: 'Task Name', width: '150',
visible: false },
        { field: 'StartDate', headerText: 'Start Date', width: '150' },
        { field: 'Duration', headerText: 'Duration', width: '150' },
        { field: 'Progress', headerText: 'Progress', width: '150' },
      ],
      toolbar: ['PdfExport'],
      toolbarClick: (args) => {
        if (args.item.id === 'GanttContainer_pdfexport') {
          var ganttChart =
document.getElementById('GanttContainer').ej2_instances[0];
          ganttChart.treeGrid.grid.columns[2].visible = false;
          ganttChart.pdfExport();
        }
      },
      pdfQueryTaskbarInfo: (args) => {
        if (args.data.Progress < 50 && !args.data.hasChildRecords) {
          args.taskbar.progressColor = new PdfColor(205, 92, 92);
          args.taskbar.taskColor = args.taskbar.taskBorderColor = new
PdfColor(240, 128, 128);
        }
      }
    };
  },
  provide: {
    gantt: [Toolbar, PdfExport]
  }
}

```

```
};
</script>
```

```
{% previewsample "page.domainurl/code-snippet/gantt/pdf-export-cs11" %}
```

### Customized Theme

PDF export provides an option to customize the Gantt style in the exported PDF documents.

To customize the Gantt style in exported PDF, define the 'ganttStyle' in `pdfExportProperties`.

### APP.VUE

```
<template>
  <div>
    <ejs-gantt ref='gantt' id="GanttContainer" :dataSource="data"
    :taskFields="taskFields" :toolbar="toolbar" :toolbarClick="toolbarClick"
    :allowPdfExport='true' :height="height"></ejs-gantt>
  </div>
</template>
<script>
import Vue from "vue";
import { GanttPlugin, Toolbar, PdfExport, Selection, PdfExportProperties }
from "@syncfusion/ej2-vue-gantt";
import { PdfColor, PdfPaddings } from '@syncfusion/ej2-pdf-export';
import { editingData } from './data-source.js';
Vue.use(GanttPlugin);
export default {
  data: function() {
    return{
      data: editingData,
      height: '450px',
      taskFields: {
        id: 'TaskID',
        name: 'TaskName',
        startDate: 'StartDate',
        duration: 'Duration',
        progress: 'Progress',
        child: 'subtasks'
      },
      toolbar: ['PdfExport'],
      toolbarClick: (args) => {
        if (args.item.id === 'GanttContainer_pdfexport') {
          let exportProperties = {
            fontFamily: 1,
            columnHeader: {
              backgroundColor: new PdfColor(179, 219, 255)
            },
            taskbar: {
              taskColor: new PdfColor(240, 128, 128),
              taskBorderColor: new PdfColor(240, 128, 128),
              progressColor: new PdfColor(205, 92, 92),
            },
            connectorLineColor: new PdfColor(128, 0, 0),
            footer: {
              backgroundColor: new PdfColor(205, 92, 92)
            },
            timeline: {
```

```

        backgroundColor: new PdfColor(179, 219, 255),
        padding: new PdfPaddings(5, 2, 0, 0),
    },
    label: {
        fontColor: new PdfColor(128, 0, 0),
    },
    cell: {
        backgroundColor: new PdfColor(240, 248, 255),
        fontColor: new PdfColor(0, 0, 0),
        borderColor: new PdfColor(179, 219, 255),
    },
};

var ganttChart =
document.getElementById('GanttContainer').ej2_instances[0];
ganttChart.pdfExport(exportProperties);
    }
}

};
},
provide: {
    gantt: [Toolbar, PdfExport]
}
};
</script>

```

{% previewsample "page.domainurl/code-snippet/gantt/pdf-export-cs12" %}

*Customizing header and footer of PDF export in ##Platform\_Name## Gantt control*

PDF export provides an option to specify and customize text, page number, line and image in header and footer of exported PDF document by using [pdfExportProperties](#).

[Write a text in header and footer](#)

This functionality helps to customize the text that appears in the header or footer sections of a PDF document. Text can be added to [header](#) or [footer](#) of exported PDF document by using [pdfExportProperties](#).

- **type** property in the content array indicates the content type, such as 'Text'.
- **Value** property determines the text.
- **Position** property determines the horizontal and vertical positions of the text element.
- **style** property define the visual styling properties for the text element

`ts

```

let exportProperties: PdfExportProperties = {
header: {
fromTop: 0,
height: 130,
contents: [
{
type: 'Text',

```



```

value: 'INVOICE',
position: { x: 380, y: 0 },
style: { textBrushColor: '#C25050', fontSize: 25 },
},
]
}
,

```

#### Draw a line in header and footer

This functionality helps to customize the line that appears in the header or footer sections of the PDF document. A line can be added to [header](#) or [footer](#) of the exported PDF document by using [pdfExportProperties](#).

- **type** determines content type, such as 'Line'.
- **style** is used to set properties like the color (penColor), size (penSize), and style (dashStyle) of the line.
- **points** specifies the coordinates for the start and end points of the line.

Supported line styles:

- dash
- dot
- dashdot
- dashdotdot
- solid

```

`ts
let exportProperties: PdfExportProperties = {
header: {
fromTop: 0,
height: 130,
contents: [
{
type: 'Line',
style: { penColor: '#000080', penSize: 2, dashStyle: 'Solid' },
points: { x1: 0, y1: 4, x2: 685, y2: 4 }
}
]
}

```

```
}
,
```

### Add page number in header and footer

This feature allows to customize the page number that appears in the header or footer sections of the PDF document. Page numbers can be added in [header](#) or [footer](#) of the exported PDF document by using [pdfExportProperties](#).

- `type` indicates that the content is a page number.
- `pageNumberType` specifies the type of numbering to be used.
- `format` is an optional attribute that allows you to customize the text format of the page number.
- `position` defines the coordinates (x, y) where the page number will be located.
- `style` sets the styling properties of the page number text, such as color (`textBrushColor`), font size (`fontSize`), and horizontal alignment (`hAlign`).

Supported page number types:

- LowerLatin - a, b, c,
- UpperLatin - A, B, C,
- LowerRoman - i, ii, iii,
- UpperRoman - I, II, III,
- Number - 1,2,3.

```
`ts
```

```
let exportProperties: PdfExportProperties = {
  header: {
    fromTop: 0,
    height: 130,
    contents: [
      {
        type: 'PageNumber',
        pageNumberType: 'Arabic',
        format: 'Page ${current} of ${total}', //optional
        position: { x: 0, y: 25 },
        style: { textBrushColor: '#ffff80', fontSize: 15, hAlign: 'Center' }
      }
    ]
  }
}
```

### Insert an image in header and footer

This feature allows to customize the image that appears in the header or footer sections of the PDF document. Image (Base64 string) can be added in the exported document in [header](#) or [footer](#) of the exported PDF document by using [pdfExportProperties](#).

- `type` indicates that the content is an image.
- `src` specifies the source of the image, which should be Base64 string.
- `Position` determines the horizontal and vertical positions of the image will be located.
- `size` sets the dimensions of the image.

Note: PDF Export supports base64 string to export the images.

```
`ts
// Replace it with a valid Base64-encoded image.
let image: string = "/9j/4AAQSkZJRgABAQEAAeAB4AAD..."
let exportProperties: PdfExportProperties = {
  header: {
    fromTop: 0,
    height: 130,
    contents: [
      {
        type: 'Image',
        src: image,
        position: { x: 40, y: 10 },
        size: { height: 100, width: 250 },
      }
    ]
  }
}
```

The below code illustrates the pdf export customization.

### APP.VUE

```
<template>
  <div>
    <ejs-gantt ref='gantt' id="GanttContainer" :dataSource="data"
    :taskFields="taskFields" :toolbar="toolbar"
    :toolbarClick="toolbarClick" :allowPdfExport='true'
    :height="height"></ejs-gantt>
```

```

</div>
</template>
<script>
import Vue from "vue";
import { GanttPlugin, Toolbar, PdfExport, Selection, PdfExportProperties }
from "@syncfusion/ej2-vue-gantt";
import { ClickEventArgs } from '@syncfusion/ej2-
navigations/src/toolbar/toolbar';
import { editingData,image } from './data-source.js';
Vue.use(GanttPlugin);
export default {
  data: function () {
    return {
      data: editingData,
      height: '450px',
      taskFields: {
        id: 'TaskID',
        name: 'TaskName',
        startDate: 'StartDate',
        duration: 'Duration',
        progress: 'Progress',
        child: 'subtasks'
      },
      toolbar: ['PdfExport'],
      toolbarClick: (args) => {
        if (args.item.id === 'GanttContainer_pdfexport') {
          var exportProperties: PdfExportProperties = {
            header: {
              fromTop: 0,
              height: 150,
              contents: [
                {
                  type: 'Text',
                  value: 'Welcome',
                  position: { x: 380, y: 0 },
                  style: { textBrushColor: '#C25050',
fontSize: 25 },
                },
                {
                  type: 'Image',
                  src: image,
                  position: { x: 400, y: 70 },
                  size: { height: 50, width: 50 },
                },
              ]
            },
            footer: {
              fromBottom: 160,
              height: 100,
              contents: [
                {
                  type: 'Text',
                  value: 'Thank you!',
                  position: { x: 350, y: 40 },
                  style: { textBrushColor: '#C67878',
fontSize: 14 }
                }
              ]
            }
          }
        }
      }
    }
  }
}

```

```

        {
            type: 'PageNumber',
            pageNumberType: 'Arabic',
            format: 'Page {$current} of {$total}',
            position: { x: 0, y: 25 },
            size: { height: 50, width: 100 },
            style: { textBrushColor: '#000000',
hAlign: 'Center', vAlign: 'Bottom' }
        }
    ],
    },
    };
    var ganttChart =
document.getElementById('GanttContainer').ej2_instances[0];
    ganttChart.pdfExport(exportProperties);
    },
    },
    },
    provide: {
        gantt: [Toolbar, PdfExport]
    }
    };
</script>

```

{% previewsample "page.domainurl/code-snippet/gantt/pdf-export-customization-cs1" %}

### Context menu in Vue Gantt component

The Gantt control allows you to perform quick actions by using context menu. When right-clicking the context menu, the context menu options are shown. To enable this feature, set the [enableContextMenu](#) to true. The default context menu options are enabled using the [editSettings](#) property. The context menu options can be customized using the [contextMenuItems](#) property.

To use the context menu, inject the [ContextMenu](#) module in the `provide` section.

The default items are listed in the following table.

Items	Description
AutoFit	Auto-fits the current column.
AutoFitAll	Auto-fits all columns.
SortAscending	Sorts the current column in ascending order.
SortDescending	Sorts the current column in descending order.
TaskInformation	Edits the current task.
Add	Adds a new row to the Gantt.
Indent	Indent the selected record to one level.
Outdent	Outdent the selected record to one level.
DeleteTask	Deletes the current task.
Save	Saves the edited task.

**Cancel** | Cancels the edited task.

**DeleteDependency** | Deletes the current dependency task link.

**Convert** | Converts current task to milestone or vice-versa.

### APP.VUE

```
<template>
  <div>
    <ejs-gantt ref='gantt' id="contextMenu" :dataSource="data"
      :taskFields = "taskFields" :height = "height" :editSettings="editSettings"
      :enableContextMenu="true" :allowSorting="true" :allowResizing= "true"></ejs-
gantt>
  </div>
</template>
<script>
import Vue from "vue";
import { GanttPlugin, ContextMenu, Edit, Sort, Resize, Selection } from
"@syncfusion/ej2-vue-gantt";
import { editingData } from './data-source.js';
Vue.use(GanttPlugin);
export default {
  data: function() {
    return{
      data: editingData,
      height: '450px',
      taskFields: {
        id: 'TaskID',
        name: 'TaskName',
        startDate: 'StartDate',
        duration: 'Duration',
        progress: 'Progress',
        dependency: 'Predecessor',
        child: 'subtasks'
      },
      editSettings: {
        allowAdding: true,
        allowEditing: true,
        allowDeleting: true
      }
    };
  },
  provide: {
    gantt: [ ContextMenu, Edit, Sort, Resize, Selection]
  }
};
</script>
```

{% previewsample "page.domainurl/code-snippet/gantt/contextmenu-cs1" %}

### Custom context menu items

The custom context menu items can be added by defining the [contextMenuItems](#) as a collection of [contextMenuItemModel](#). Actions for the customized items can be defined in the [contextMenuClick](#) event and the visibility of customized items can be defined in the [contextMenuOpen](#) event.

To create custom context menu items for header area, define the target property as `.e-gridheader`.

The following sample shows context menu item for parent rows to expand or collapse child rows in the content area and a context menu item to hide columns in the header area.

#### APP.VUE

```
<template>
  <div>
    <ejs-gantt ref='gantt' id="customContextMenu" :dataSource="data"
    :taskFields = "taskFields" :height = "height" :editSettings="editSettings"
    :enableContextMenu="true" :allowSorting="true" :allowResizing= "true"
    :contextMenuItems="contextMenuItems" :contextMenuClick = "contextMenuClick"
    :contextMenuOpen="contextMenuOpen"></ejs-gantt>
  </div>
</template>
<script>
import Vue from "vue";
import { GanttPlugin, ContextMenu, Edit, Sort, Resize, Selection } from
"@syncfusion/ej2-vue-gantt";
import { editingData } from './data-source.js';
Vue.use(GanttPlugin);
export default {
  data: function() {
    return{
      data: editingData,
      height: '450px',
      taskFields: {
        id: 'TaskID',
        name: 'TaskName',
        startDate: 'StartDate',
        duration: 'Duration',
        progress: 'Progress',
        dependency: 'Predecessor',
        child: 'subtasks'
      },
      editSettings: {
        allowAdding: true,
        allowEditing: true,
        allowDeleting: true
      },
      contextMenuItems: ['AutoFitAll', 'AutoFit', 'TaskInformation',
        'DeleteTask', 'Save', 'Cancel', 'SortAscending', 'SortDescending', 'Add',
        'DeleteDependency', 'Convert',
        { text: 'Collapse the Row', target: '.e-content', id: 'collapserow'
      },
        { text: 'Expand the Row', target: '.e-content', id: 'expandrow' },
        { text: 'Hide Column', target: '.e-gridheader', id: 'hidecols' }
      ],
    };
  },
  provide: {
    gantt: [ ContextMenu, Edit, Sort, Resize, Selection]
  },
  methods: {
    contextMenuClick: function (args) {
      var record = args.rowData;
```

```

var ganttObj =
document.getElementById('customContextMenu').ej2_instances[0];
if (args.item.id === 'collapserow') {
    ganttObj.collapseByID(Number(record.ganttProperties.taskId));
}
if (args.item.id === 'expandrow') {
    ganttObj.expandByID(Number(record.ganttProperties.taskId));
}
if (args.item.id === 'hidecols') {
    ganttObj.hideColumn(args.column.headerText);
}
},
contextMenuOpen: function (args) {
var record = args.rowData;
if (args.type !== 'Header') {
    if (!record.hasChildRecords) {
        args.hideItems.push('Collapse the Row');
        args.hideItems.push('Expand the Row');
    } else {
        if (record.expanded) {
            args.hideItems.push("Expand the Row");
        } else {
            args.hideItems.push("Collapse the Row");
        }
    }
}
}
}
};
</script>

```

{% previewsample "page.domainurl/code-snippet/gantt/customContextMenu-cs1" %}

You can show an specific item in context menu for header/content area in the Gantt control by defining the **target** property.

### Touch interaction

To perform **long press** action on a row, [context menu](#) is opened, and then tap a menu item to trigger its action.

### State persistence in Vue Gantt component

State persistence refers to the Gantt's state maintained in the browser's [localStorage](#) even if the browser is refreshed or if you move to the next page within the browser.

State persistence stores gantt's model object in the local storage when the [enablePersistence](#) is defined as true.

### Get or set localStorage value

If the [enablePersistence](#) property is set to true, the gantt property value is saved in the `window.localStorage` for reference. You can get/set the localStorage value by using the `getItem/setItem` method in the `window.localStorage`.

```
`ts
```

```
//get the Gantt model.
```



```
let value: string = window.localStorage.getItem('ganttGantt'); //"ganttGantt" is component name +
component id.
```

```
let model: Object = JSON.parse(model);
```

```
,
```

```
`ts
```

```
//set the Gantt model.
```

```
window.localStorage.setItem('ganttGantt', JSON.stringify(model)); //"ganttGantt" is component name +
component id.
```

```
,
```

You can refer to our [Vue Gantt](#) feature tour page for its groundbreaking feature representations. You can also explore our [Vue Gantt example](#) to know how to present and manipulate data.

### Prevent columns from persisting

When the [enablePersistence](#) property is set to true, the Gantt properties such as [Filtering](#), [Sorting](#), and [Columns](#) will persist. You can use the `addOnPersist` method to prevent these Gantt properties from persisting.

The following example demonstrates how to prevent Gantt columns from persisting. In the [dataBound](#) event of the Gantt, you can override the `addOnPersist` method and remove the columns from the key list given for persistence.

Note: When the `enablePersistence` property is set to true, the Gantt features such as column template, column formatter, header text, and value accessor will not persist.

### APP.VUE

```
<template>
  <div id="app">
    <button id="add" @click="clickAdd">Add Columns</button>
    <button id="remove" @click="clickRemove">Remove Columns</button>
    <br /><br />
    <ejs-gantt ref="gantt" :dataSource='data' :enablePersistence='true'
height='230px' id="Gantt" :taskFields = "taskFields" :dataBound='dataBound'>
      <e-columns>
        <e-column field='TaskID' headerText='Task ID'
textAlign='Right' width=120></e-column>
        <e-column field='TaskName' headerText='Task Name'
width=150></e-column>
        <e-column field='StartDate' headerText='Start Date'
width=150 ></e-column>
        <e-column field='Duration' headerText='Duration' width=150
></e-column>
      </e-columns>
    </ejs-gantt>
  </div>
</template>
<script>
import Vue from "vue";
import { GanttPlugin } from "@syncfusion/ej2-vue-gantt";
import { projectNewData } from './data-source.js';
Vue.use(GanttPlugin);
```

```

export default {
  data() {
    return {
      data: projectNewData,
      taskFields: {
        id: 'TaskID',
        name: 'TaskName',
        startDate: 'StartDate',
        duration: 'Duration',
        progress: 'Progress',
        child: 'subtasks'
      }
    };
  },
  methods: {
    dataBound: function () {
      let cloned = this.$refs.gantt.ej2Instances.addOnPersist;
      this.$refs.gantt.ej2Instances.addOnPersist = function (key: any) {
        key = key.filter((item) => item !== "columns");
        return cloned.call(this, key);
      };
    },
    clickAdd: function () {
      let obj = { field: "Progress", headerText: 'Progress', width: 100 };
      this.$refs.gantt.ej2Instances.columns.push(obj); //you can add the columns by using the Gantt columns method
      this.$refs.gantt.ej2Instances.treeGrid.refreshColumns();
    },
    clickRemove: function () {
      this.$refs.gantt.ej2Instances.columns.pop();
      this.$refs.gantt.ej2Instances.treeGrid.refreshColumns();
    }
  }
}
</script>
<style>
</style>

```

{% previewsample "page.domainurl/code-snippet/gantt/column-prevent-cs1" %}

### Persist the header template and header Text

By default, the Gantt properties such as column template, header text, header template, column formatter, and value accessor will not persist when [enablePersistence](#) is set to true. Because the column template and header text can be customized at the application level.

If you wish to restore all these column properties, then you can achieve it by cloning the gantt columns property using JavaScript Object's assign method and storing this along with the persist data manually. While restoring the settings, this column object must be assigned to the gantt's columns property to restore the column settings as demonstrated in the following sample.

### APP.VUE

```

<template>
  <div id="app">
    <button id="restore" @click="clickRestore">Restore</button>
    <br /><br />
  </div>
</template>

```

```

    <ejs-gantt ref="gantt" :dataSource='data' :enablePersistence='true'
height='230px' id="Gantt" :taskFields = "taskFields">
    <e-columns>
        <e-column field='TaskID' headerText='Task ID'
textAlign='Right' width=120></e-column>
        <e-column field='TaskName' headerText='Task Name' width=150
:headerTemplate="hTemplate"></e-column>
        <e-column field='StartDate' headerText='Start Date'
width=150 ></e-column>
        <e-column field='Duration' headerText='Duration' width=150
></e-column>
    </e-columns>
</ejs-gantt>
</div>
</template>

<script id="customertemplate" type="text/x-template">
    <span class="e-icons e-header" ></span>
    Task Name
</script>
<script>
import Vue from "vue";
import { GanttPlugin } from "@syncfusion/ej2-vue-gantt";
import { projectNewData } from './data-source.js';
Vue.use(GanttPlugin);
var headTemplate = Vue.component("header", {
    template: '<span class="e-icons e-header">Task Name</span>',
    data() {
        return {
            data: {}
        };
    }
});
export default {
    data() {
        return {
            data: projectNewData,
            taskFields: {
                id: 'TaskID',
                name: 'TaskName',
                startDate: 'StartDate',
                duration: 'Duration',
                progress: 'Progress',
                child: 'subtasks'
            },
            hTemplate: function (e) {
                return {
                    template: headTemplate
                };
            }
        };
    },
    methods: {
        clickRestore: function () {
            let savedProperties =
JSON.parse(this.$refs.gantt.ej2Instances.getPersistData());

```

```

        let gridColumnState = Object.assign([],
    this.$refs.gantt.ej2Instances.ganttColumns);
        savedProperties.columns.forEach((col: {
            field: any;
            headerText: any;
            template: any;
            headerTemplate: any;
        }) => {
            let headerText = gridColumnState.find((colColumnsState) =>
colColumnsState.field === col.field) ['headerText'];
            let colTemplate = gridColumnState.find((colColumnsState) =>
colColumnsState.field === col.field) ['template'];
            let headerTemplate = gridColumnState.find((colColumnsState) =>
colColumnsState.field === col.field) ['headerTemplate'];
            col.headerText = 'Text Changed';
            col.template = colTemplate;
            col.headerTemplate = headerTemplate; //likewise you can restore
required column properties as per your wants.
        });
        console.log(savedProperties);

    this.$refs.gantt.ej2Instances.treeGrid.setProperties(savedProperties);
    }
}
</script>
<style>
    .e-column:before {
        content: '\e815';
    }
    .e-header:before {
        content: '\ea9a';
    }
</style>

```

{% previewsample "page.domainurl/code-snippet/gantt/column-persist-cs1" %}

### Accessibility in Vue Gantt component

The Gantt component followed the accessibility guidelines and standards, including [ADA, Section 508](#), [WCAG 2.2](#) standards, and [WCAG roles](#) that are commonly used to evaluate accessibility.

The accessibility compliance for the Gantt component is outlined below.

| Accessibility Criteria | Compatibility |

| -- | -- |

| [WCAG 2.2](#) Support |  |

| [Section 508](#) Support |  |

| Screen Reader Support |  |

| Right-To-Left Support |  |

| Color Contrast |  |

| Mobile Device Support |  |

| Keyboard Navigation Support |  |

| [Accessibility Checker](#) Validation |  |

| [Axe-core](#) Accessibility Validation | 

|

<style>

.post .post-content img {

display: inline-block;

margin: 0.5em 0;

}

</style>

<div> - All features of the component meet the requirement.</div>

<div> - Some features of the component do not meet the requirement.</div>

<div> - The component does not meet the requirement.</div>

### WAI-ARIA attributes

The Gantt component followed the [WAI-ARIA](#) patterns to meet the accessibility. The following ARIA attributes are used in the Gantt component:

The following ARIA attributes are used in Gantt:

| Attributes | Purpose |

| --- | --- |

| **grid (role)** | This attribute is added to the **e-table** element present in the Gantt, which represents Grid part |

| **gridcell (role)** | This attribute is added to the **td** elements present within the **e-table**, which represents the work cells of Gantt |

| **columnheader (role)** | This attribute is added to the **th** elements within the **e-table**, which represents the header cells of Grid table |

| **separator (role)** | This attribute is added to the **e-split-bar** element, which represents the splitter between the Grid table and Chart |

| **dialog (role)** | This attribute is added to the **e-dialog** element, which represents the pop-up dialog |

| **toolbar (role)** | This attribute is added to the **e-gantt-toolbar** element, which represents the toolbars of Gantt |

| **aria-label** | It indicates the element's information  
It is assigned to the Gantt UI elements such as timeline cell, taskbar, left label, right label, dependency line, and event markers. |

| **aria-selected** | This attribute is assigned to the Gantt chart row, and it defaults to **false**. The value is changed to **true** when the user selects a grid cell or task |

| **aria-expanded** | This attribute is assigned to the Gantt chart parent task row. The value is changed to **true** when the user clicks a parent taskbar to expand. After the user clicked a parent taskbar to collapse, the attribute value is changed to **false** |

| **aria-grabbed** | This attribute is assigned to the taskbars of Gantt when the user tries to achieve taskbar editing |

### Keyboard navigation

The Gantt component followed the [keyboard interaction](#) guideline, making it easy for people who use assistive technologies (AT) and those who completely rely on keyboard navigation. The following keyboard shortcuts are supported by the Gantt component.

| **Press** | **To do this** |

| --- | --- |

| **Home** | Selects the first row. |

| **End** | Selects the last row. |

| **DownArrow** | Moves the cell focus/row or cell selection downward. |

| **UpArrow** | Moves the cell focus/row or cell selection upward. |

| **LeftArrow** | Moves the cell focus/row or cell selection left side. |

| **RightArrow** | Moves the cell focus/row or cell selection right side. |

| **Ctrl + Up Arrow** | Collapses all tasks. |

| **Ctrl + Down Arrow** | Expands all tasks. |

| **Ctrl + Shift + Up Arrow** | Collapses the selected row. |

| **Ctrl + Shift + Down Arrow** | Expands the selected row. |

| **Enter** | Saves request. |

| **Esc** | Cancels request. |

| **Insert** | Adds a new row. |

| **Ctrl + Insert** | Opens addRowDialog. |

| **Ctrl + F2** | Opens editRowDialog. |

- | Delete | Deletes the selected row. |
- | Shift + F5 | FocusTask |
- | Ctrl + Shift + F | Focus search |
- | Shift + DownArrow | Extends the row/cell selection downwards. |
- | Shift + UpArrow | Extends the row/cell selection upwards. |
- | Shift + LeftArrow | Extends the cell selection to the left side. |
- | Shift + RightArrow | Extends the cell selection to the right side. |
- | Tab / Shift + Tab | To focus the close icon in the message. |
- | Alt + j | Focus Gantt component. |

### Ensuring accessibility

The Gantt component's accessibility levels are ensured through an [accessibility-checker](#) and [axe-core](#) software tools during automated testing.

The accessibility compliance of the Gantt component is shown in the following sample. Open the [sample](#) in a new window to evaluate the accessibility of the Gantt component with accessibility tools.

```
{% previewsample "https://ej2.syncfusion.com/accessibility/gantt.html" %}
```

See also

- [Accessibility in Syncfusion Vue components](#)

## Global local in Vue Gantt component

### Localization

The [Localization](#) library allows you to localize default text content of the Gantt.

The Gantt component has static text on some features (like toolbar area text, etc.) that can be changed to other cultures (Arabic, Deutsch, French, etc.) by defining the [locale](#) value and translation object.

The following list of properties and its values are used in the Gantt.

Locale key words |Text

emptyRecord |No records to display

id |ID

name |Name

startDate |Start Date

endDate |End Date

duration |Duration

progress |Progress

dependency |Dependency

notes | Notes

baselineStartDate | Baseline Start Date

baselineEndDate | Baseline End Date

type | Type

offset | Offset

resourceName | Resources

resourceID | Resource ID

day | day

hour | hour

minute | minute

days | days

hours | hours

minutes | minutes

generalTab | General

customTab | Custom Columns

writeNotes | Write Notes

addDialogTitle | New Task

editDialogTitle | Task Information

add | Add

edit | Edit

update | Update

delete | Delete

cancel | Cancel

search | Search

task | task

tasks | tasks

zoomIn | Zoom in

zoomOut | Zoom out

zoomToFit | Zoom to fit

expandAll | Expand all

collapseAll | Collapse all

nextTimeSpan | Next timespan

prevTimeSpan | Previous timespan



saveButton | Save

taskBeforePredecessor\_FS | You moved "{0}" to start before "{1}" finishes and the two tasks are linked. As the result, the links cannot be honored. Select one action below to perform

taskAfterPredecessor\_FS | You moved "{0}" away from "{1}" and the two tasks are linked. As the result, the links cannot be honored. Select one action below to perform

taskBeforePredecessor\_SS | You moved "{0}" to start before "{1}" starts and the two tasks are linked. As the result, the links cannot be honored. Select one action below to perform

taskAfterPredecessor\_SS | You moved "{0}" to start after "{1}" starts and the two tasks are linked. As the result, the links cannot be honored. Select one action below to perform

taskBeforePredecessor\_FF | You moved "{0}" to finish before "{1}" finishes and the two tasks are linked. As the result, the links cannot be honored. Select one action below to perform

taskAfterPredecessor\_FF | You moved "{0}" to finish after "{1}" finishes and the two tasks are linked. As the result, the links cannot be honored. Select one action below to perform

taskBeforePredecessor\_SF | You moved "{0}" away from "{1}" to starts and the two tasks are linked. As the result, the links cannot be honored. Select one action below to perform

taskAfterPredecessor\_SF | You moved "{0}" to finish after "{1}" starts and the two tasks are linked. As the result, the links cannot be honored. Select one action below to perform

okText | Ok

confirmDelete | Are you sure you want to Delete Record?

from | From

to | To

taskLink | Task Link

lag | Lag

start | Start

finish | Finish

enterValue | Enter the value

taskInformation | Task Information

deleteTask | Delete Task

deleteDependency | Delete Dependency

convert | Convert

save | Save

above | Above

below | Below

child | Child

milestone | Milestone

toTask | To Task

toMilestone | To Milestone

eventMarkers | Event markers

leftTaskLabel | Left task label

rightTaskLabel | Right task label

timelineCell | Timeline cell

confirmPredecessorDelete | Are you sure you want to remove dependency link?taskMode | Task Mode

changeScheduleMode | Change Schedule Mode

subTasksStartDate | SubTasks Start Date

subTasksEndDate | SubTasks End Date

scheduleStartDate | Schedule Start Date

scheduleEndDate | Schedule End Date

auto | Auto

manual | Manual

zoomToFit | Zoom to fit

excelExport | Excel export

csvExport | CSV export

pdfExport | Pdf export

unit | Unit

work | Work

taskType | Task Type

unassignedTask | Unassigned Task

group | Group

*Loading translations*

To load translation object in an application use [load](#) function of [L10n](#) class.

The below example demonstrates the Gantt in **Deutsch** culture.

#### **APP.VUE**

```
<template>
  <div>
    <ejs-gantt ref='gantt' id="GanttContainer" :dataSource="data"
    :taskFields = "taskFields" :height = "height" locale = "de-DE"></ejs-gantt>
  </div>
</template>
<script>
import Vue from "vue";
import { L10n, setCulture } from '@syncfusion/ej2-base';
import { GanttPlugin } from "@syncfusion/ej2-vue-gantt";
import { projectNewData } from './data-source.js';
setCulture('de-DE');
L10n.load({
```

```

    'de-DE': {
      'gantt': {
        "id": "Ich würde",
        "name": "Name",
        "startDate": "Anfangsdatum",
        "duration": "Dauer",
        "progress": "Fortschritt",
      }
    }
  });
Vue.use(GanttPlugin);
export default {
  data: function() {
    return {
      data: projectNewData,
      height: '450px',
      taskFields: {
        id: 'TaskID',
        name: 'TaskName',
        startDate: 'StartDate',
        duration: 'Duration',
        progress: 'Progress',
        child: 'subtasks'
      }
    };
  }
};
</script>

```

{% previewsample "page.domainurl/code-snippet/gantt/global-cs1" %}

### Internationalization

The [Internationalization](#) library is used to globalize number, date, and time values in gantt component.

### APP.VUE

```

<template>
  <div>
    <ejs-gantt ref='gantt' id="GanttContainer" :dataSource="data"
    :taskFields = "taskFields" :height = "height" locale = "de-DE"></ejs-gantt>
  </div>
</template>
<script>
import Vue from "vue";
import { L10n, loadCldr, setCulture } from '@syncfusion/ej2-base';
import * as cagregorian from './ca-gregorian.js';
import * as numbers from './numbers.js';
import { GanttPlugin } from "@syncfusion/ej2-vue-gantt";
import { projectNewData } from './data-source.js';
setCulture('de-DE');
L10n.load({
  'de-DE': {
    'gantt': {
      "id": "Ich würde",
      "name": "Name",
      "startDate": "Anfangsdatum",

```

```

        "duration": "Dauer",
        "progress": "Fortschritt",
    }
}
});
loadCldr(cagregorian,numbers);
Vue.use(GanttPlugin);
export default {
  data: function() {
    return{
      data: projectNewData,
      height: '450px',
      taskFields: {
        id: 'TaskID',
        name: 'TaskName',
        startDate: 'StartDate',
        duration: 'Duration',
        progress: 'Progress',
        child: 'subtasks'
      }
    };
  }
};
</script>

```

{% previewsample "page.domainurl/code-snippet/gantt/global-cs2" %}

\* In the above sample, **Timeline** is formatted by **NumberFormatOptions** and **DateFormatOptions**.

\* By default, **locale** value is **en-US**. If you want to change **en-US** culture, then set the **locale**.

### Right to left (RTL)

RTL provides an option to switch the text direction and layout of the Gantt component from right to left. It improves the user experiences and accessibility for users who use right-to-left languages (Arabic, Urdu, etc.). To enable RTL Gantt, set the **enableRtl** to true.

### APP.VUE

```

<template>
  <div>
    <ejs-gantt ref='gantt' id="GanttContainer" :dataSource="data"
    :taskFields = "taskFields" :height = "height" :toolbar='toolbar'
    :enableRtl='true' locale = "ar-AE"></ejs-gantt>
  </div>
</template>
<script>
import Vue from "vue";
import { L10n, loadCldr, setCulture } from '@syncfusion/ej2-base';
import { GanttPlugin, Toolbar } from "@syncfusion/ej2-vue-gantt";
import { projectNewData } from './data-source.js';
setCulture('ar-AE');
L10n.load({
  'ar-AE': {
    "gantt": {
      "emptyRecord": "لا سجلات لعرضها",
      "id": "هوية شخصية",
    }
  }
});

```

```

"name": "اسم",
"startDate": "تاريخ البدء",
"endDate": "تاريخ الانتهاء",
"duration": "المدة الزمنية",
"progress": "تقدم",
"dependency": "الاعتماد",
"notes": "ملاحظات",
"baselineStartDate": "تاريخ البدء الأساسي",
"baselineEndDate": "تاريخ نهاية خط الأساس",
"taskMode": "وضع المهام",
"changeScheduleMode": "تغيير وضع الجدول",
"subTasksStartDate": "تاريخ بدء المهام الفرعية",
"subTasksEndDate": "تاريخ انتهاء المهام الفرعية",
"scheduleStartDate": "جدولة تاريخ البدء",
"scheduleEndDate": "تاريخ انتهاء الجدول الزمني",
"auto": "تلقائي",
"manual": "كتيب",
"type": "اكتب",
"offset": "عوض",
"resourceName": "مصادر",
"resourceID": "معرف المورد",
"day": "يوم",
"hour": "ساعة",
"minute": "دقيقة",
"days": "أيام",
"hours": "ساعات",
"minutes": "الدقائق",
"generalTab": "جنرال لواء",
"customTab": "أعمدة مخصصة",
"writeNotes": "اكتب ملاحظات",
"addDialogTitle": "مهمة جديدة",
"editDialogTitle": "معلومات المهمة",
"saveButton": "حفظ",
"add": "إضافة",
"edit": "تعديل",
"update": "تحديث",
"delete": "حذف",
"cancel": "إلغاء",
"search": "بحث",
"task": "مهمة",
"tasks": "مهام",
"zoomIn": "تكبير",
"zoomOut": "تصغير",
"zoomToFit": "تكبير لتناسب",
"excelExport": "اكسل التصدير",
"csvExport": "تصدير CSV",
"expandAll": "توسيع الكل",
"collapseAll": "انهيار جميع",
"nextTimeSpan": "الجدول الزمني التالي",
"prevTimeSpan": "الجدول الزمني السابق",
"okText": "حسنًا",
"confirmDelete": "هل أنت متأكد أنك تريد حذف السجل؟",
"from": "من عند",
"to": "إلى",
"taskLink": "رابط المهمة",
"lag": "بطئ",
"start": "بداية",

```

```

        "finish": "إنهاء",
        "enterValue": "أدخل القيمة",
        "taskBeforePredecessor_FS": "0 '{للبدء قبل انتهاء}' لا يمكن احترام الروابط. حدد إجراء '{1}' ويتم ربط المهمتين. ونتيجة لذلك ، واحدا أدناه للقيام به",
        "taskAfterPredecessor_FS": "0 '{بعيدا عن 1}' ويتم ربط المهمتين. ونتيجة لذلك ، لا يمكن احترام الروابط. حدد إجراء واحدا أدناه للقيام به",
        "taskBeforePredecessor_SS": "0 '{للبدء قبل أن يبدأ}' لا يمكن احترام الروابط. حدد إجراء '{1}' واحدا أدناه للقيام به",
        "taskAfterPredecessor_SS": "0 '{للبدء بعد بدء 1}' لا يمكن احترام الروابط. حدد إجراء واحدا أدناه للقيام به",
        "taskBeforePredecessor_FF": "0 '{للإنهاء قبل انتهاء}' لا يمكن احترام الروابط. حدد إجراء '{1}' واحدا أدناه للقيام به",
        "taskAfterPredecessor_FF": "0 '{للإنهاء بعد انتهاء}' لا يمكن احترام الروابط. حدد إجراء '{1}' واحدا أدناه للقيام به",
        "taskBeforePredecessor_SF": "0 '{بعيدا عن 1}' لا يمكن احترام الروابط. حدد إجراء التشغيل وترتبط المهمتان. ونتيجة لذلك ، واحدا أدناه للقيام به",
        "taskAfterPredecessor_SF": "0 '{للإنهاء بعد بدء 1}' لا يمكن احترام الروابط. حدد إجراء واحدا أدناه للقيام به",
        "taskInformation": "معلومات المهمة",
        "deleteTask": "حذف المهمة",
        "deleteDependency": "حذف التبعية",
        "convert": "تحويل",
        "save": "حفظ",
        "above": "في الاعلى",
        "below": "أدناه",
        "child": "طفل",
        "milestone": "معلما",
        "toTask": "لمهمة",
        "toMilestone": "إلى معلم",
        "eventMarkers": "علامات الحدث",
        "leftTaskLabel": "تسمية المهمة اليسرى",
        "rightTaskLabel": "تسمية المهمة الصحيحة",
        "timelineCell": "خلية الجدول الزمني",
        "confirmPredecessorDelete": "هل أنت متأكد أنك تريد إزالة رابط التبعية؟",
        "unit": "وحدة",
        "work": "عمل",
        "taskType": "نوع المهمة",
        "unassignedTask": "مهمة غير محددة",
        "group": "مجموعة",
        "indent": "مسافة بادئة",
        "outdent": "عفا عليها الزمن",
        "segments": "شرائح",
        "splitTask": "تقسيم المهمة",
        "mergeTask": "مهمة الدمج",
        "left": "اليسار",
        "right": "حق"
    }
}

```

```

});
Vue.use(GanttPlugin);
export default {
  data: function() {
    return{
      data: projectNewData,
      height: '450px',
      toolbar: ['ExpandAll', 'CollapseAll'],
      taskFields: {
        id: 'TaskID',
        name: 'TaskName',
        startDate: 'StartDate',
        duration: 'Duration',
        progress: 'Progress',
        child: 'subtasks'
      }
    };
  },
  provide: {
    gantt: [ Toolbar ]
  }
};
</script>

```

{% previewsample "page.domainurl/code-snippet/gantt/global-cs3" %}

See Also

- [Internationalization](#)
- [Localization](#)

### Style and appearance in Vue Gantt component

To modify the Gantt Chart appearance, you need to override the default CSS of gantt chart. Please find the list of CSS classes and its corresponding section in Gantt Chart. Also, you have an option to create your own custom theme for all the JavaScript controls using our [Theme Studio](#).

Section | CSS Class | Purpose of Class

**Root** | e-gantt | This class is in the root element (div) of the gantt chart control.

**Header** | e-gridheader | This class is added in the root element of header element. In this class, You can override thin line between header and content of the gantt chart.

| e-table | This class is added at 'table' of the gantt chart header. This CSS class makes table width as 100 %.

| e-columnheader | This class is added at 'tr' of the gantt chart header.

**Grid Content** | e-gridcontent | This class is added at root of body content. This is to override background color of the body.

| e-table | This class is added to table of content. This CSS class makes table width as 100 %.

| e=row | This class is added to rows of gantt chart.

**|e-altrow** | This class is added to alternate rows of gantt chart. This is to override alternate row color of the gantt chart.

**|e-rowcell** | This class is added to all cells in the gantt chart. This is to override cells appearance and styling.

**Chart Content** |e-gantt-chart| This class is added to the chart side of the gantt chart.

**|e-chart-row** | This class is added to rows of gantt chart.

**Timeline** |e-timeline-header-container| This class is added to timeline of the gantt chart.

**|e-header-cell-label** | This class is added to the header cell of the timeline.

**|e-weekend-header-cell** | This class is added to the weekend cells.

**Taskbar** |e-taskbar-main-container| This class is added to taskbar of the gantt chart.

**|e-gantt-parent-taskbar** | This class is added to the parent task bar of the gantt chart.

**|e-gantt-milestone** | This class is added to the milestone tasks of the gantt chart.

**|e-gantt-unscheduled-taskbar** | This class is added to the unscheduled tasks.

**|e-gantt-manualparenttaskbar** | This class is added to the manual scheduled parent taskbar.

**|e-gantt-child-manualtaskbar** | This class is added to the manual scheduled child taskbar.

**|e-gantt-unscheduled-manualtask** | This class is added to the manual unscheduled tasks.

**Splitter** |e-split-bar| This class is added to the gantt chart splitter.

**|e-resize-handler** | This class is added to the resize handler of the gantt chart splitter.

**|e-arrow-left** | This class is added to the left arrow of the resize handler.

**|e-arrow=right** | This class is added to the right arrow of the resize handler.

**Connector Lines** |e-line| This class is added to the connector lines.

**|e-connector-line-right-arrow** | This class is added to the right arrow of the connector line.

**|e-connector-line-left-arrow** | This class is added to the left arrow of the connector line.

**Labels** |e-task-label| This class is added to the task labels.

**|e-right-label-container** | This class is added to the right label.

**|e-left-label-container** | This class is added to the left label.

**Event Markers** |e-event-markers| This class is added to the event markers.

**Baseline** |e-baseline-bar| This class is added to the baseline.

**|e-baseline-gantt-milestone-container** | This class is added to the baseline of milestone tasks.

**Tooltip** |e-gantt-tooltip| This class is added to the tooltip.

### Grid lines

In Gantt component, you can show or hide the grid lines in the TreeGrid side and chart side by using the [gridLines](#) property.

The following options are available in Gantt component for rendering grid lines,



- Horizontal: The horizontal grid lines alone will be visible.
- Vertical: The vertical grid lines alone will be visible.
- Both: Both the horizontal and vertical grid lines will be visible on the TreeGrid and chart sides.
- None: Gridlines will not be visible on TreeGrid and chart sides.

By default, the [gridLines](#) property is set with **Horizontal** type.

The following code example shows how to change the gridlines rendering mode in the Gantt component.

#### APP.VUE

```
<template>
  <div>
    <ejs-gantt ref='gantt' id="GanttContainer" :dataSource="data"
:height = "height" :taskFields = "taskFields" :gridLines="gridLines"></ejs-
gantt>
  </div>
</template>
<script>
import Vue from "vue";
import { GanttPlugin } from "@syncfusion/ej2-vue-gantt";
import { editingData } from './data-source.js';
Vue.use(GanttPlugin);
export default {
  data: function() {
    return{
      data: editingData,
      height: '450px',
      taskFields: {
        id: 'TaskID',
        name: 'TaskName',
        startDate: 'StartDate',
        duration: 'Duration',
        progress: 'Progress',
        child: 'subtasks'
      },
      gridLines : 'Both',
    };
  },
};
</script>
```

{% previewsample "page.domainurl/code-snippet/gantt/appearance-customization-cs7" %}

#### Timezone in Vue Gantt component

The Gantt makes use of the current system time zone by default. If it needs to follow some other user-specific time zone, then the **timezone** property needs to be used.

#### Understanding date manipulation in JavaScript

The **new Date()** in JavaScript returns the exact current date object with complete time and timezone information. For example, it may return value such as **Wed Dec 12 2018 05:23:27 GMT+0530 (India Standard Time)** which indicates that the current date is December 12, 2018 and the current time is 5.23 AM on browsers following the IST timezone.

Display same time everywhere with no time difference

Setting **timezone** to UTC for Gantt will display the same time as in the database for all the users in different time zone.

### APP.VUE

```
<template>
  <div>
    <ejs-gantt ref='gantt' id="GanttContainer" :dataSource="data"
    :taskFields="taskFields" :height="height" :includeWeekend="true"
    :timelineSettings="timelineSettings" :timezone="timezone"
    :durationUnit="durationUnit" :dateFormat="dateFormat"
    >
    </ejs-gantt>
  </div>
</template>
<script>
import Vue from "vue";
import { GanttPlugin, Selection } from "@syncfusion/ej2-vue-gantt";
Vue.use(GanttPlugin);
export default {
  data: function() {
    return{
      data: [
        {
          TaskID: 1,
          TaskName: 'Project Initiation',
          StartDate: new Date('04/02/2019'),
          EndDate: new Date('04/21/2019'),
          subtasks: [
            { TaskID: 2, TaskName: 'Site location', StartDate: new
Date('04/02/2019'), Duration: 4, Progress: 50 },
            { TaskID: 3, TaskName: 'Perform Soil test', StartDate: new
Date('04/02/2019'), Duration: 4, Progress: 50 },
            { TaskID: 4, TaskName: 'Soil test approval', StartDate: new
Date('04/02/2019'), Duration: 4, Progress: 50 },
          ]
        },
        {
          TaskID: 5,
          TaskName: 'Project Estimation',
          StartDate: new Date('04/02/2019'),
          EndDate: new Date('04/21/2019'),
          subtasks: [
            { TaskID: 6, TaskName: 'Develop floor plan for estimation',
StartDate: new Date('04/04/2019'), Duration: 3, Progress: 50 },
            { TaskID: 7, TaskName: 'List materials', StartDate: new
Date('04/04/2019'), Duration: 3, Progress: 50 },
            { TaskID: 8, TaskName: 'Estimation approval', StartDate: new
Date('04/04/2019'), Duration: 3, Progress: 50 }
          ]
        }
      ],
      height: '450px',
      taskFields: {
        id: 'TaskID',
        name: 'TaskName',

```

```

        startDate: 'StartDate',
        endDate: 'EndDate',
        duration: 'Duration',
        progress: 'Progress',
        dependency: 'Predecessor',
        child: 'subtasks'
    },
    timelineSettings: {
        timelineUnitSize: 65,
        topTier: {
            unit: "Day",
            format: "MMM dd, yyyy",
        },
        bottomTier: {
            unit: "Hour",
            format: "hh:mm a",
        },
    },
    dateFormat: "hh:mm a",
    selectionSettings: {
        mode: 'Both',
    },
    durationUnit: "Hour",
    timezone: "UTC"
};
},
provide: {
    gantt: [ Selection ]
}
};
</script>

```

{% previewsample "page.domainurl/code-snippet/gantt/timezone-cs1" %}

### CRUD operations with timezone

CRUD operations can be performed with timezone, and the gantt is rendered based on the timezone specified in the load time. All the editing actions will be done based on the user timezone, but on database save action, we have reversed this conversion to local time and provided data to client side events for the better understanding. Refer to the following code example.

### APP.VUE

```

<template>
  <div>
    <ejs-gantt ref='gantt' id="GanttContainer" :dataSource="data"
      :taskFields="taskFields" :height="height" :includeWeekend="true"
      :editSettings="editSettings" :timelineSettings="timelineSettings"
      :timezone="timezone" :durationUnit="durationUnit" :dateFormat="dateFormat"
      :actionComplete= "actionComplete"
    >
    </ejs-gantt>
  </div>
</template>
<script>
import Vue from "vue";
import { GanttPlugin, Selection, Edit } from "@syncfusion/ej2-vue-gantt";

```

```
Vue.use(GanttPlugin);
export default {
  data: function() {
    return{
      data: [
        {
          taskID: 1,
          taskName: 'Project Schedule',
          startDate: new Date('02/04/2019 08:00'),
          endDate: new Date('03/10/2019')
        },
        {
          taskID: 2,
          taskName: 'Planning',
          startDate: new Date('02/04/2019 08:00'),
          endDate: new Date('02/10/2019'),
          parentID: 1
        },
        {
          taskID: 3,
          taskName: 'Plan timeline',
          startDate: new Date('02/04/2019 08:00'),
          endDate: new Date('02/10/2019'),
          duration: 6,
          progress: '60',
          parentID: 2
        },
        {
          taskID: 4,
          taskName: 'Plan budget',
          startDate: new Date('02/04/2019 08:00'),
          endDate: new Date('02/10/2019'),
          duration: 6,
          progress: '90',
          parentID: 2
        },
        {
          taskID: 5,
          taskName: 'Allocate resources',
          startDate: new Date('02/04/2019 08:00'),
          endDate: new Date('02/10/2019'),
          duration: 6,
          progress: '75',
          parentID: 2
        },
        {
          taskID: 6,
          taskName: 'Planning complete',
          startDate: new Date('02/06/2019 08:00'),
          endDate: new Date('02/10/2019'),
          duration: 0,
          predecessor: '3FS,4FS,5FS',
          parentID: 2
        },
        {
          taskID: 7,
          taskName: 'Design',
```

```

      startDate: new Date('02/13/2019 08:00'),
      endDate: new Date('02/17/2019 08:00'),
      parentID: 1
    },
    {
      taskID: 8,
      taskName: 'Software Specification',
      startDate: new Date('02/13/2019 08:00'),
      endDate: new Date('02/15/2019'),
      duration: 3,
      progress: '60',
      predecessor: '6FS',
      parentID: 7
    },
    {
      taskID: 9,
      taskName: 'Develop prototype',
      startDate: new Date('02/13/2019 08:00'),
      endDate: new Date('02/15/2019'),
      duration: 3,
      progress: '100',
      predecessor: '6FS',
      parentID: 7
    },
    {
      taskID: 10,
      taskName: 'Get approval from customer',
      startDate: new Date('02/16/2019 08:00'),
      endDate: new Date('02/17/2019 08:00'),
      duration: 2,
      progress: '100',
      predecessor: '9FS',
      parentID: 7
    },
    {
      taskID: 11,
      taskName: 'Design complete',
      startDate: new Date('02/17/2019 08:00'),
      endDate: new Date('02/17/2019 08:00'),
      duration: 0,
      predecessor: '10FS',
      parentID: 7
    }
  ],
  height: '450px',
  taskFields: {
    id: 'taskID',
    name: 'taskName',
    startDate: 'startDate',
    duration: 'duration',
    progress: 'progress',
    dependency: 'predecessor',
    parentID: 'parentID'
  },
  editSettings: {
    allowAdding: true,
    allowEditing: true,

```

```

        allowDeleting: true,
        allowTaskbarEditing: true,
        showDeleteConfirmDialog: true
    },
    dateFormat: "hh:mm a",
    selectionSettings: {
        mode: 'Both',
    },
    timelineSettings: {
        timelineUnitSize: 65,
        topTier: {
            unit: "Day",
            format: "MMM dd, yyyy",
        },
        bottomTier: {
            unit: "Hour",
            format: "hh:mm a",
        },
    },
    durationUnit: "Hour",
    timezone: "America/New_York",
    };
},
provide: {
    gantt: [ Selection, Edit]
},
methods:{
    actionComplete(args) {
        if (args.action == "TaskbarEditing") {
            console.log(args.data.ganttProperties.endDate);
        }
    }
}
};
</script>

```

{% previewsample "page.domainurl/code-snippet/gantt/timezone-cs2" %}

## Timezone methods

### offset

This method is used to calculate the difference between passed UTC date and timezone.

Parameters	Type	Description
----- -----	-----	-----
Date	Date	UTC time as date object.
Timezone	String	Timezone.

Returns **number**

`ts

// Assume your local timezone as IST/UTC+05:30

let timezone: Timezone = new Timezone();

let date: Date = new Date(2018,11,5,15,25,11);

```
let timeZoneOffset: number = timezone.offset(date,"Europe/Paris");  
console.log(timeZoneOffset); //-60
```

,

#### *convert*

This method is used to convert the passed date from one timezone to another timezone.

Parameters	Type	Description
----- ----- -----		
Date	Date	UTC time as date object.
fromOffset	number/string	Timezone from which date need to be converted.
toOffset	number/string	Timezone to which date need to be converted.

Returns **Date**

`ts

```
// Assume your local timezone as IST/UTC+05:30  
let timezone: Timezone = new Timezone();  
let date: Date = new Date(2018,11,5,15,25,11);  
let convertedDate: Date = timezone.convert(date, "Europe/Paris", "Asia/Tokya");  
let convertedDate1: Date = timezone.convert(date, 60, -360);  
console.log(convertedDate); //2018-12-05T08:55:11.000Z  
console.log(convertedDate1); //2018-12-05T16:55:11.000Z
```

,

#### *remove*

This method is used to remove the time difference between passed UTC date and timezone.

Parameters	Type	Description
----- ----- -----		
Date	Date	UTC as date object.
Timezone	String	Timezone.

Returns **Date**

`ts

```
// Assume your local timezone as IST/UTC+05:30  
let timezone: Timezone = new Timezone();  
let date: Date = new Date(2018,11,5,15,25,11);  
let convertedDate: Date = timezone.remove(date, "Europe/Paris");  
console.log(convertedDate); //2018-12-05T14:25:11.000Z
```

,

## How To

### Open add edit dialog in Vue Gantt component

In the Gantt component, add and edit dialogs can be opened dynamically by using [openAddDialog](#) and [openEditDialog](#) methods. The following code example shows how to open add and dialog on separate button click actions.

#### APP.VUE

```
<template>
  <div>
    <ejs-button id="editDialog" cssClass="e-info" v-
on:click.native="edit">edit</ejs-button>
    <br><br>
    <ejs-button id="addDialog" cssClass="e-info" v-
on:click.native="add">add</ejs-button>
    <br><br>
    <ejs-gantt ref='gantt' id="GanttContainer" :dataSource="data"
:taskFields= "taskFields" :height="height"
:editDialogFields="editDialogFields" :editSettings="editSettings"
:resourceNameMapping= "resourceNameMapping"
:resourceIdMapping="resourceIdMapping" :resources= "resources"></ejs-gantt>
  </div>
</template>
<script>
import Vue from "vue";
import { GanttPlugin, Edit, Selection } from "@syncfusion/ej2-vue-gantt";
import { ButtonPlugin } from "@syncfusion/ej2-vue-buttons";
import { editingData , editingResources } from './data-source.js';
Vue.use(GanttPlugin);
Vue.use(ButtonPlugin);
export default {
  data: function() {
    return{
      data: editingData,
      taskFields: {
        id: 'TaskID',
        name: 'TaskName',
        startDate: 'StartDate',
        endDate: 'EndDate',
        duration: 'Duration',
        progress: 'Progress',
        dependency: 'Predecessor',
        child: 'subtasks',
        notes: 'info',
        resourceInfo: 'resources'
      },
      editDialogFields: [
        { type: 'General', headerText: 'General' },
        { type: 'Dependency' },
        { type: 'Resources' },
        { type: 'Notes' }
      ],
      height: '450px',
      resourceNameMapping: 'resourceName',
      resourceIdMapping: 'resourceId',
      resources: editingResources,
```



```

        editSettings: {
            allowEditing: true,
            allowTaskbarEditing: true
        }
    };
},
provide: {
    gantt: [ Edit, Selection ]
},
methods: {
    edit: function(e) {
        var ganttObj =
document.getElementById('GanttContainer').ej2_instances[0];
        ganttObj.editModule.dialogModule.openEditDialog();
    },
    add: function(e) {
        var ganttObj =
document.getElementById('GanttContainer').ej2_instances[0];
        ganttObj.editModule.dialogModule.openAddDialog();
    }
}
});
</script>

```

{% previewsample "page.domainurl/code-snippet/gantt/how-to/open-add-edit-cs1" %}

NOTE: You should select any one of the row in the Gantt to open the edit dialog.

### Change schedule dates in Vue Gantt component

In the Gantt component, you can change the schedule start and end dates by clicking the custom button programmatically using the [updateProjectDates](#) method. You can pass the start and end dates as method arguments to the [updateProjectDates](#) method. You can also pass the Boolean value as an additional parameter, which is used to round-off the schedule start and end dates displayed in Gantt timeline.

### APP.VUE

```

<template>
  <div>
    <ejs-button id="changedate" cssClass="e-info" v-
on:click.native="change">Change Date</ejs-button>
    <br><br><br>
    <ejs-gantt id="GanttContainer" :dataSource="data" :taskFields =
"taskFields" :height = "height"></ejs-gantt>
  </div>
</template>
<script>
import Vue from "vue";
import { GanttPlugin } from "@syncfusion/ej2-vue-gantt";
import { ButtonPlugin } from "@syncfusion/ej2-vue-buttons";
import { editingData } from './data-source.js';
Vue.use(GanttPlugin);
Vue.use(ButtonPlugin);
export default {
  data: function() {
    return{

```

```

        data: editingData,
        height: '450px',
        taskFields: {
            id: 'TaskID',
            name: 'TaskName',
            startDate: 'StartDate',
            duration: 'Duration',
            progress: 'Progress',
            child: 'subtasks'
        },
    },
    methods: {
        change: function(e) {
            var ganttObj =
document.getElementById('GanttContainer').ej2_instances[0];
            ganttObj.updateProjectDates(new Date('03/10/2019'), new
Date('06/20/2019'), true);
        }
    },
};
</script>

```

{% previewsample "page.domainurl/code-snippet/gantt/how-to/changescheduledates-cs1" %}

### Copypasterecords in Vue Gantt component

You can copy and paste a record in the Gantt chart by using the `addRecord` method and `custom context menu`. It is also possible to copy and paste the parent record with multiple hierarchical child records on the required position.

### APP.VUE

```

<template>
  <div>
    <ejs-gantt ref='gantt' id="customContextMenu" :dataSource="data"
:taskFields = "taskFields" :height = "height" :editSettings="editSettings"
:enableContextMenu="true" :contextMenuItems="contextMenuItems"
:contextMenuClick = "contextMenuClick" :contextMenuOpen= "contextMenuOpen"
:addChildRecords= "addChildRecords"></ejs-gantt>
  </div>
</template>
<script>
import Vue from "vue";
import { GanttPlugin, ContextMenu, Edit, Selection } from "@syncfusion/ej2-
vue-gantt";
import { editingData } from './data-source.js';
Vue.use(GanttPlugin);
var copiedRecord;
export default {
  data: function() {
    return{
      data: editingData,
      height: '450px',
      taskFields: {
        id: 'TaskID',
        name: 'TaskName',

```

```

        startDate: 'StartDate',
        duration: 'Duration',
        progress: 'Progress',
        dependency: 'Predecessor',
        child: 'subtasks'
    },
    editSettings: {
        allowAdding: true,
        allowEditing: true,
        allowDeleting: true
    },
    contextMenuItems: [{ text: 'Copy', target: '.e-content', id: 'copy'
},
    { text: 'Paste', target: '.e-content', id: 'paste' }
],
    };
},
provide: {
    gantt: [ ContextMenu, Edit, Selection]
},
methods: {
    contextMenuClick: function (args) {
        var gantt =
document.getElementById('customContextMenu').ej2_instances[0];
        if (args.item.id === 'copy') {
            copiedRecord = args.rowData;
            copiedRecord.taskData.TaskID = gantt.currentViewData.length + 1;
        }
        if (args.item.id === 'paste') {
            gantt.addRecord(copiedRecord.taskData, 'Below', args.rowData.index);
            if(copiedRecord.hasChildRecords) {
                addChildRecords(copiedRecord, args.rowData.index + 1);
            }
            copiedRecord = undefined;
        }
    },
    contextMenuOpen: function (args) {
        if (args.type !== 'Header') {
            if (copiedRecord) {
                args.hideItems.push('Copy');
            } else {
                args.hideItems.push('Paste');
            }
        }
    },
    addChildRecords(record, index) {
        for(var i=0; i<record.childRecords.length; i++) {
            var childRecord = record.childRecords[i];
            childRecord.taskData.TaskID = gantt.currentViewData.length + 1;
            gantt.addRecord(childRecord.taskData, 'Child', index);
            if(childRecord.hasChildRecords) {
                addChildRecords(childRecord, index + (i+1));
            }
        }
    }
}
}
}

```

```
};  
</script>
```

```
{% previewsample "page.domainurl/code-snippet/gantt/how-to/copypasterecords-cs1" %}
```

### Maintain record index in Vue Gantt component

Row dropped record's index position can be maintained in the Gantt chart by changing the database table index position using the `rowDrop` event. In this event, the `fromIndex` and `dropIndex` values can be passed to the server side using Ajax request. On the server side, the `insert` and `insertAtTop` methods are used to update the row index position. The following code snippets explain the solution.

```
`ts
```

```
<template>
```

```
<div>
```

```
<ejs-gantt :dataSource="data" :allowRowDragAndDrop='true' :taskFields = "taskFields" :height =  
"height"></ejs-gantt>
```

```
</div>
```

```
</template>
```

```
<script>
```

```
import Vue from "vue";
```

```
import { GanttPlugin, RowDD, Edit, Selection } from "@syncfusion/ej2-vue-gantt";
```

```
import { DataManager, UrlAdaptor } from '@syncfusion/ej2-data';
```

```
Vue.use(GanttPlugin);
```

```
export default {
```

```
  data: function() {
```

```
    return{
```

```
      data: new DataManager({
```

```
        url: 'https://localhost:44379/Home/UrlDatasource',
```

```
        adaptor: new UrlAdaptor,
```

```
        crossDomain: true,
```

```
        batchUrl: 'https://localhost:44379/Home/BatchUpdate'
```

```
      }},
```

```
      height: '450px',
```

```
      taskFields: {
```

```
        id: 'TaskID',
```

```
        name: 'TaskName',
```

```
        startDate: 'StartDate',
```

```
duration: 'Duration',
progress: 'Progress',
dependency: 'Predecessor',
child: 'subtasks'
},
rowDrop: function(args) {
var record = this.flatData[args.fromIndex][this.taskFields.id];
var record2 = this.flatData[args.dropIndex][this.taskFields.id];
var data: IGanttData = args.data[0];
var uri = 'https://localhost:44379/Home/RowDropMethod';
var dragdropdata = {
record: data[0].taskData,
position: args.dropIndex,
dragidMapping: record,
dropidMapping: record2
};
var ajax = new Ajax(
{
url: uri,
type: 'POST',
contentType: "application/json",
data: JSON.stringify(dragdropdata),
});
ajax.send();
}
};
},
provide: {
ganttt: [ RowDD, Edit, Selection ]
}
};
</script>
`
```

```
`ts
public IActionResult RowDropMethod([FromBody]DragDropData value)
{
    var data = new CRUDModel();
    copyRecord = true;
    if (value.position == "bottomSegment" || value.position == "topSegment")
    {
        {
            var childCount = 0;
            int parent1 = (int)value.record.parentID;
            childCount += FindChildRecords(parent1);
            if (childCount == 1)
            {
                var i = 0;
                for (; i < DataList.Count; i++)
                {
                    if (DataList[i].taskID == parent1)
                    {
                        DataList[i].isParent = false;
                        break;
                    }
                    if (DataList[i].taskID == value.record.taskID)
                    {
                        DataList[i].parentID = null;
                        break;
                    }
                }
            }
            DataList.Remove(DataList.Where(ds => ds.taskID == value.dragidMapping).FirstOrDefault());
            var j = 0;
            for (; j < DataList.Count; j++)
            {
```

```

if (DataList[j].taskID == value.dropidMapping)
{
    value.record.parentID = DataList[j].parentID;
    break;
}
}
data.Value = value.record;
if (value.position == "bottomSegment")
{
    this.Insert(data, value.dropidMapping);
}
else if (value.position == "topSegment")
{
    this.InsertAtTop(data, value.dropidMapping);
}
}
else if (value.position == "middleSegment")
{
    DataList.Remove(DataList.Where(ds => ds.taskID == value.dragidMapping).FirstOrDefault());
    value.record.parentID = value.dropidMapping;
    FindDropdata(value.dropidMapping);
    data.Value = value.record;
    this.Insert(data, value.dropidMapping);
}
copyRecord = false;
return Json(new { updatedRecords = value.record });
}
,

```

#### Custom field in Vue Gantt component

Generally in Gantt, Custom fields are displayed in the Custom Tab of the Add/Edit dialogs. However, they can be included in the General Tab of Add/Edit Dialog Box using `actionBegin` and `actionComplete` events. These events are used to append the custom field to the dialog box. The following code snippets demonstrate the solution.

#### **APP.VUE**

```

<template>
  <div id="app">
    <ejs-gantt ref="gantt" :dataSource="data" id="GanttContainer"
      :taskFields="taskFields" :height="height" :editSettings="editSettings"
      :editDialogFields="editDialogFields" :addDialogFields=
      "addDialogFields" :toolbar= "toolbar" :actionBegin="actionBegin"
      :actionComplete="actionComplete" >
      <e-columns>
        <e-column field="TaskID" headerText="Task ID" textAlign="Left"
width="100"></e-column>
        <e-column field="TaskName" headerText="Task Name" width="150"></e-
column>
        <e-column field="StartDate" headerText="Start Date" width="150"></e-
column>
        <e-column field="EndDate" headerText="End Date" width="150"></e-
column>
        <e-column field="Duration" headerText="Duration" width="150"></e-
column>
        <e-column field="Progress" headerText="Progress" width="150"></e-
column>
        <e-column field="CustomField" headerText="CustomField"
width="150"></e-column>
      </e-columns>
    </ejs-gantt>
  </div>
</template>
<script>
import Vue from "vue";
import { GanttPlugin, Edit, Toolbar, Selection } from "@syncfusion/ej2-vue-
gantt";
import { CheckBox } from "@syncfusion/ej2-buttons";
import { TextBox, NumericTextBox, MaskedTextBox } from "@syncfusion/ej2-
inputs";
import { DatePicker, DateTimePicker } from "@syncfusion/ej2-calendars";
import { DropDownList } from "@syncfusion/ej2-dropdowns";
import { editingData } from './data-source.js';
Vue.use(GanttPlugin);
var divElement;
var inputs = {
  booleanedit: CheckBox,
  dropdownedit: DropDownList,
  datepickeredit: DatePicker,
  datetimeredit: DateTimePicker,
  maskededit: MaskedTextBox,
  numericedit: NumericTextBox,
  stringedit: TextBox
};
export default {
  data: function() {
    return {
      data: editingData,
      height: "450px",
      taskFields: {
        id: "TaskID",
        name: "TaskName",
        startDate: "StartDate",
        endDate: "EndDate",

```



```

        duration: "Duration",
        progress: "Progress",
        child: "subtasks"
    },
    editSettings: {
        allowAdding: true,
        allowEditing: true,
        allowDeleting: true,
        mode: "Auto"
    },
    toolbar: ['Add', 'Edit', 'Update', 'Delete', 'Cancel', 'ExpandAll',
'CollapseAll'],
    editDialogFields: [
        { type: "General", headerText: "General" },
        { type: "Dependency" },
        { type: "Resources" },
        { type: "Notes" }
    ],
    addDialogFields : [
        { type: 'General', headerText: 'General' },
        { type: 'Dependency' },
        { type: "Resources" },
        { type: "Notes" }
    ],
    actionBegin: function(args) {
        if (args.requestType === "beforeOpenEditDialog" || args.requestType
=== "beforeOpenAddDialog" ) {
            var column = this.columnByField["CustomField"];
            divElement = this.createElement("div", {
                className: "e-edit-form-column"
            });
            var inputElement;
            inputElement = this.createElement("input", {
                attrs: {
                    type: "text",
                    id: this.controlId + "" + column.field,
                    name: column.field,
                    title: column.field
                }
            });
            divElement.appendChild(inputElement);
            var input = {
                enabled: true,
                floatLabelType: "Auto",
                placeholder: "CustomField",
                value: args.rowData.CustomField
            };
            var inputObj = new inputs[column.editType](input);
            inputObj.appendTo(inputElement);
        }
    },
    actionComplete: function(args) {
        if (args.requestType === "openEditDialog" || args.requestType ===
"openAddDialog" ) {
            var generalTab = document.getElementById(
                this.controlId + "GeneralTabContainer"
            );

```

```

        generalTab.appendChild(divElement);
    }
}
};
},
provide: {
    gantt: [Edit, Toolbar, Selection]
},
methods: {}
};
</script>

```

{% previewsample "page.domainurl/code-snippet/gantt/how-to/customfields-cs1" %}

### Maintain zoom to fit in Vue Gantt component

In the Gantt control, While performing edit actions or dynamically change dataSource, the timeline gets refreshed. When zoomToFit toolbar item is clicked and perform editing actions or dynamically change dataSource, the timeline gets refreshed. So that, the timeline will not fit to the project any more.

### Maintain zoomToFit after edit actions

We can maintain zoomToFit after editing actions(cell edit,dialog edit,taskbar edit) by using [fitToProject](#) method in `actionComplete` and `taskbarEdited` event.

### APP.VUE

```

<template>
  <div>
    <ejs-gantt ref='gantt' id="GanttContainer" :dataSource="data"
    :taskFields = "taskFields" :height = "height" :toolbar="toolbar"
    :editSettings= "editSettings" :actionComplete="actionComplete"
    :taskbarEdited="taskbarEdited"></ejs-gantt>
  </div>
</template>
<script>
import Vue from "vue";
import { GanttPlugin, Toolbar, Edit, Selection } from "@syncfusion/ej2-vue-gantt";
import { projectNewData } from './data-source.js';
Vue.use(GanttPlugin);
export default {
  data: function() {
    return{
      data: projectNewData,
      height: '450px',
      taskFields: {
        id: 'TaskID',
        name: 'TaskName',
        startDate: 'StartDate',
        duration: 'Duration',
        progress: 'Progress',
        dependency: 'Predecessor',
        child: 'subtasks'
      },
      toolbar: ['Edit','ZoomToFit'],
      editSettings: {
        allowEditing: true,

```

```

        allowTaskbarEditing: true
    },
    actionComplete: function(args) {
        if ((args.action === "CellEditing" || args.action ===
"DialogEditing") && args.requestType === "save") {
            var obj =
document.getElementById("GanttContainer").ej2_instances[0];
            obj.dataSource = projectNewData;
            obj.fitToProject();
        }
    },
    taskbarEdited: function(args) {
        if (args) {
            var obj =
document.getElementById("GanttContainer").ej2_instances[0];
            obj.dataSource = projectNewData;
            obj.fitToProject();
        }
    }
};
},
provide: {
    gantt: [Toolbar, Edit, Selection ]
}
};
</script>

```

{% previewsample "page.domainurl/code-snippet/gantt/how-to/maintainzoomtofit-cs1" %}

*Maintain zoomToFit after change dataSource dynamically*

We can maintain **zoomToFit** after change dataSource dynamically, by calling [fitToProject](#) method in dataBound event.

#### APP.VUE

```

<template>
  <div>
    <ejs-button id="changeData" cssClass="e-info" v-
on:click.native="changeData">Change Data</ejs-button>
    <br><br><br>
    <ejs-gantt ref='gantt' id="GanttContainer" :dataSource="data"
:taskFields = "taskFields" :height = "height" :toolbar="toolbar"
:dataBound="dataBound"></ejs-gantt>
  </div>
</template>
<script>
import Vue from "vue";
import { GanttPlugin, Toolbar } from "@syncfusion/ej2-vue-gantt";
import { ButtonPlugin } from "@syncfusion/ej2-vue-buttons";
import { projectNewData, data } from './data-source.js';
Vue.use(GanttPlugin);
Vue.use(ButtonPlugin);
export default {
  data: function() {
    return{
      data: projectNewData,

```

```

        height: '450px',
        taskFields: {
            id: 'TaskID',
            name: 'TaskName',
            startDate: 'StartDate',
            duration: 'Duration',
            progress: 'Progress',
            dependency: 'Predecessor',
            child: 'subtasks'
        },
        toolbar: ['ZoomToFit'],
        dataBound: function(args) {
            var obj =
document.getElementById('GanttContainer').ej2_instances[0];
            obj.fitToProject();
        },
    };
},
methods: {
    changeData: function(e) {
        var obj =
document.getElementById('GanttContainer').ej2_instances[0];
        obj.dataSource = data;
    }
}
provide: {
    gantt: [Toolbar]
}
};
</script>

```

{% previewsample "page.domainurl/code-snippet/gantt/how-to/maintainzoomtofitdatasource-cs1" %}

### Drag and drop from another in Vue Gantt component

In Gantt, it is possible to drag a record from another component and drop it in Gantt chart with updating the Gantt record. Here, dragging an item from **TreeView** component to Gantt and that item is updated as a resource for the Gantt record, we can achieve this, by using [nodeDragStop](#) event of [Link to the Video](#) control.

To learn about Gantt Chart Drag and Drop, you can check on this video:

### APP.VUE

```

<template>
  <div>
    <p><b>Gantt</b></p>
    <ejs-gantt ref='gantt' id="GanttContainer" :dataSource="data"
:taskFields = "taskFields" :height = "height"
:resourceFields= "resourceFields" :resources= "resources"
:labelSettings= "labelSettings" :editSettings= "editSettings"></ejs-gantt>
    <p><b>List</b></p>
    <ejs-treeview id='treeview' :fields="fields" :allowDragAndDrop='true'
:nodeDragStop="nodeDragStop"></ejs-treeview>
  </div>
</template>
<script>

```

```

import Vue from "vue";
import { GanttPlugin, Edit, Selection } from "@syncfusion/ej2-vue-gantt";
import { TreeViewPlugin } from "@syncfusion/ej2-vue-navigations";
import { editingData, editingResources } from './data-source.js';
import { closest, addClass } from '@syncfusion/ej2-base';
Vue.use(GanttPlugin);
Vue.use(TreeViewPlugin);
export default {
  data: function() {
    return {
      data: editingData,
      resources: editingResources,
      fields: { dataSource: editingResources, id: 'resourceId', text:
'resourceName' },
      height: '450px',
      taskFields: {
        id: 'TaskID',
        name: 'TaskName',
        startDate: 'StartDate',
        duration: 'Duration',
        progress: 'Progress',
        dependency: 'Predecessor',
        resourceInfo: 'resources',
        child: 'subtasks'
      },
      splitterSettings: {
        position: "30%"
      },
      labelSettings: {
        rightLabel: 'resources'
      },
      editSettings: {
        allowEditing: true
      },
      resourceFields: {
        id: 'resourceId',
        name: 'resourceName'
      },
    };
  },
  provide: {
    gantt: [Edit, Selection]
  },
  methods: {
    nodeDragStop: function (args) {
      var ganttChart =
document.getElementById('GanttContainer').ej2_instances[0];
      var chartEle = closest(args.target, '.e-chart-row');
      var gridEle = closest(args.target, '.e-row');
      if(gridEle) {
        var index = ganttChart.treeGrid.getRows().indexOf(gridEle);
        ganttChart.selectRow(index);
      }
      if(chartEle) {
        var index = chartEle.ariaRowIndex;
        ganttChart.selectRow(Number(index));
      }
    }
  }
}

```

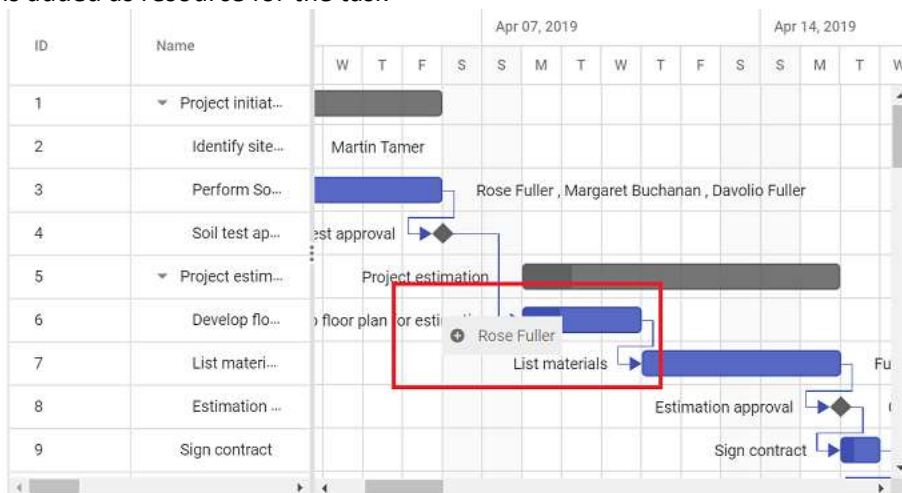
```

var record= args.draggedNodeData;
var selectedData =
ganttChart.flatData[ganttChart.selectedRowIndex];
var selectedDataResource = selectedData.taskData.resources;
var resources = [];
if (selectedDataResource) {
    for (var i = 0; i < selectedDataResource.length; i++) {
        resources.push(selectedDataResource[i].resourceId);
    }
}
resources.push(parseInt(record.id));
if (chartEle || gridEle) {
    var data = {
        TaskID: selectedData.taskData.TaskID,
        resources: resources
    };
    ganttChart.updateRecordByID(data);
    var elements = document.querySelectorAll('.e-drag-item');
    while (elements.length > 0 && elements[0].parentNode) {
        elements[0].parentNode.removeChild(elements[0]);
    }
}
}
}
};
</script>

```

{% previewsample "page.domainurl/code-snippet/gantt/how-to/draganddrop-cs1" %}

The following screenshot shows dropping record from another component in to Gantt, and **Rose Fuller** is added as resource for the task



#### Flat list

Martin Tamer
Rose Fuller
Margaret Buchanan
Fuller King

#### Develop floor plan

estimation.

![Dropping Record](../images/dropping.png)

### New row position in Vue Gantt component

In Gantt, a new row can be added in one of the following positions: Top, Bottom, Above, Below and Child. This position can be specified through the `newRowPosition` property. We can make use of the `toolbarClick` event to create a context menu that specifies the position in which the new row is to be added when adding a record through toolbar click.

The following code snippets demonstrate how to achieve this.

#### APP.VUE

```
<template>
  <div>
    <ejs-contextmenu id='cmenu' :items='menuItems' :select="select"></ejs-
contextmenu>
    <ejs-gantt ref='gantt' id="GanttContainer" :dataSource="data"
:taskFields = "taskFields" :height = "height" :toolbar="toolbar"
:toolbarClick="toolbarClick" :editSettings= "editSettings"></ejs-gantt>
  </div>
</template>
<script>
import Vue from "vue";
import {
  GanttPlugin,
  Edit,
  Toolbar,
  Selection,
} from "@syncfusion/ej2-vue-gantt";
import { ContextMenuPlugin } from "@syncfusion/ej2-vue-navigations";
Vue.use(GanttPlugin);
Vue.use(ContextMenuPlugin);
export default {
  data: function () {
    return {
      menuItems: [
        {
          text: "Bottom",
        },
        {
          text: "Above",
        },
        {
          text: "Below",
        },
        {
          text: "Child",
        },
        {
          text: "Top",
        },
      ],
      data: [
        {
          TaskID: 1,
          TaskName: "Project Initiation",
          StartDate: new Date("04/02/2019"),
```

```

        EndDate: new Date("04/21/2019"),
        subtasks: [
            {
                TaskID: 2,
                TaskName: "Identify Site location",
                StartDate: new Date("04/02/2019"),
                Duration: 4,
                Progress: 50,
            },
            {
                TaskID: 3,
                TaskName: "Perform Soil test",
                StartDate: new Date("04/02/2019"),
                Duration: 4,
                Progress: 50,
            },
            {
                TaskID: 4,
                TaskName: "Soil test approval",
                StartDate: new Date("04/02/2019"),
                Duration: 4,
                Progress: 50,
            },
        ],
    },
    {
        TaskID: 5,
        TaskName: "Project Estimation",
        StartDate: new Date("04/02/2019"),
        EndDate: new Date("04/21/2019"),
        subtasks: [
            {
                TaskID: 6,
                TaskName: "Develop floor plan for estimation",
                StartDate: new Date("04/04/2019"),
                Duration: 3,
                Progress: 50,
            },
            {
                TaskID: 7,
                TaskName: "List materials",
                StartDate: new Date("04/04/2019"),
                Duration: 3,
                Progress: 50,
            },
            {
                TaskID: 8,
                TaskName: "Estimation approval",
                StartDate: new Date("04/04/2019"),
                Duration: 3,
                Progress: 50,
            },
        ],
    },
],
height: "450px",
taskFields: {

```



```

        id: "TaskID",
        name: "TaskName",
        startDate: "StartDate",
        duration: "Duration",
        progress: "Progress",
        child: "subtasks",
    },
    editSettings: {
        allowAdding: true,
    },
    toolbar: [
        {
            text: "Add",
            tooltipText: "Add",
            id: "Add",
        }
    ],
    toolbarClick: (args) => {
        if (args.item.id === "Add") {
            document.getElementById("cmenu").ej2_instances[0].open(40, 20);
        }
    },
};
},
methods: {
    select: function (args) {
        var ganttObj =
document.getElementById("GanttContainer").ej2_instances[0];
        if (args.item.text === "Bottom") {
            ganttObj.editSettings.newRowPosition = "Bottom";
            ganttObj.openAddDialog();
        } else if (args.item.text === "Above") {
            if (ganttObj.selectedRowIndex === -1) {
                alert("Please select any row");
            } else {
                ganttObj.editSettings.newRowPosition = "Above";
                ganttObj.openAddDialog();
            }
        } else if (args.item.text === "Below") {
            if (ganttObj.selectedRowIndex === -1) {
                alert("Please select any row");
            } else {
                ganttObj.editSettings.newRowPosition = "Below";
                ganttObj.openAddDialog();
            }
        } else if (args.item.text === "Child") {
            if (ganttObj.selectedRowIndex === -1) {
                alert("Please select any row");
            } else {
                ganttObj.editSettings.newRowPosition = "Child";
                ganttObj.openAddDialog();
            }
        } else if (args.item.text === "Top") {
            ganttObj.editSettings.newRowPosition = "Top";
            ganttObj.openAddDialog();
        }
    },
},

```

```
    },  
    provide: {  
      gantt: [Edit, Toolbar, Selection],  
    },  
  };  
</script>
```

{% previewsample "page.domainurl/code-snippet/gantt/how-to/new-row-position-cs1" %}

## Grid

### Getting Started with the Vue Grid Component in Vue 2

This article provides a step-by-step guide for setting up a Vue 2 project using [Vue-CLILink to the Video](#) and integrating the Syncfusion Vue Grid component

To get start quickly with Vue Grid, you can check on this video:

Prerequisites

[System requirements for Syncfusion Vue UI components](#)

Setting up the Vue 2 project

To generate a Vue 2 project using Vue-CLI, use the [vue create](#) command. Follow these steps to install Vue CLI and create a new project:

```
`bash
```

```
npm install -g @vue/cli
```

```
vue create quickstart
```

```
cd quickstart
```

```
npm run serve
```

```
,
```

```
or
```

```
`bash
```

```
yarn global add @vue/cli
```

```
vue create quickstart
```

```
cd quickstart
```

```
yarn run serve
```

```
,
```

When creating a new project, choose the option **Default ([Vue 2] babel, eslint)** from the menu.

```
E:\samples>vue create quickstart

Vue CLI v5.0.8
? Please pick a preset:
  Default ([Vue 3] babel, eslint)
> Default ([Vue 2] babel, eslint)
  Manually select features
```

Once the `quickstart` project is set up with default settings, proceed to add Syncfusion components to the project.

#### Vue 3 Application

Select the option `Default ([Vue 3] babel, eslint)` from the menu.

```
E:\samples>vue create quickstart

Vue CLI v5.0.8
? Please pick a preset:
> Default ([Vue 3] babel, eslint)
  Default ([Vue 2] babel, eslint)
  Manually select features
```

#### Add Syncfusion Vue packages

Syncfusion packages are available at [npmjs.com](https://www.npmjs.com). To use Vue components, install the required npm package.

This article uses the [Vue Grid component](#) as an example. Install the `@syncfusion/ej2-vue-grids` package by running the following command:

```
`bash
```

```
npm install @syncfusion/ej2-vue-grids --save
```

```
`
```

or

```
`bash
```

```
yarn add @syncfusion/ej2-vue-grids
```

```
`
```

#### Import Syncfusion CSS styles

You can import themes for the Syncfusion Vue component in various ways, such as using CSS or SASS styles from npm packages, CDN, CRG and [Theme Studio](#). Refer to [themes topic](#) to know more about built-in themes and different ways to refer to themes in a Vue project.

In this article, the `Material` theme is applied using CSS styles, which are available in installed packages. The necessary Material CSS styles for the Grid component and its dependents were imported into the `<style>` section of `src/App.vue` file.

~/SRC/APP.VUE

```
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-buttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-calendars/styles/material.css";
@import "../node_modules/@syncfusion/ej2-dropdowns/styles/material.css";
@import "../node_modules/@syncfusion/ej2-inputs/styles/material.css";
@import "../node_modules/@syncfusion/ej2-navigations/styles/material.css";
@import "../node_modules/@syncfusion/ej2-popups/styles/material.css";
@import "../node_modules/@syncfusion/ej2-splitbuttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
</style>
```

Grid components use other Syncfusion components as well, so CSS references for the dependent component must be added in order to use all grid functionalities. Use this same order to display the Syncfusion Grid component's predefined appearance.

### Add Syncfusion Vue component

Follow the below steps to add the Vue Grid component:

- 1\ First, import and register the Grid component in the `script` section of the `src/App.vue` file.

~/SRC/APP.VUE

```
<script>
import { GridComponent } from "@syncfusion/ej2-vue-grids";
export default {
  components: {
    'ejs-grid': GridComponent
  }
}
</script>
```

- 2\ In the `template` section, define the Grid component.

~/SRC/APP.VUE

```
<template>
<div id="app">
<ejs-grid> </ejs-grid>
</div>
</template>
```

### Register the Syncfusion Vue component

A Syncfusion Vue component needs to be registered so that Vue knows where to locate its implementation when it is encountered in a template.

### Vue 3 Application

Import the Grid component along with the required child directives from the installed packages into the `<script>` section of the `src/App.vue` file. Register the Grid component along with the required child directives using following code.

```
`js
```

```
import { GridComponent, ColumnsDirective, ColumnDirective } from '@syncfusion/ej2-vue-grids';
// Component registration
export default {
  name: "App",
  components: {
    'ejs-grid' : GridComponent,
    'e-columns' : ColumnsDirective,
    'e-column' : ColumnDirective
  }
},
,
```

Now, the Grid and column directives are registered to use it in this application.

#### Defining Row Data

Data for the Grid component is bind by using [dataSource](#) property and value is defined in the vue component. It accepts either array of JavaScript object or **DataManager** instance.

```
,
<template>
<div id="app">
<ejs-grid :dataSource="data"> </ejs-grid>
</div>
</template>
<script>
import { GridComponent } from "@syncfusion/ej2-vue-grids";
export default {
  components: {
    'ejs-grid': GridComponent
  },
  data () {
    return {
      data: [
        { OrderID: 10248, CustomerID: 'VINET', Freight: 32.38 },
        { OrderID: 10249, CustomerID: 'TOMSP', Freight: 11.61 },
        { OrderID: 10250, CustomerID: 'HANAR', Freight: 65.83 },
        { OrderID: 10251, CustomerID: 'VICTE', Freight: 41.34 },
```

```

{ OrderID: 10252, CustomerID: 'SUPRD', Freight: 51.3 },
{ OrderID: 10253, CustomerID: 'HANAR', Freight: 58.17 },
{ OrderID: 10254, CustomerID: 'CHOPS', Freight: 22.98 },
{ OrderID: 10255, CustomerID: 'RICSU', Freight: 148.33 },
{ OrderID: 10256, CustomerID: 'WELLI', Freight: 13.97 }
]
}
}
}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
</style>
`

```

### Defining Columns

The Grid has an option to define columns as directives. In these column directives, we have properties to customize columns. Let's check the properties used here:

- We have added [field](#) to map with a property name an array of JavaScript objects.
- We have added [headerText](#) to change the title of columns.
- We have used [textAlign](#) to change the alignment of columns. By default, columns will be left aligned. To change columns to right align, we need to define [textAlign](#) as **Right**.
- Also, we have used another useful property, [format](#). Using this, we can format number and date values to standard or custom formats.

Here, we have defined it for the conversion of numeric values to currency.

```

<ejs-grid :dataSource="data">
  <e-columns>
    <e-column field='OrderID' headerText='Order ID' textAlign='Right' width=90></e-column>
    <e-column field='CustomerID' headerText='Customer ID' width=120></e-column>
    <e-column field='Freight' headerText='Freight' textAlign='Right' format='C2' width=90></e-column>
  </e-columns>
</ejs-grid>
`

```

### Module injection

To create Vue Grid with additional features, inject the required modules. The following modules are used to extend Grid's basic functionality.

- **Page** - Inject this module to use paging feature.
- **Sort** - Inject this module to use sorting feature.
- **Filter** - Inject this module to use filtering feature.
- **Group** - Inject this module to use grouping feature.
- **Aggregate** - Inject this module to use aggregate feature.

Register the required array of modules under the key **grid** in the **provide** section.

Additional feature modules are available [here](#)

### Enable Paging

The paging feature enables users to view the Grid record in a paged view.

It can be enabled by setting [allowPaging](#) property to true. Also, need to inject the **Page** module in the **provide** section as follows.

If we didn't inject the **Page** module, then the pager will not be rendered in Grid. The pager can be customized using [pageSettings](#) property.

### ~/SRC/APP.VUE

```
<template>
  <div id="app">
    <ejs-grid :dataSource="data" :allowPaging="true"
    :pageSettings='pageSettings'>
      <e-columns>
        <e-column field='OrderID' headerText='Order ID'
        textAlign='Right' width=90></e-column>
        <e-column field='CustomerID' headerText='Customer ID'
        width=120></e-column>
        <e-column field='Freight' headerText='Freight' textAlign='Right'
        format='C2' width=90></e-column>
      </e-columns>
    </ejs-grid>
  </div>
</template>
<script>
import { GridComponent, ColumnDirective, ColumnsDirective, Page } from
"@syncfusion/ej2-vue-grids";
export default {
  components: {
    'ejs-grid': GridComponent,
    'e-column': ColumnDirective,
    'e-columns': ColumnsDirective
  },
  data() {
    return {
      data: [
        { OrderID: 10248, CustomerID: 'VINET', Freight: 32.38 },
        { OrderID: 10249, CustomerID: 'TOMSP', Freight: 11.61 },
        { OrderID: 10250, CustomerID: 'HANAR', Freight: 65.83 },
        { OrderID: 10251, CustomerID: 'VICTE', Freight: 41.34 },
```

```

        { OrderID: 10252, CustomerID: 'SUPRD', Freight: 51.3 },
        { OrderID: 10253, CustomerID: 'HANAR', Freight: 58.17 },
        { OrderID: 10254, CustomerID: 'CHOPS', Freight: 22.98 },
        { OrderID: 10255, CustomerID: 'RICSU', Freight: 148.33 },
        { OrderID: 10256, CustomerID: 'WELLI', Freight: 13.97 }
    ],
    pageSettings: { pageSize: 5 }
  };
},
provide: {
  grid: [Page]
}
}
</script>
<style>
  @import "../node_modules/@syncfusion/ej2-base/styles/material.css";
  @import "../node_modules/@syncfusion/ej2-buttons/styles/material.css";
  @import "../node_modules/@syncfusion/ej2-calendars/styles/material.css";
  @import "../node_modules/@syncfusion/ej2-dropdowns/styles/material.css";
  @import "../node_modules/@syncfusion/ej2-inputs/styles/material.css";
  @import "../node_modules/@syncfusion/ej2-navigations/styles/material.css";
  @import "../node_modules/@syncfusion/ej2-popups/styles/material.css";
  @import "../node_modules/@syncfusion/ej2-splitbuttons/styles/material.css";
  @import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/grid/getting-started/default-cs1" %}

### Enable Sorting

The sorting feature enables the user to order the records. It can be enabled by setting `[allowSorting]` (<https://ej2.syncfusion.com/vue/documentation/api/grid/#allowsorting>) property as true. Also, need to inject the **Sort** module in the **provide** section as follow. If we didn't inject the **Sort** module, then user not able to sort when click on headers. Sorting feature can be customized using [sortSettings](#) property.

### ~/SRC/APP.VUE

```

<template>
  <div id="app">
    <ejs-grid :dataSource="data" :allowPaging="true"
    :allowSorting='true' :pageSettings='pageSettings'>
      <e-columns>
        <e-column field='OrderID' headerText='Order ID'
        textAlign='Right' width=90></e-column>
        <e-column field='CustomerID' headerText='Customer ID'
        width=120></e-column>
        <e-column field='Freight' headerText='Freight' textAlign='Right'
        format='C2' width=90></e-column>
      </e-columns>
    </ejs-grid>
  </div>
</template>
<script>
import { GridComponent, ColumnDirective, ColumnsDirective, Page, Sort } from
"@syncfusion/ej2-vue-grids";

```



```

export default {
  components: {
    'ejs-grid': GridComponent,
    'e-column': ColumnDirective,
    'e-columns': ColumnsDirective
  },
  data() {
    return {
      data: [
        { OrderID: 10248, CustomerID: 'VINET', Freight: 32.38 },
        { OrderID: 10249, CustomerID: 'TOMSP', Freight: 11.61 },
        { OrderID: 10250, CustomerID: 'HANAR', Freight: 65.83 },
        { OrderID: 10251, CustomerID: 'VICTE', Freight: 41.34 },
        { OrderID: 10252, CustomerID: 'SUPRD', Freight: 51.3 },
        { OrderID: 10253, CustomerID: 'HANAR', Freight: 58.17 },
        { OrderID: 10254, CustomerID: 'CHOPS', Freight: 22.98 },
        { OrderID: 10255, CustomerID: 'RICSU', Freight: 148.33 },
        { OrderID: 10256, CustomerID: 'WELLI', Freight: 13.97 }
      ],
      pageSettings: { pageSize: 5 }
    };
  },
  provide: {
    grid: [Page, Sort]
  }
}
</script>
<style>
  @import "../node_modules/@syncfusion/ej2-base/styles/material.css";
  @import "../node_modules/@syncfusion/ej2-buttons/styles/material.css";
  @import "../node_modules/@syncfusion/ej2-calendars/styles/material.css";
  @import "../node_modules/@syncfusion/ej2-dropdowns/styles/material.css";
  @import "../node_modules/@syncfusion/ej2-inputs/styles/material.css";
  @import "../node_modules/@syncfusion/ej2-navigations/styles/material.css";
  @import "../node_modules/@syncfusion/ej2-popups/styles/material.css";
  @import "../node_modules/@syncfusion/ej2-splitbuttons/styles/material.css";
  @import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/grid/getting-started/default-cs2" %}

### Enable Filtering

The filtering feature enables the user to view the reduced amount of records based on filter criteria. It can be enabled by setting `[allowFiltering]`

https://ej2.syncfusion.com/vue/documentation/api/grid/#allowfiltering) property as true. Also, need to inject the **Filter** module in the **provide** section as follow. If we didn't inject the **Filter** module, then filter bar will not be rendered in Grid. Filtering feature can be customized using [filterSettings](#) property.

### ~/SRC/APP.VUE

```

<template>
  <div id="app">

```

```

    <ejs-grid :dataSource="data" :allowPaging="true"
:allowSorting='true' :allowFiltering='true' :pageSettings='pageSettings'>
      <e-columns>
        <e-column field='OrderID' headerText='Order ID'
textAlign='Right' width=90></e-column>
        <e-column field='CustomerID' headerText='Customer ID'
width=120></e-column>
        <e-column field='Freight' headerText='Freight' textAlign='Right'
format='C2' width=90></e-column>
      </e-columns>
    </ejs-grid>
  </div>
</template>
<script>
import { GridComponent, ColumnDirective, ColumnsDirective, Page, Sort,
Filter } from "@syncfusion/ej2-vue-grids";
export default {
  components: {
    'ejs-grid': GridComponent,
    'e-column': ColumnDirective,
    'e-columns': ColumnsDirective
  },
  data() {
    return {
      data: [
        { OrderID: 10248, CustomerID: 'VINET', Freight: 32.38 },
        { OrderID: 10249, CustomerID: 'TOMSP', Freight: 11.61 },
        { OrderID: 10250, CustomerID: 'HANAR', Freight: 65.83 },
        { OrderID: 10251, CustomerID: 'VICTE', Freight: 41.34 },
        { OrderID: 10252, CustomerID: 'SUPRD', Freight: 51.3 },
        { OrderID: 10253, CustomerID: 'HANAR', Freight: 58.17 },
        { OrderID: 10254, CustomerID: 'CHOPS', Freight: 22.98 },
        { OrderID: 10255, CustomerID: 'RICSU', Freight: 148.33 },
        { OrderID: 10256, CustomerID: 'WELLI', Freight: 13.97 }
      ],
      pageSettings: { pageSize: 5 }
    };
  },
  provide: {
    grid: [Page, Sort, Filter]
  }
}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-buttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-calendars/styles/material.css";
@import "../node_modules/@syncfusion/ej2-dropdowns/styles/material.css";
@import "../node_modules/@syncfusion/ej2-inputs/styles/material.css";
@import "../node_modules/@syncfusion/ej2-navigations/styles/material.css";
@import "../node_modules/@syncfusion/ej2-popups/styles/material.css";
@import "../node_modules/@syncfusion/ej2-
splitbuttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
</style>

```

```
{% previewsample "page.domainurl/code-snippet/grid/getting-started/default-cs3" %}
```

### Enable Grouping

The grouping feature enables users to view the Grid record in a grouped view. It can be enabled by setting `[allowGrouping]`(

<https://ej2.syncfusion.com/vue/documentation/api/grid/#allowgrouping>) property to true. Also, need to inject the **Group** module in the **provide** section as follow. If we didn't inject the **Group** module, then the group drop area will not be rendered in Grid. Grouping feature can be customized using [groupSettings](#) property.

### ~/SRC/APP.VUE

```
<template>
  <div id="app">
    <ejs-grid :dataSource="data" :allowPaging="true"
:allowSorting='true' :allowFiltering='true' :allowGrouping='true'
:pageSettings='pageSettings'>
      <e-columns>
        <e-column field='OrderID' headerText='Order ID'
textAlign='Right' width=90></e-column>
        <e-column field='CustomerID' headerText='Customer ID'
width=120></e-column>
        <e-column field='Freight' headerText='Freight' textAlign='Right'
format='C2' width=90></e-column>
      </e-columns>
    </ejs-grid>
  </div>
</template>
<script>
import { GridComponent, ColumnDirective, ColumnsDirective, Page, Sort,
Filter, Group } from "@syncfusion/ej2-vue-grids";
export default {
  components: {
    'ejs-grid': GridComponent,
    'e-column': ColumnDirective,
    'e-columns': ColumnsDirective
  },
  data() {
    return {
      data: [
        { OrderID: 10248, CustomerID: 'VINET', Freight: 32.38 },
        { OrderID: 10249, CustomerID: 'TOMSP', Freight: 11.61 },
        { OrderID: 10250, CustomerID: 'HANAR', Freight: 65.83 },
        { OrderID: 10251, CustomerID: 'VICTE', Freight: 41.34 },
        { OrderID: 10252, CustomerID: 'SUPRD', Freight: 51.3 },
        { OrderID: 10253, CustomerID: 'HANAR', Freight: 58.17 },
        { OrderID: 10254, CustomerID: 'CHOPS', Freight: 22.98 },
        { OrderID: 10255, CustomerID: 'RICSU', Freight: 148.33 },
        { OrderID: 10256, CustomerID: 'WELLI', Freight: 13.97 }
      ],
      pageSettings: { pageSize: 5 }
    };
  },
  provide: {
    grid: [Page, Sort, Filter, Group]
  }
}
```

```

}
</script>
<style>
  @import "../node_modules/@syncfusion/ej2-base/styles/material.css";
  @import "../node_modules/@syncfusion/ej2-buttons/styles/material.css";
  @import "../node_modules/@syncfusion/ej2-calendars/styles/material.css";
  @import "../node_modules/@syncfusion/ej2-dropdowns/styles/material.css";
  @import "../node_modules/@syncfusion/ej2-inputs/styles/material.css";
  @import "../node_modules/@syncfusion/ej2-navigations/styles/material.css";
  @import "../node_modules/@syncfusion/ej2-popups/styles/material.css";
  @import "../node_modules/@syncfusion/ej2-
splitbuttons/styles/material.css";
  @import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/grid/getting-started/default-cs4" %}

### Enable Aggregate

The Aggregate feature enables users to view the aggregates of Grid records. It can be enabled by configured through **e-aggregates** directive. The **field** and **type** are the minimum properties required to represent an aggregate column. Also, need to inject the **Aggregate** module in the **provide** section as follow. If we didn't inject the **Aggregate** module, then the footer table will not be rendered in Grid. Check [aggregate](#) to know more about its configuration.

To avoid runtime-compilation overhead and CSP restrictions, install the template compiler using the following code:

```
`bash
```

```
npm i vue-template-compiler
```

```
,
```

Additionally, you need to set **runtimeCompiler** to true in the Vue.js CLI configuration, as shown below:

```
`bash
```

```
const { defineConfig } = require('@vue/cli-service')
```

```
module.exports = defineConfig({
```

```
  transpileDependencies: true,
```

```
  runtimeCompiler: true,
```

```
})
```

```
,
```

### ~/SRC/APP.VUE

```

{% raw %}
<template>
<div id="app">
<ejs-grid :dataSource="data" :allowPaging="true" :allowSorting='true'
:allowFiltering='true' :allowGrouping='true' :pageSettings='pageSettings' >
<e-columns>
<e-column field='OrderID' headerText='Order ID' textAlign='Right'
width=90></e-column>

```

```

<e-column field='CustomerID' headerText='Customer ID' width=120></e-column>
<e-column field='Freight' headerText='Freight' textAlign='Right' format='C2'
width=90></e-column>
</e-columns>
<e-aggregates>
<e-aggregate>
<e-columns>
<e-column type="Sum" field="Freight" format="C2"
:footerTemplate='footerSum'></e-column>
</e-columns>
</e-aggregate>
</e-aggregates>
</ejs-grid>
</div>
</template>
<script>
import Vue from 'vue';
import { GridComponent, ColumnDirective, ColumnsDirective,
AggregateDirective, AggregatesDirective, Page, Sort, Filter, Group,
Aggregate } from "@syncfusion/ej2-vue-grids";
export default {
  components: {
    'ejs-grid': GridComponent,
    'e-column': ColumnDirective,
    'e-columns': ColumnsDirective,
    'e-aggregates': AggregatesDirective,
    'e-aggregate': AggregateDirective,
  },
  data() {
    return {
      data: [
        { OrderID: 10248, CustomerID: 'VINET', Freight: 32.38 },
        { OrderID: 10249, CustomerID: 'TOMSP', Freight: 11.61 },
        { OrderID: 10250, CustomerID: 'HANAR', Freight: 65.83 },
        { OrderID: 10251, CustomerID: 'VICTE', Freight: 41.34 },
        { OrderID: 10252, CustomerID: 'SUPRD', Freight: 51.3 },
        { OrderID: 10253, CustomerID: 'HANAR', Freight: 58.17 },
        { OrderID: 10254, CustomerID: 'CHOPS', Freight: 22.98 },
        { OrderID: 10255, CustomerID: 'RICSU', Freight: 148.33 },
        { OrderID: 10256, CustomerID: 'WELLI', Freight: 13.97 }
      ],
      pageSettings: { pageSize: 5 },
      footerSum: function () {
        return { template : Vue.component('sumTemplate', {
          template: `<span>Sum: {{data.Sum}}</span>`,
          data () {return { data: {data: {}}};}
        })
      }
    };
  },
  provide: {
    grid: [Page, Sort, Filter, Group, Aggregate]
  }
}
</script>
<style>

```

```
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-buttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-calendars/styles/material.css";
@import "../node_modules/@syncfusion/ej2-dropdowns/styles/material.css";
@import "../node_modules/@syncfusion/ej2-inputs/styles/material.css";
@import "../node_modules/@syncfusion/ej2-navigations/styles/material.css";
@import "../node_modules/@syncfusion/ej2-popups/styles/material.css";
@import "../node_modules/@syncfusion/ej2-splitbuttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
</style>
{% endraw %}
```

```
{% previewsample "page.domainurl/code-snippet/grid/getting-started/default-cs5" %}
```

### Run the project

To run the project, use the following command:

```
`bash
```

```
npm run serve
```

```
,
```

or

```
`bash
```

```
yarn run serve
```

```
,
```

### ~/SRC/APP.VUE

```
{% raw %}
<template>
<div id="app">
<ejs-grid :dataSource="data" :allowPaging="true" :allowSorting='true'
:allowFiltering='true' :allowGrouping='true' :pageSettings='pageSettings' >
<e-columns>
<e-column field='OrderID' headerText='Order ID' textAlign='Right'
width=90></e-column>
<e-column field='CustomerID' headerText='Customer ID' width=120></e-column>
<e-column field='Freight' headerText='Freight' textAlign='Right' format='C2'
width=90></e-column>
</e-columns>
<e-aggregates>
<e-aggregate>
<e-columns>
<e-column type="Sum" field="Freight" format="C2"
:footerTemplate='footerSum'></e-column>
</e-columns>
</e-aggregate>
</e-aggregates>
</ejs-grid>
</div>
</template>
<script>
import Vue from 'vue';
```

```

import { GridComponent, ColumnDirective, ColumnsDirective,
AggregateDirective, AggregatesDirective, Page, Sort, Filter, Group,
Aggregate } from "@syncfusion/ej2-vue-grids";
export default {
  components: {
    'ejs-grid': GridComponent,
    'e-column': ColumnDirective,
    'e-columns': ColumnsDirective,
    'e-aggregates': AggregatesDirective,
    'e-aggregate': AggregateDirective,
  },
  data() {
    return {
      data: [
        { OrderID: 10248, CustomerID: 'VINET', Freight: 32.38 },
        { OrderID: 10249, CustomerID: 'TOMSP', Freight: 11.61 },
        { OrderID: 10250, CustomerID: 'HANAR', Freight: 65.83 },
        { OrderID: 10251, CustomerID: 'VICTE', Freight: 41.34 },
        { OrderID: 10252, CustomerID: 'SUPRD', Freight: 51.3 },
        { OrderID: 10253, CustomerID: 'HANAR', Freight: 58.17 },
        { OrderID: 10254, CustomerID: 'CHOPS', Freight: 22.98 },
        { OrderID: 10255, CustomerID: 'RICSU', Freight: 148.33 },
        { OrderID: 10256, CustomerID: 'WELLI', Freight: 13.97 }
      ],
      pageSettings: { pageSize: 5 },
      footerSum: function () {
        return { template : Vue.component('sumTemplate', {
          template: `<span>Sum: {{data.Sum}}</span>`,
          data () {return { data: {data: {}}};}
        })
      }
    };
  },
  provide: {
    grid: [Page, Sort, Filter, Group, Aggregate]
  }
}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-buttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-calendars/styles/material.css";
@import "../node_modules/@syncfusion/ej2-dropdowns/styles/material.css";
@import "../node_modules/@syncfusion/ej2-inputs/styles/material.css";
@import "../node_modules/@syncfusion/ej2-navigations/styles/material.css";
@import "../node_modules/@syncfusion/ej2-popups/styles/material.css";
@import "../node_modules/@syncfusion/ej2-splitbuttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
</style>
{% endraw %}

```

{% previewsample "page.domainurl/code-snippet/grid/getting-started/default-cs6" %}

You can refer to our [Vue Grid](#) feature tour page for its groundbreaking feature representations. You can also explore our [Vue Grid example](#) that shows how to render the Grid in Vue.

## See Also

- [Testing the Vue Grid](#)
- [Switching themes programmatically in Vue Grid](#)

## Getting Started with the Vue Grid Component in Vue 3

This article provides a step-by-step guide for setting up a [Vite](#) project with a JavaScript environment and integrating the Syncfusion Vue Grid component using the [Composition API](#) / [Options API](#).

The **Composition API** is a new feature introduced in Vue.js 3 that provides an alternative way to organize and reuse component logic. It allows developers to write components as functions that use smaller, reusable functions called composition functions to manage their properties and behavior.

The **Options API** is the traditional way of writing Vue.js components, where the component logic is organized into a series of options that define the component's properties and behavior. These options include data, methods, computed properties, watchers, lifecycle hooks, and more.

### Prerequisites

[System requirements for Syncfusion Vue UI components](#)

### Set up the Vite project

A recommended approach for beginning with Vue is to scaffold a project using [Vite](#). To create a new Vite project, use one of the commands that are specific to either NPM or Yarn.

```
`bash
npm create vite@latest
`
```

or

```
`bash
yarn create vite
`
```

Using one of the above commands will lead you to set up additional configurations for the project as below:

1. Define the project name: We can specify the name of the project directly. Let's specify the name of the project as **my-project** for this article.

```
`bash
? Project name: » my-project
`
```

2. Select **Vue** as the framework. It will create a Vue 3 project.

```
`bash
? Select a framework: » - Use arrow-keys. Return to submit.
Vanilla
```



Vue

React

Preact

Lit

Svelte

Others

,

3. Choose **JavaScript** as the framework variant to build this Vite project using JavaScript and Vue.

```
`bash
```

? Select a variant: » - Use arrow-keys. Return to submit.

JavaScript

TypeScript

Customize with create-vue ↗

Nuxt ↗

,

4. Upon completing the aforementioned steps to create the **my-project**, run the following command to install its dependencies:

```
`bash
```

```
cd my-project
```

```
npm install
```

,

or

```
`bash
```

```
cd my-project
```

```
yarn install
```

,

Now that **my-project** is ready to run with default settings, let's add Syncfusion components to the project.

#### Add Syncfusion Vue packages

Syncfusion Vue component packages are available at [npmjs.com](https://www.npmjs.com). To use Syncfusion Vue components in the project, install the corresponding npm package.

This article uses the [Vue Grid component](#) as an example. To use the Vue Grid component in the project, the `@syncfusion/ej2-vue-grids` package needs to be installed using the following command:

```
`bash
```

```
npm install @syncfusion/ej2-vue-grids --save
```

```
,
```

```
or
```

```
`bash
```

```
yarn add @syncfusion/ej2-vue-grids
```

```
,
```

### Import Syncfusion CSS styles

You can import themes for the Syncfusion Vue component in various ways, such as using CSS or SASS styles from npm packages, CDN, [CRG](#) and [Theme Studio](#). Refer to [themes topic](#) to know more about built-in themes and different ways to refer to themes in a Vue project.

In this article, **Material** theme is applied using CSS styles, which are available in installed packages. The necessary **Material** CSS styles for the Grid component and its dependents were imported into the `<style>` section of **src/App.vue** file.

#### ~/SRC/APP.VUE

```
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-buttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-calendars/styles/material.css";
@import "../node_modules/@syncfusion/ej2-dropdowns/styles/material.css";
@import "../node_modules/@syncfusion/ej2-inputs/styles/material.css";
@import "../node_modules/@syncfusion/ej2-navigations/styles/material.css";
@import "../node_modules/@syncfusion/ej2-popups/styles/material.css";
@import "../node_modules/@syncfusion/ej2-splitbuttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
</style>
```

The order of importing CSS styles should be in line with its dependency graph.

### Add Syncfusion Vue component

Follow the below steps to add the Vue Grid component using **Composition API** or **Options API**:

1.First, import and register the Grid component and its child directives in the **script** section of the **src/App.vue** file. If you are using the **Composition API**, you should add the **setup** attribute to the **script** tag to indicate that Vue will be using the **Composition API**.

#### COMPOSITION API (~/SRC/APP.VUE)

```
<script setup>
import { GridComponent as EjsGrid, ColumnsDirective as EColumns,
ColumnDirective as EColumn } from '@syncfusion/ej2-vue-grids';
</script>
```

#### OPTIONS API (~/SRC/APP.VUE)

```
<script>
import { GridComponent, ColumnsDirective, ColumnDirective } from
'@syncfusion/ej2-vue-grids';
```

```
//Component registration
export default {
  name: "App",
  components: {
    'ejs-grid': GridComponent,
    'e-columns': ColumnsDirective,
    'e-column': ColumnDirective
  }
}
</script>
```

2. In the **template** section, define the Grid component with the [dataSource](#) property and column definitions.

#### ~/SRC/APP.VUE

```
<template>
<ejs-grid :dataSource='data'>
<e-columns>
<e-column field='OrderID' width='100' textAlign="Right"></e-column>
<e-column field='CustomerID' width='100'></e-column>
<e-column field='EmployeeID' width='100' textAlign="Right"></e-column>
<e-column field='Freight' width='100' format="C2" textAlign="Right"></e-
column>
<e-column field='ShipCountry' width='100'></e-column>
</e-columns>
</ejs-grid>
</template>
```

3. Declare the values for the **dataSource** property in the **script** section.

#### COMPOSITION API (~/SRC/APP.VUE)

```
<script setup>
const data = [
  {
    OrderID: 10248, CustomerID: 'VINET', EmployeeID: 5, ShipCountry: 'France',
    Freight: 32.38
  },
  {
    OrderID: 10249, CustomerID: 'TOMSP', EmployeeID: 6, ShipCountry: 'Germany',
    Freight: 11.61
  },
  {
    OrderID: 10250, CustomerID: 'HANAR', EmployeeID: 4, ShipCountry: 'Brazil',
    Freight: 65.83
  }
];
</script>
```

#### OPTIONS API (~/SRC/APP.VUE)

```
<script>
data() {
  return {
```

```

data:[
  {
    OrderID: 10248, CustomerID: 'VINET', EmployeeID: 5, ShipCountry: 'France',
    Freight: 32.38
  },
  {
    OrderID: 10249, CustomerID: 'TOMSP', EmployeeID: 6, ShipCountry: 'Germany',
    Freight: 11.61
  },
  {
    OrderID: 10250, CustomerID: 'HANAR', EmployeeID: 4, ShipCountry: 'Brazil',
    Freight: 65.83
  }
];
}
</script>

```

Here is the summarized code for the above steps in the **src/App.vue** file:

#### **COMPOSITION API (~ /SRC/APP.VUE)**

```

<template>
<ejs-grid :dataSource='data'>
<e-columns>
<e-column field='OrderID' width='100' textAlign="Right"></e-column>
<e-column field='CustomerID' width='100'></e-column>
<e-column field='EmployeeID' width='100' textAlign="Right"></e-column>
<e-column field='Freight' width='100' format="C2" textAlign="Right"></e-
column>
<e-column field='ShipCountry' width='100'></e-column>
</e-columns>
</ejs-grid>
</template>
<script setup>
import { GridComponent as EjsGrid, ColumnsDirective as EColumns,
ColumnDirective as EColumn } from '@syncfusion/ej2-vue-grids';
const data = [
  {
    OrderID: 10248, CustomerID: 'VINET', EmployeeID: 5, ShipCountry: 'France',
    Freight: 32.38
  },
  {
    OrderID: 10249, CustomerID: 'TOMSP', EmployeeID: 6, ShipCountry: 'Germany',
    Freight: 11.61
  },
  {
    OrderID: 10250, CustomerID: 'HANAR', EmployeeID: 4, ShipCountry: 'Brazil',
    Freight: 65.83
  }
];
</script>
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-buttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-calendars/styles/material.css";

```

```
@import "../node_modules/@syncfusion/ej2-dropdowns/styles/material.css";
@import "../node_modules/@syncfusion/ej2-inputs/styles/material.css";
@import "../node_modules/@syncfusion/ej2-navigations/styles/material.css";
@import "../node_modules/@syncfusion/ej2-popups/styles/material.css";
@import "../node_modules/@syncfusion/ej2-splitbuttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
</style>
```

### OPTIONS API (~SRC/APP.VUE)

```
<template>
<ejs-grid :dataSource='data'>
<e-columns>
<e-column field='OrderID' width='100' textAlign="Right"></e-column>
<e-column field='CustomerID' width='100'></e-column>
<e-column field='EmployeeID' width='100' textAlign="Right"></e-column>
<e-column field='Freight' width='100' format="C2" textAlign="Right"></e-column>
<e-column field='ShipCountry' width='100'></e-column>
</e-columns>
</ejs-grid>
</template>
<script>
import { GridComponent, ColumnsDirective, ColumnDirective } from
'@syncfusion/ej2-vue-grids';
// Component registration
export default {
name: "App",
// Declaring component and its directives
components: {
'ejs-grid': GridComponent,
'e-columns': ColumnsDirective,
'e-column': ColumnDirective
},
// Bound properties declarations
data() {
return {
data:[
{
OrderID: 10248, CustomerID: 'VINET', EmployeeID: 5, ShipCountry: 'France',
Freight: 32.38
},
{
OrderID: 10249, CustomerID: 'TOMSP', EmployeeID: 6, ShipCountry: 'Germany',
Freight: 11.61
},
{
OrderID: 10250, CustomerID: 'HANAR', EmployeeID: 4, ShipCountry: 'Brazil',
Freight: 65.83
}
],
};
}
};
</script>
<style>
```

```
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-buttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-calendars/styles/material.css";
@import "../node_modules/@syncfusion/ej2-dropdowns/styles/material.css";
@import "../node_modules/@syncfusion/ej2-inputs/styles/material.css";
@import "../node_modules/@syncfusion/ej2-navigations/styles/material.css";
@import "../node_modules/@syncfusion/ej2-popups/styles/material.css";
@import "../node_modules/@syncfusion/ej2-splitbuttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
</style>
```

### Run the project

To run the project, use the following command:

```
`bash
```

```
npm run dev
```

```
,
```

or

```
`bash
```

```
yarn run dev
```

```
,
```

The output will appear as follows:

OrderID	CustomerID	EmployeeID	Freight	ShipCountry
10248	VINET	5	\$32.38	France
10249	TOMSP	6	\$11.61	Germany
10250	HANAR	4	\$65.83	Brazil

**Sample:** [vue-3-grid-getting-started](#).

For migrating from Vue 2 to Vue 3, refer to the [migration](#) documentation.

See also

- [Getting Started with Vue UI Components using Composition API and TypeScript](#)
- [Getting Started with Vue UI Components using Options API and TypeScript](#)

### Module in Vue Grid component

The following modules should be injected to extend grid's functionality.

The available grid modules are

Module	Description
--------	-------------

-----	-----
-------	-------

<a href="#">Page</a>	Inject this module to use paging feature.
----------------------	-------------------------------------------

| [Sort](#) | Inject this module to use sorting feature. |

| [Filter](#) | Inject this module to use filtering feature. |

| [Group](#) | Inject this module to use grouping feature |

| [Edit](#) | Inject this module is use editing feature. |

| [Aggregate](#) | Inject this module to use aggregate feature. |

| [ColumnChooser](#) | Inject this module to use column chooser feature. |

| [ColumnMenu](#) | Inject this module to use column menu feature. |

| [CommandColumn](#) | Inject this module to use command column feature. |

| [ContextMenu](#) | Inject this module to use context menu feature. |

| [DetailRow](#) | Inject this module to use detail template feature. |

| [ForeignKey](#) | Inject this module to use foreign key feature. |

| [Freeze](#) | Inject this module to use frozen rows and columns feature. |

| [Resize](#) | Inject this module to use resize feature. |

| [Reorder](#) | Inject this module to use reorder feature. |

| [RowDD](#) | Inject this module to use row drag and drop feature. |

| [Search](#) | Inject this module to use search feature and this is a default injected module. |

| [Selection](#) | Inject this module to use selection feature and this is a default injected module. |

| [Scroll](#) | Inject this module to use scrolling feature and this is a default injected module. |

| [Print](#) | Inject this module to use to use print feature and this is a default injected module. |

| [Toolbar](#) | Inject this module to use toolbar feature. |

| [VirtualScroll](#) | Inject this module to use virtual scroll feature. |

| [ExcelExport](#) | Inject this module to use excel export feature. |

| [PdfExport](#) | Inject this module to use PDF export feature. |

These modules should be injected into the `provide` section and use `grid` as a key of the object.

## Data Binding

### Data binding in Vue Grid component

The Grid uses `DataManager` which supports both RESTful JSON data services binding and local JavaScript object array binding. The `dataSource` property can be assigned either with the instance of [Link to the Video](#) or JavaScript object array collection. It supports two kinds of data binding methods:

- Local data
- Remote data

To learn about Grid data binding quickly, you can check on this video:

*Sending additional parameters to the server*

To add a custom parameter to the data request, use the `addParams` method of `Query` class. Assign the `Query` object with additional parameters to the grid `query` property.

**APP.VUE**

```
<template>
  <div id="app">
    <ejs-grid :dataSource="data" :query='query'>
      <e-columns>
        <e-column field='OrderID' headerText='Order ID'
textAlign='Right' width=90></e-column>
        <e-column field='CustomerID' headerText='Customer ID'
width=120></e-column>
        <e-column field='Freight' headerText='Freight' textAlign='Right'
format='C2' width=90></e-column>
        <e-column field='OrderDate' headerText='Order Date'
textAlign='Right' format='yMd' type='date' width=120></e-column>
      </e-columns>
    </ejs-grid>
  </div>
</template>
<script>
import Vue from "vue";
import { GridPlugin, Page } from "@syncfusion/ej2-vue-grids";
import { DataManager, ODataAdaptor, Query } from "@syncfusion/ej2-data";
Vue.use(GridPlugin);
export default {
  data() {
    let SERVICE_URI =
      "https://services.syncfusion.com/js/production/api/Orders";
    return {
      data: new DataManager({
        url: SERVICE_URI,
        adaptor: new ODataAdaptor()
      }),
      pageSettings: {pageSize: 7},
      query: new Query().addParams('ej2grid', 'true');
    };
  },
  provide: {
    grid: [Page]
  }
}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
</style>
```

{% previewsample "page.domainurl/code-snippet/grid/databind/remote-cs1" %}

The parameters added using the `query` property will be sent along with the data request for every grid action.



*Handling HTTP error*

During server interaction from the grid, some server-side exceptions may occur, and you can acquire those error messages or exception details in client-side using the [actionFailure](#) event.

The argument passed to the [actionFailure](#) Grid event contains the error details returned from the server.

,

```
<template>
<div id="app">
<ejs-grid :dataSource="data" :actionFailure='actionFailure'>
<e-columns>
<e-column field='OrderID' headerText='Order ID' textAlign='Right' width=90></e-column>
<e-column field='CustomerID' headerText='Customer ID' width=120></e-column>
<e-column field='Freight' headerText='Freight' textAlign='Right' format='C2' width=90></e-column>
<e-column field='OrderDate' headerText='Order Date' textAlign='Right' format='yMd' type='date'
width=120></e-column>
</e-columns>
</ejs-grid>
</div>
</template>
<script>
import Vue from "vue";
import { GridPlugin } from "@syncfusion/ej2-vue-grids";
import { DataManager, ODataAdaptor } from "@syncfusion/ej2-data";
Vue.use(GridPlugin);
export default Vue.extend({
data() {
return {
data: new DataManager({
url: 'http://some.com/invalidUrl',
adaptor: new ODataAdaptor()
})
};
},
methods: {
actionFailure: function() {
```

```

alert('Server exception: 404 Not found');
}
}
});
</script>
<style>
@import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
</style>
`

```

The [actionFailure](#) event will be triggered not only for the server errors, but also when there is an exception while processing the grid actions.

#### *Binding with ajax*

You can use Grid [dataSource](#) property to bind the datasource to Grid from external ajax request. In the below code we have fetched the datasource from the server with the help of ajax request and provided that to [dataSource](#) property by using `onSuccess` event of the ajax.

```

<template>
<div id="app">
<ejs-button v-on:click.native="btnClick">Play</ejs-button>
<ejs-grid>
<e-columns>
<e-column field='OrderID' headerText='Order ID' textAlign='Right' width=120 ></e-column>
<e-column field='CustomerID' headerText='Customer ID' textAlign='Right' width=120 ></e-column>
<e-column field='EmployeeID' headerText='Employee ID' textAlign='Right' width=120 ></e-column>
<e-column field='ShipCountry' headerText='Ship Country' textAlign='Right' width=120 ></e-column>
</e-columns>
</ejs-grid>
</div>
</template>
<script>
import Vue from "vue";
import { GridPlugin } from "@syncfusion/ej2-vue-grids";
import { ButtonPlugin } from "@syncfusion/ej2-vue-buttons";
import { Ajax } from '@syncfusion/ej2-base';

```

```

Vue.use(GridPlugin);
Vue.use(ButtonPlugin);
export default Vue.extend({
  data() {
    return {
      data: {}
    };
  },
  methods:{
    btnClick: function (args){
      var grid = document.getElementsByClassName("e-grid")[0].ej2_instances[0]; // Grid instance
      var ajax = new Ajax("https://ej2services.syncfusion.com/production/web-services/api/Orders", "GET");
      ajax.send();
      ajax.onSuccess = function (result) {
        grid.dataSource = JSON.parse(result);
      };
    }
  }
});
</script>
<style>
@import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
</style>

```

### Custom binding

It is possible to handle data processing externally and bind the result to the grid. This help you to provide your own custom data logic. Grid expects an object as the result of the custom logic and the emitted value should be an object with properties `result` and `count`.

In this context, we are going to use `Ajax` from our `@syncfusion/ej2-base` library for handling remote interaction, you can choose any HTTP client as per your choice.

```

<template>
<div id="app">
<ejs-grid :dataSource='data' :allowPaging='true' :pageSettings='pageOptions' :allowSorting='true'
:allowGrouping='true' :dataStateChange='dataStateChange'>

```

```
<e-columns>
<e-column field= "OrderID" headerText="Order ID" width="130" textAlign='Right' ></e-column>
<e-column field= "CustomerID" headerText="Customer Name" width="150"></e-column>
<e-column field= "ShipName" headerText="Ship Name" width="200"></e-column>
<e-column field= "ShipCity" headerText="Ship City" width="150"></e-column>
</e-columns>
</ejs-grid>
</div>
</template>
<script>
import Vue from "vue";
import { GridPlugin, DataStateChangeEventArgs, Sorts, Sort, Group, Page, DataResult } from
"@syncfusion/ej2-vue-grids";
import { Ajax } from '@syncfusion/ej2-base';
Vue.use(GridPlugin);
export default Vue.extend({
  data() {
    return {
      data: {},
      pageOptions: { pageSize: 10, pageCount: 4 },
      orderService: new OrderService()
    };
  },
  mounted() {
    let state = { skip: 0, take: 10 };
    this.dataStateChange(state);
  },
  methods:{
    dataStateChange: function (state) {
      this.orderService.execute(state).then(( gridData ) => this.data = gridData );
    }
  },
  provide: {
```

```

grid: [Sort, Group, Page]
}
});
export class OrderService {
  public ajax: Ajax = new Ajax({
    type: 'GET', mode: true,
    onFailure: (e: Error) => { return false; }
  });
  private BASE_URL: string = 'https://services.syncfusion.com/js/production/api/Orders';
  public execute(state: DataStateChangeEventArgs): Promise<DataResult> {
    return this.getData(state);
  }
  private getData(state: DataStateChangeEventArgs): Promise<DataResult> {
    const pageQuery = `skip=${state.skip}&stop=${state.take}`;
    let sortQuery: string = '';
    if ((state.sorted || []).length) {
      sortQuery = `orderby= ` + (<any> state).sorted.map((obj: Sorts) => {
        return obj.direction === 'descending' ? `${obj.name} desc` : obj.name;
      }).reverse().join(',');
    }
    this.ajax.url =
      `${this.BASE_URL}?${pageQuery}${sortQuery}&$inlinecount=allpages&$format=json`;
    return this.ajax.send().then((response: any) => {
      let data: any = JSON.parse(response);
      return <DataResult> { result: data['d']['results'], count: parseInt(data['d']['count'], 10) };
    });
  }
}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
</style>
`

```

*Handling Grid actions*

For grid actions such as **paging**, **grouping**, **sorting** etc, the **dataStateChange** event will be invoked. You have to query and resolve data using **AJAX** in this event based on the state arguments.

```

<template>
<div id="app">
<ejs-grid :dataSource='data' :allowPaging='true' :pageSettings='pageOptions' :allowSorting='true'
:allowGrouping='true' :dataStateChange='dataStateChange'>
<e-columns>
<e-column field= "OrderID" headerText="Order ID" width="130" textAlign='Right' ></e-column>
<e-column field= "CustomerID" headerText="Customer Name" width="150"></e-column>
<e-column field= "ShipName" headerText="Ship Name" width="200"></e-column>
<e-column field= "ShipCity" headerText="Ship City" width="150"></e-column>
</e-columns>
</ejs-grid>
</div>
</template>
<script>
import Vue from "vue";

import { GridPlugin, DataStateChangeEventArgs, Sorts, Sort, Group, Page, DataResult } from
"@syncfusion/ej2-vue-grids";

import { Ajax } from '@syncfusion/ej2-base';

Vue.use(GridPlugin);

export default Vue.extend({
data() {
return {
data: {},
pageOptions: { pageSize: 10, pageCount: 4 },
orderService: new OrderService()
};
},
mounted() {
let state = { skip: 0, take: 10 };
this.dataStateChange(state);

```

```

},
methods:{
  dataStateChange: function (state) {
    this.orderService.execute(state).then(( gridData ) => this.data = gridData );
  }
},
provide: {
  grid: [Sort, Group, Page]
}
});

export class OrderService {
  public ajax: Ajax = new Ajax({
    type: 'GET', mode: true,
    onFailure: (e: Error) => { return false; }
  });

  private BASE_URL: string = 'https://services.syncfusion.com/js/production/api/Orders';
  public execute(state: DataStateChangeEventArgs): Promise<DataResult> {
    return this.getData(state);
  }

  private getData(state: DataStateChangeEventArgs): Promise<DataResult> {
    const pageQuery = `&$skip=${state.skip}&$top=${state.take}`;
    let sortQuery: string = '';
    if ((state.sorted || []).length) {
      sortQuery = `&$orderby= ` + (state.sorted.map((obj: Sorts) => {
        return obj.direction === 'descending' ? `${obj.name} desc` : obj.name;
      })).reverse().join(',');
    }

    this.ajax.url =
      `${this.BASE_URL}?${pageQuery}${sortQuery}&$inlinecount=allpages&$format=json`;
    return this.ajax.send().then((response: any) => {
      let data: any = JSON.parse(response);
      return { result: data['d']['results'], count: parseInt(data['d']['count'], 10) };
    });
  }
}

```

```

}
}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
</style>
`

```

### Perform CRUD operations

The `dataSourceChanged` event will be triggered for updating the grid data. You can perform the save operation based on the event arguments and you need to call the `endEdit` method to indicate the completion of save operation.

```

<template>
<div id="app">
<ejs-grid :dataSource='data' :allowPaging='true' :pageSettings='pageOptions' :allowSorting='true'
:allowGrouping='true' :dataStateChange='dataStateChange' :editSettings='editSettings'
:toolbar='toolbar'>
<e-columns>
<e-column field= "OrderID" headerText="Order ID" :isPrimaryKey='true' width="130" textAlign='Right'
></e-column>
<e-column field= "CustomerID" headerText="Customer Name" width="150"></e-column>
<e-column field= "ShipName" headerText="Ship Name" width="200"></e-column>
<e-column field= "ShipCity" headerText="Ship City" width="150"></e-column>
</e-columns>
</ejs-grid>
</div>
</template>
<script>
import Vue from "vue";

import { GridPlugin, DataStateChangeEventArgs, Sorts, Sort, Group, Page, DataResult, Toolbar, Edit }
from "@syncfusion/ej2-vue-grids";

import { OrderService } from "../order-service";

Vue.use(GridPlugin);

export default Vue.extend({
data() {

```



```
return {
  data: {},
  editSettings: { allowEditing: true, allowAdding: true, allowDeleting: true },
  toolbar: ['Add', 'Edit', 'Delete', 'Update', 'Cancel'],
  pageOptions: { pageSize: 10, pageCount: 4 },
  orderService: new OrderService()
};
},
mounted() {
  let state = { skip: 0, take: 10 };
  this.dataStateChange(state);
},
methods:{
  dataStateChange: function (state) {
    this.orderService.execute(state).then(( gridData ) => this.data = gridData );
  },
  dataSourceChanged: function (state) {
    if (state.action === 'add') {
      this.orderService.addRecord(state).subscribe(() => state.endEdit());
    } else if (state.action === 'edit') {
      this.orderService.updateRecord(state).subscribe(() => state.endEdit());
    } else if (state.requestType === 'delete') {
      this.orderService.deleteRecord(state).subscribe(() => state.endEdit());
    }
  }
},
provide: {
  grid: [Sort, Group, Page, Edit, Toolbar]
}
});
</script>
<style>
@import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
```

```
</style>
```

```
,
```

### Calculate aggregates

The footer aggregate values should be calculated and send along with the `dataSource` property as follows. The aggregate property of the data source should contain the aggregate value assigned to the property named in the `field – type` format. For example, the `Sum` aggregate value for the `Freight` field should be assigned to the property named as `Freight - sum`.

```
,
```

```
{
  result: [{..}, {..}, {..}, ...],
  count: 830,
  aggregates: { 'Freight - sum' : 450,'EmployeeID - min': 1 }
}
```

```
,
```

The group footer and caption aggregate values will be calculated by the grid itself.

### Provide excel filter data source

The `dataStateChange` event will be triggered with appropriate arguments when the excel filter requests the filter choice data source. You need to resolve the excel filter data source using the `dataSource` resolver function from the state argument as follows.

```
,
```

```
<template>
<div id="app">
  <ejs-grid :dataSource='data' :allowPaging='true' :pageSettings='pageOptions' :allowSorting='true'
  :allowGrouping='true' :dataStateChange='dataStateChange' :filterSettings='filterOptions'>
    <e-columns>
      <e-column field= "OrderID" headerText="Order ID" width="130" textAlign='Right' ></e-column>
      <e-column field= "CustomerID" headerText="Customer Name" width="150"></e-column>
      <e-column field= "ShipName" headerText="Ship Name" width="200"></e-column>
      <e-column field= "ShipCity" headerText="Ship City" width="150"></e-column>
    </e-columns>
  </ejs-grid>
</div>
</template>
<script>
import Vue from "vue";
```

```
import { GridPlugin, DataStateChangeEventArgs, Sorts, Sort, Group, Page, DataResult, Filter } from
"@syncfusion/ej2-vue-grids";
import { Ajax } from '@syncfusion/ej2-base';
Vue.use(GridPlugin);
export default Vue.extend({
  data() {
    return {
      data: {},
      filterOptions: { type: 'Excel' },
      pageOptions: { pageSize: 10, pageCount: 4 },
      orderService: new OrderService()
    };
  },
  mounted() {
    let state = { skip: 0, take: 10 };
    this.dataStateChange(state);
  },
  methods:{
    dataStateChange: function (state) {
      if (state.action.requestType === "filterchoicerequest" || state.action.requestType
        === "filtersearchbegin") {
        this.orderService.execute(state).then((e) => state.dataSource(e));
      } else {
        this.orderService.execute(state).then(( gridData ) => {this.grid.dataSource = gridData} );
      }
    },
  },
  provide: {
    grid: [Sort, Group, Page, Filter]
  }
});
</script>
<style>
@import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
```

</style>

You can find the code at our [GitHub repository](#)

*See Also*

- [How to bind SQL Server data in Vue DataGrid using SqlClient data provider](#)

### Local data in Vue Grid component

To bind local data to the grid, you can assign a JavaScript object array to the `dataSource` property. The local data source can also be provided as an instance of the `DataManager`.

### APP.VUE

```
<template>
  <div id="app">
    <ejs-grid :dataSource="data" height='315px' >
      <e-columns>
        <e-column field='OrderID' headerText='Order ID'
textAlign='Right' width=90></e-column>
        <e-column field='CustomerID' headerText='Customer ID'
width=120></e-column>
        <e-column field='Freight' headerText='Freight' textAlign='Right'
format='C2' width=90></e-column>
        <e-column field='OrderDate' headerText='Order Date'
textAlign='Right' format='yMd' type='date' width=120></e-column>
      </e-columns>
    </ejs-grid>
  </div>
</template>
<script>
import Vue from "vue";
import { GridPlugin } from "@syncfusion/ej2-vue-grids";
import { data } from './datasource.js';
Vue.use(GridPlugin);
export default {
  data() {
    return {
      data: data
    };
  }
}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
</style>
```

{% previewsample "page.domainurl/code-snippet/grid/databind/default-cs4" %}

By default, `DataManager` uses `JsonAdaptor` for local data-binding.

### *Refresh the data source*

You can add/delete the datasource records through an external button. To reflect the datasource changes in grid, you need to invoke the [refresh](#) method.

Please follow the below steps to refresh the grid after datasource change.

**Step 1:** Add/delete the datasource record by using the following code.

```
`ts
this.data.unshift(customData); // Add a new record.
this.data.splice(selectedRow, 1); // Delete a record.
`
```

**Step 2:** When applied the changes in dataSource then refresh Grid at own.

```
`ts
this.data = [...this.data]; // Refresh the Grid dataSource.
`
```

### APP.VUE

```
<template>
  <div id="app">
    <ejs-button @click.native='addAction'>Add</ejs-button>
    <ejs-button @click.native='deleteAction'>Delete</ejs-button>
    <ejs-grid ref='grid' :dataSource='data' height='280px' >
      <e-columns>
        <e-column field='OrderID' headerText='Order ID'
textAlign='Right' width=100></e-column>
        <e-column field='CustomerID' headerText='Customer ID'
width=120></e-column>
        <e-column field='Freight' headerText='Freight'
textAlign='Center' format='C2' width=80></e-column>
        <e-column field='OrderDate' headerText='Order Date' type='date'
format='yMd' width=120></e-column>
      </e-columns>
    </ejs-grid>
  </div>
</template>
<script>
import Vue from "vue";
import { GridPlugin } from "@syncfusion/ej2-vue-grids";
import { ButtonPlugin } from "@syncfusion/ej2-vue-buttons";
import { data } from './datasource.js';
Vue.use(GridPlugin);
Vue.use(ButtonPlugin);
export default {
  data: () => {
    return {
      data: data,
    };
  },
  methods: {
    addAction: function() {
      let customData = { OrderID: 10247, CustomerID: "ASDFG", Freight: 40.4,
OrderDate: new Date(8367642e5) };
      this.data.unshift(customData);
      this.data = [...this.data];
    },
  },
}
```

```

deleteAction: function() {
  let selectedRow = this.$refs.grid.getSelectedRowIndex() [0];
  if (selectedRow !== undefined) {
    this.data.splice(selectedRow, 1);
  }
  else {
    alert("No records selected for delete operation");
  }
  this.data = [...this.data];
}
}
}
</script>
<style>
  @import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/grid/how-to/default-cs5" %}

### Remote data in Vue Grid component

To bind remote data to grid component, assign service data as an instance of **DataManager** to the [dataSource](#) property. To interact with remote data source, provide the endpoint url.

### APP.VUE

```

<template>
  <div id="app">
    <ejs-grid :dataSource="data">
      <e-columns>
        <e-column field='OrderID' headerText='Order ID'
textAlign='Right' width=90></e-column>
        <e-column field='CustomerID' headerText='Customer ID'
width=120></e-column>
        <e-column field='Freight' headerText='Freight' textAlign='Right'
format='C2' width=90></e-column>
        <e-column field='OrderDate' headerText='Order Date'
textAlign='Right' format='yMd' type='date' width=120></e-column>
      </e-columns>
    </ejs-grid>
  </div>
</template>
<script>
import Vue from "vue";
import { GridPlugin } from "@syncfusion/ej2-vue-grids";
import { DataManager, ODataAdaptor } from "@syncfusion/ej2-data";
Vue.use(GridPlugin);
export default {
  data() {
    let SERVICE_URI =
      "https://services.syncfusion.com/js/production/api/Orders";
    return {
      data: new DataManager({
        url: SERVICE_URI,
        adaptor: new ODataAdaptor()
      })
    };
  }
};

```

```

    }
  }
</script>
<style>
  @import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/grid/databind/remote-cs2" %}

By default, **DataManager** uses **ODataAdaptor** for remote data-binding.

#### *OData adaptor - Binding OData service*

**OData** is a standardized protocol for creating and consuming data. You can retrieve data from OData service using **DataManager**. You can refer to the following code example of remote data binding using OData service.

#### **APP.VUE**

```

<template>
  <div id="app">
    <ejs-grid :dataSource="data">
      <e-columns>
        <e-column field='OrderID' headerText='Order ID'
textAlign='Right' width=90></e-column>
        <e-column field='CustomerID' headerText='Customer ID'
width=120></e-column>
        <e-column field='Freight' headerText='Freight' textAlign='Right'
format='C2' width=90></e-column>
        <e-column field='OrderDate' headerText='Order Date'
textAlign='Right' format='yMd' type='date' width=120></e-column>
      </e-columns>
    </ejs-grid>
  </div>
</template>
<script>
import Vue from "vue";
import { GridPlugin } from "@syncfusion/ej2-vue-grids";
import { DataManager, ODataAdaptor } from "@syncfusion/ej2-data";
Vue.use(GridPlugin);
export default {
  data() {
    let SERVICE_URI =
      "https://services.syncfusion.com/js/production/api/Orders";
    return {
      data: new DataManager({
        url: SERVICE_URI,
        adaptor: new ODataAdaptor(),
        crossDomain: true
      })
    };
  }
}
</script>
<style>
  @import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
</style>

```

```
{% previewsample "page.domainurl/code-snippet/grid/databind/remote-cs3" %}
```

#### *OData v4 adaptor - Binding OData v4 service*

The ODataV4 is an improved version of OData protocols, and the **DataManager** can also retrieve and consume OData v4 services. For more details on OData v4 services, refer to the [odata documentation](#). To bind OData v4 service, use the **ODataV4Adaptor**.

#### **APP.VUE**

```
<template>
  <div id="app">
    <ejs-grid :dataSource="data">
      <e-columns>
        <e-column field='OrderID' headerText='Order ID'
textAlign='Right' width=90></e-column>
        <e-column field='CustomerID' headerText='Customer ID'
width=120></e-column>
        <e-column field='Freight' headerText='Freight' textAlign='Right'
format='C2' width=90></e-column>
        <e-column field='OrderDate' headerText='Order Date'
textAlign='Right' format='yMd' type='date' width=120></e-column>
      </e-columns>
    </ejs-grid>
  </div>
</template>
<script>
import Vue from "vue";
import { GridPlugin } from "@syncfusion/ej2-vue-grids";
import { DataManager, ODataV4Adaptor } from "@syncfusion/ej2-data";
Vue.use(GridPlugin);
export default {
  data() {
    let SERVICE_URI =
      "https://services.odata.org/V4/Northwind/Northwind.svc/Orders/";
    return {
      data: new DataManager({
        url: SERVICE_URI,
        adaptor: new ODataV4Adaptor(),
        crossDomain: true
      })
    };
  }
}
</script>
<style>
  @import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
</style>
```

```
{% previewsample "page.domainurl/code-snippet/grid/databind/remote-cs4" %}
```

#### *Web API Adaptor*

You can use **WebApiAdaptor** to bind grid with Web API created using OData endpoint.



```
<template>
<div id="app">
<ejs-grid :dataSource="data">
<e-columns>
<e-column field='OrderID' headerText='Order ID' textAlign='Right' width=90></e-column>
<e-column field='CustomerID' headerText='Customer ID' width=120></e-column>
<e-column field='Freight' headerText='Freight' textAlign='Right' format='C2' width=90></e-column>
<e-column field='OrderDate' headerText='Order Date' textAlign='Right' format='yMd' type='date'
width=120></e-column>
</e-columns>
</ejs-grid>
</div>
</template>
<script>
import Vue from "vue";
import { GridPlugin } from "@syncfusion/ej2-vue-grids";
import { DataManager, WebApiAdaptor } from "@syncfusion/ej2-data";
Vue.use(GridPlugin);
export default Vue.extend({
  data() {
    return {
      data: new DataManager({
        url: 'https://services.syncfusion.com/vue/production/api/Orders',
        adaptor: new WebApiAdaptor(),
        crossDomain: true
      })
    };
  }
});
</script>
<style>
@import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
</style>
```

The response object should contain properties `Items` and `Count` whose values are a collection of entities and the total count of the entities respectively.

The sample response object should look like below.

```
{
  Items: [{..}, {..}, {..}, ...],
  Count: 830
}
```

#### *Remote save adaptor*

You may need to perform all Grid Actions in client-side except the CRUD operations, that should be interacted with server-side to persist data. It can be achieved in Grid by using `RemoteSaveAdaptor`.

Datasource must be set to `json` property and set `RemoteSaveAdaptor` to the `adaptor` property. CRUD operations can be mapped to server-side using `updateUrl`, `insertUrl`, `removeUrl`, `batchUrl`, `crudUrl` properties.

You can use the following code example to use `RemoteSaveAdaptor` in Grid.

```
<template>
<div id="app">
  <ejs-grid :dataSource="dataSource" :editSettings='editSettings' :toolbar='toolbar'>
    <e-columns>
      <e-column field='OrderID' headerText='Order ID' :isPrimaryKey='true' textAlign='Right' width=90></e-column>
      <e-column field='CustomerID' headerText='Customer ID' width=120></e-column>
      <e-column field='Freight' headerText='Freight' textAlign='Right' format='C2' width=90></e-column>
      <e-column field='OrderDate' headerText='Order Date' textAlign='Right' format='yMd' type='date' width=120></e-column>
    </e-columns>
  </ejs-grid>
</div>
</template>
<script>
import Vue from "vue";
import { GridPlugin, Toolbar, Edit } from "@syncfusion/ej2-vue-grids";
```

```

import { DataManager, RemoteSaveAdaptor } from "@syncfusion/ej2-data";
import { data } from './datasource.js';
Vue.use(GridPlugin);
export default Vue.extend({
  data() {
    return {
      dataSource: new DataManager({
        json: data,
        adaptor: new RemoteSaveAdaptor,
        insertUrl: '/Home/Insert',
        updateUrl: '/Home/Update',
        removeUrl: '/Home/Delete'
      }),
      editSettings: { allowEditing: true, allowAdding: true, allowDeleting: true },
      toolbar: ['Add', 'Edit', 'Delete', 'Update', 'Cancel']
    };
  },
  provide: {
    grid: [Edit, Toolbar]
  }
});
</script>
<style>
@import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
</style>

```

The following code example describes the CRUD operations handled at server-side.

```

public ActionResult Update(OrdersDetails value)
{
  ...
  return Json(value);
}

```

```

public ActionResult Insert(OrdersDetails value)
{
    ...
    return Json(value);
}

public ActionResult Delete(int key)
{
    ...
    return Json(data);
}

```

### Custom adaptor

You can create your own adaptor by extending the built-in adaptors. For the sake of demonstrating custom adaptor approach, we are going to see how to add a serial number for the records by overriding the built-in response processing using the `processResponse` method of the `ODataAdaptor`.

### APP.VUE

```

<template>
  <div id="app">
    <ejs-grid :dataSource="data">
      <e-columns>
        <e-column field='Sno' headerText='SNO' textAlign='Right'
width=150></e-column>
        <e-column field='CustomerID' headerText='Customer ID'
width=120></e-column>
        <e-column field='Freight' headerText='Freight' textAlign='Right'
format='C2' width=90></e-column>
        <e-column field='OrderDate' headerText='Order Date'
textAlign='Right' format='yMd' type='date' width=120></e-column>
      </e-columns>
    </ejs-grid>
  </div>
</template>
<script>
import Vue from "vue";
import { GridPlugin } from "@syncfusion/ej2-vue-grids";
import { DataManager, ODataAdaptor } from "@syncfusion/ej2-data";
Vue.use(GridPlugin);
class SerialNoAdaptor extends ODataAdaptor {
  processResponse() {
    let i = 0;
    //calling base class processResponse function
    let original = super.processResponse.apply(this, arguments);
    //Adding serial number
    original.result.forEach((item) => item['Sno'] = ++i);
    return { result: original.result, count: original.count };
  }
}

```

```

export default {
  data() {
    let SERVICE_URI =
      "https://services.syncfusion.com/js/production/api/Orders";
    return {
      data: new DataManager({
        url: SERVICE_URI,
        adaptor: new SerialNoAdaptor()
      })
    };
  }
}
</script>
<style>
  @import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/grid/databind/remote-cs5" %}

### Offline mode

On remote data binding, all grid actions such as paging, sorting, editing, grouping, filtering, etc, will be processed on server-side. To avoid post back for every action, set the grid to load all data on initialization and make the actions process in client-side. To enable this behavior, use the **offline** property of **DataManager**.

### APP.VUE

```

<template>
  <div id="app">
    <ejs-grid :dataSource="data" :allowPaging="true"
    :pageSettings='pageSettings'>
      <e-columns>
        <e-column field='OrderID' headerText='Order ID'
        textAlign='Right' width=90></e-column>
        <e-column field='CustomerID' headerText='Customer ID'
        width=120></e-column>
        <e-column field='Freight' headerText='Freight' textAlign='Right'
        format='C2' width=90></e-column>
        <e-column field='OrderDate' headerText='Order Date'
        textAlign='Right' format='yMd' type='date' width=120></e-column>
      </e-columns>
    </ejs-grid>
  </div>
</template>
<script>
import Vue from "vue";
import { GridPlugin, Page } from "@syncfusion/ej2-vue-grids";
import { DataManager, ODataAdaptor } from "@syncfusion/ej2-data";
Vue.use(GridPlugin);
export default {
  data() {
    let SERVICE_URI =
      "https://services.syncfusion.com/js/production/api/Orders";
    return {
      data: new DataManager({
        url: SERVICE_URI,

```

```

        adaptor: new ODataAdaptor(),
        offline: true
    }},
    pageSettings: {pageSize: 7}
};
},
provide: {
    grid: [Page]
}
});
</script>
<style>
    @import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/grid/databind/remote-cs6" %}

### How to change the data source or columns dynamically in Vue Grid component

The grid can be dynamically update the data source, columns, or both, using the [changeDataSource](#) method. However, it is important to note that during the changing process for the data source and columns, the grid's existing actions such as sorting, filtering, grouping, aggregation, and searching will be removed. The `changeDataSource` method has two optional arguments, the first argument being the data source, and the second argument being the columns. So, you need to pass the two required arguments to the `changeDataSource` method. Some of the uses for using the `changeDataSource` method are explained in the topic below.

#### 1. Change both data source and columns:

To modify both the existing columns and the data source, you need to pass the both arguments to the `changeDataSource` method. The following example demonstrates how to change both the data source and columns.

You can assign a JavaScript object array to the [dataSource](#) property to bind local data to the grid. The code below provides an example of how to create a data source for the grid.

```

`typescript
export let data: Object[] = [
{
    OrderID: 10248, CustomerID: 'VINET', Freight: 32.38,
    ShipCity: 'Reims'
},
{
    OrderID: 10249, CustomerID: 'TOMSP', Freight: 11.61,
    ShipCity: 'Münster'
},
{
    OrderID: 10250, CustomerID: 'HANAR', Freight: 61.34,

```

```
ShipCity: 'Rio de Janeiro'
```

```
});
```

```
,
```

The following code demonstrates how to create the [columns](#) for the grid, which are based on the provided grid data source.

```
`typescript
```

```
newcolumn: [
```

```
{ field: 'Freight', headerText: 'Employee ID', width: 120, textAlign: 'Right' },
```

```
{ field: 'ShipRegion', headerText: 'Ship City', width: 140 }
```

```
]
```

```
,
```

The following code demonstrates updating the data source and columns defined above using the `changeDataSource` method.

```
`typescript
```

```
this.$refs.gridInstance.changeDataSource(data, newColumn);
```

```
,
```

## 2. Modify only the existing columns:

To modify the existing columns in a grid, you can either add or remove columns or change the entire set of columns using the `changeDataSource` method. To use this method, you should set the first parameter to null and provide the new columns as the second parameter. However, please note that if a column field is not specified in the data source, its corresponding column values will be empty. The following example illustrates how to modify existing columns.

The following code demonstrates how to add new columns with existing grid columns ('newColumn') by using `changeDataSource` method.

```
`typescript
```

```
newcolumn: [
```

```
{ field: 'Freight', headerText: 'Employee ID', width: 120, textAlign: 'Right' },
```

```
{ field: 'ShipRegion', headerText: 'Ship City', width: 140 }
```

```
]
```

```
let column: any = this.newColumn.push(...this.newColumn1);
```

```
this.$refs.gridInstance.changeDataSource(null, column);
```

```
,
```

## 3. Modify only the data source:

You can change the entire data source in the grid using the `changeDataSource` method. To use this method, you should provide the data source as the first argument, and the second argument which is optional can be used to specify new columns for the grid. If you are not specifying the columns, the grid

will generate the columns automatically based on the data source. The following example demonstrates how to modify the data source.

You can assign a JavaScript object array to the `dataSource` property to bind local data to the grid. The code below provides an example of how to create a new data source for the grid.

```
`typescript
export let employeeData: Object[] = [
{
  FirstName: 'Nancy', City: 'Seattle', Region: 'WA',
  Country: 'USA'
},
{
  FirstName: 'Andrew', City: 'London', Region: null,
  Country: 'UK',
},
{
  FirstName: 'Janet', City: 'Kirkland', Region: 'WA',
  Country: 'USA'
}
];
`
```

The following code demonstrates, how to use the `changeDataSource` method to bind the new **employeeData** to the grid.

```
`typescript
this.$refs.gridInstance.changeDataSource(employeeData);
`
```

\* The Grid state persistence feature does not support the `changeDataSource` method.

\* In this document, the above sample uses the local data for `changeDataSource` method. For those using a remote data source, refer to the [FlexibleData](#) resource.

### Immutable mode in Vue Grid component

The immutable mode optimizes the Grid re-rendering performance by using the object reference and [deep compare](#) concept. When performing the Grid actions, it will only re-render the modified or newly added rows and prevent the re-rendering of the unchanged rows.

To enable this feature, you have to set the [enableImmutableMode](#) property as **true**.

This feature uses the primary key value for data comparison. So, you need to provide the [isPrimaryKey](#) column.

### **APP.VUE**



```

<template>
  <div id="app">
    <table>
      <tbody>
        <tr>
          <td>
            <span id="immutableDelete"></span>
          </td>
        </tr>
        <tr>
          <td>
            <span id="normalDelete"></span>
          </td>
        </tr>
        <tr>
          <td>
            <div>
              <ejs-button cssClass="e-info" v-on:click.native="addTopEvent">
                Add 5 rows at top</ejs-button>
              <
                <ejs-button cssClass="e-info" v-
on:click.native="addBottomEvent"
                >Add 5 rows at bottom</ejs-button>
              >
                <ejs-button cssClass="e-info" v-on:click.native="deleteEvent">
                Delete 5 rows</ejs-button>
              >
                <ejs-button cssClass="e-info" v-on:click.native="sortEvent">
                Sort Order ID</ejs-button>
              >
                <ejs-button cssClass="e-info" v-on:click.native="pageEvent">
                Paging</ejs-button>
              >
            </div>
          </td>
        </tr>
        <tr>
          <td>
            <span><b>Immutable Grid</b></span>
            <ejs-grid
              ref="immutablegrid"
              :dataSource="data"
              height="250"
              :enableImmutableMode="true"
              :allowPaging="true"
              :pageSettings="pageSettings"
              :beforeDataBound="immutableBeforeDataBound"
              :dataBound="immutableDataBound"
            >
              <e-columns>
                <e-column
                  field="OrderID"
                  headerText="Order ID"
                  width="120"
                  isPrimaryKey="true"
                  textAlign="Right"
                ></e-column>

```

```

        <e-column
          field="ProductName"
          headerText="Product Name"
          width="160"
        ></e-column>
        <e-column
          field="ProductID"
          headerText="Product ID"
          width="120"
          textAlign="Right"
        ></e-column>
        <e-column
          field="CustomerID"
          headerText="Customer ID"
          width="120"
        ></e-column>
        <e-column
          field="CustomerName"
          headerText="Customer Name"
          width="160"
        ></e-column>
      </e-columns>
    </ejs-grid>
  </td>
</tr>
<tr>
  <td>
    <span><b>Normal Grid</b></span>
    <ejs-grid
      ref="nomalgrid"
      :dataSource="data"
      height="250"
      :allowPaging="true"
      :pageSettings="pageSettings"
      :beforeDataBound="beforeDataBound"
      :dataBound="dataBound"
    >
      <e-columns>
        <e-column
          field="OrderID"
          headerText="Order ID"
          width="120"
          isPrimaryKey="true"
          textAlign="Right"
        ></e-column>
        <e-column
          field="ProductName"
          headerText="Product Name"
          width="160"
        ></e-column>
        <e-column
          field="ProductID"
          headerText="Product ID"
          width="120"
          textAlign="Right"
        ></e-column>
      </e-columns>
    </ejs-grid>
  </td>

```

```

        field="CustomerID"
        headerText="Customer ID"
        width="120"
    ></e-column>
    <e-column
        field="CustomerName"
        headerText="Customer Name"
        width="160"
    ></e-column>
</e-columns>
</ejs-grid>
</td>
</tr>
</tbody>
</table>
</div>
</template>
<script>
import Vue from "vue";
import { GridPlugin, Page } from "@syncfusion/ej2-vue-grids";
import { ButtonPlugin } from "@syncfusion/ej2-vue-buttons";
import { data } from "../datasource.js";
Vue.use(GridPlugin);
Vue.use(ButtonPlugin);
export default {
    data() {
        return {
            data: data,
            pageSettings: { pageSize: 50 },
            immutableStart: 0,
            normalStart: 0,
            primaryKey: 0,
            immutableInit: true,
            init: true
        };
    },
    provide: {
        grid: [Page],
    },
    methods: {
        addTopEvent: function () {
            let immutableInstance = this.$refs.immutablegrid;
            let instance = this.$refs.normalgrid;
            let addedRecords = [
                { OrderID: ++this.primaryKey,
                  ProductName: "Chai",
                  ProductID: "Sasquatch Ale",
                  CustomerID: "QUEDE",
                  CustomerName: "Yoshi Tannamuri",
                },
                {
                  OrderID: ++this.primaryKey,
                  ProductName: "Georg Pippis",
                  ProductID: "Valkoinen suklaa",
                  CustomerID: "RATTC",
                  CustomerName: "Martín Sommer",
                },
            ],

```

```

        {
            OrderID: ++this.primaryKey,
            ProductName: "Yoshi Tannamuri",
            ProductID: "Gula Malacca",
            CustomerID: "COMMI",
            CustomerName: "Ann Devon",
        },
        {
            OrderID: ++this.primaryKey,
            ProductName: "Palle Ibsen",
            ProductID: "Rogede sild",
            CustomerID: "RATTC",
            CustomerName: "Paula Wilson",
        },
        {
            OrderID: ++this.primaryKey,
            ProductName: "Francisco Chang",
            ProductID: "Mascarpone Fabioli",
            CustomerID: "ROMEY",
            CustomerName: "Jose Pavarotti",
        },
    ];
    let aData = addedRecords.concat(immutableInstance.dataSource);
    instance.setProperties({ dataSource: aData });
    immutableInstance.setProperties({ dataSource: aData });
},
addBottomEvent: function () {
    let immutableInstance = this.$refs.immutablegrid;
    let instance = this.$refs.nomalgrid;
    let addedRecords = [
        {
            OrderID: ++this.primaryKey,
            ProductName: "Chai",
            ProductID: "Sasquatch Ale",
            CustomerID: "QUEDE",
            CustomerName: "Yoshi Tannamuri",
        },
        {
            OrderID: ++this.primaryKey,
            ProductName: "Georg Pippis",
            ProductID: "Valkoinen suklaa",
            CustomerID: "RATTC",
            CustomerName: "Martín Sommer",
        },
        {
            OrderID: ++this.primaryKey,
            ProductName: "Yoshi Tannamuri",
            ProductID: "Gula Malacca",
            CustomerID: "COMMI",
            CustomerName: "Ann Devon",
        },
        {
            OrderID: ++this.primaryKey,
            ProductName: "Palle Ibsen",
            ProductID: "Rogede sild",
            CustomerID: "RATTC",
            CustomerName: "Paula Wilson",
        },
    ];

```

```

    },
    {
      OrderID: ++this.primaryKey,
      ProductName: "Francisco Chang",
      ProductID: "Mascarpone Fabioli",
      CustomerID: "ROMEY",
      CustomerName: "Jose Pavarotti",
    },
  ];
  let aData = immutableInstance.dataSource.concat(addedRecords);
  instance.setProperties({ dataSource: aData });
  immutableInstance.setProperties({ dataSource: aData });
},
deleteEvent: function () {
  let immutableInstance = this.$refs.immutablegrid;
  let instance = this.$refs.normalgrid;
  immutableInstance.dataSource.splice(0, 5);
  instance.setProperties({ dataSource: immutableInstance.dataSource });
  immutableInstance.setProperties({
    dataSource: immutableInstance.dataSource,
  });
},
sortEvent: function () {
  let immutableInstance = this.$refs.immutablegrid;
  let instance = this.$refs.normalgrid;
  let aData = immutableInstance.dataSource.reverse();
  instance.setProperties({ dataSource: aData });
  immutableInstance.setProperties({ dataSource: aData });
},
pageEvent: function () {
  let immutableInstance = this.$refs.immutablegrid;
  let instance = this.$refs.normalgrid;
  let totalPages =
    immutableInstance.dataSource.length /
    immutableInstance.pageSettings.pageSize;
  let page = Math.floor(Math.random() * totalPages) + 1;
  instance.setProperties({ pageSettings: { currentPage: page } });
  immutableInstance.setProperties({ pageSettings: { currentPage: page }
});
},
immutableBeforeDataBound: function () {
  this.immutableStart = new Date().getTime();
},
immutableDataBound: function () {
  let val = this.immutableInit ? '' : new Date().getTime() -
this.immutableStart;
  document.getElementById("immutableDelete").innerHTML =
    "Immutable rendering Time: " + "<b>" + val + "</b>" + "<b>ms</b>";
  this.immutableStart = 0; this.immutableInit = false;
},
beforeDataBound: function () {
  this.normalStart = new Date().getTime();
},
dataBound: function () {
  let val = this.init ? '' : new Date().getTime() - this.normalStart;
  document.getElementById("normalDelete").innerHTML =
    "Normal rendering Time: " + "<b>" + val + "</b>" + "<b>ms</b>";

```

```
        this.normalStart = 0; this.init = false;
    },
    },
};
</script>
<style>
.e-grid {
    pointer-events: none;
}
@import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
</style>
```

{% previewsample "page.domainurl/code-snippet/grid/immutable-mode-cs1" %}

### Limitations

The following features are not supported in the immutable mode:

- Frozen rows and columns
- Row Template
- Detail Template
- Hierarchy Grid
- Column reorder
- Virtual scroll
- Infinite scroll
- Grouping

### Performance tips for Vue DataGrid Component

This article is a comprehensive guide on improving the loading performance of the Vue DataGrid, especially when dealing with large datasets along with large number of columns. It provides valuable insights into the steps that need to be followed to bind a large data source without experiencing any performance degradations. By offering detailed explanations and actionable tips, this resource aims to empower readers with the knowledge and best practices necessary to optimize the performance of the Vue DataGrid during data binding, ensuring a smooth and efficient user experience.

#### How to improve loading performance by binding large dataset

As you all know, a grid is made up of rows and columns. For instance, when you bind 10 rows and 10 columns, it means 100 elements will be rendered in the DOM (Document Object Model). So, it is recommended to render only a limited number of rows and columns to guarantee the best loading performance for the component.

#### *Optimizing performance with paging*

To boost the performance efficiency of your application, especially when dealing with large datasets, it is advised to implement paging. [Paging](#) allows you to display grid data in segmented pages, facilitating easier navigation through substantial datasets. This feature proves particularly beneficial in enhancing the overall performance of your application. For more information on implementing paging, you can refer to the [documentation](#) section dedicated to this feature.

#### *Optimizing performance with row virtualization or infinite scrolling*

To enhance your application's efficiency, especially when dealing with substantial datasets, it is recommended to either using [virtualization](#) or [infinite scrolling](#). Implementing these techniques can significantly reduce the load on your application and elevate its overall performance.

1. **Virtualization:** The Virtual scrolling feature in the Vue Data Grid enables the efficient handling and display of large volumes of data without compromising performance. This approach optimizes the rendering process by loading only the visible rows within the Grid viewport, rather than rendering the entire dataset simultaneously. For more information on implementing row virtualization , you can refer to the [documentation](#) section dedicated to this feature.
2. **Infinite scrolling:** The Infinite Scrolling feature in the Vue Data Grid is a powerful tool for seamlessly handling extensive data sets without compromising grid performance. It operates on a "load-on-demand" concept, ensuring that data is fetched only when needed. In the default infinite scrolling mode, a new block of data is loaded each time the scrollbar reaches the end of the vertical scroller. For more information on implementing infinite scrolling , you can refer to the [documentation](#) section dedicated to this feature.

#### *Optimizing performance with column virtualization in large no of columns*

[Column virtualization](#) feature in the Vue Data Grid that allows you to optimize the rendering of columns by displaying only the columns that are currently within the viewport. It allows horizontal scrolling to view additional columns. This feature is particularly useful when dealing with grids that have a large number of columns, as it helps to improve the performance and reduce the initial loading time.

It is possible to enable both row and column virtualization. This feature allows for efficient handling of large datasets by dynamically loading only the visible rows and columns, optimizing performance and enhancing the overall responsiveness of the grid. For more information on implementing column virtualization , you can refer to the [documentation](#) section dedicated to this feature.

#### *How to overcome browser height limitation in virtual scrolling*

##### [Documentation link](#)

#### *How to improve loading performance by binding large data by showing custom text or element*

When integrating image or template elements into a column, it's recommended to utilize the [Column Template](#) feature rather than customizing the data through [rowDataBound](#) or [queryCellInfo](#) events. These events are triggered for each row and cell rendering, introducing delays in the component's rendering process. Moreover, rendering custom elements using these events may result in the persistence of rendered elements, potentially causing longer rendering times over time. By opting for the column template feature, you can efficiently meet this requirement without experiencing rendering delays and ensure a more streamlined rendering process.

#### *How to improve loading performance by referring individual script and CSS*

To improve the performance of Syncfusion Grid component during the initial render as well as certain actions, suggested you to download the specific component scripts using CRG (Custom Resource Generator) to speed up the project. By default, the ej2.min.js script file contains all the Syncfusion component scripts. So, it will take some time to load the scripts to the project. Using [CRG](#), you can select the components which you want to use, and the modules for those components, then you can download the scripts and CSS for the selected components and use them as per your need.

##### [CRG website link](#)

So to improve the performance of grid during the initial rendering, suggested you to refer individual script and CSS.

#### *How to update cell values without frequent server calls*

Efficiently update cell values without the need for frequent server calls, especially beneficial for live update scenarios. Even when the data is initially bound from the server, performing edit operations can

be done without triggering a database refresh. Utilize the [setCellValue](#) method to update the DataGrid without affecting the database and only refresh the UI.

#### How to optimize server-side data operations with adaptors

The Vue DataGrid provides support for various adaptors (OData, ODataV4, WebAPI, URL, etc.) to facilitate server-side data operations and CRUD functionalities. By leveraging these adaptors along with the **DataManager** component, you can seamlessly bind remote data sources to the grid and execute actions. During data operations like filtering, sorting, and paging, the corresponding action queries are generated as per the adaptor's requirements. It is crucial to handle these actions on the application end and return the processed data back to the grid. Refer to the documentation for comprehensive details. It's worth noting that for efficient data processing, the suggested order for returning processed data to the grid is as follows

- Filtering
- Sorting
- Aggregates
- Paging
- Grouping

#### How to avoid MaxJsonLength error while passing large amount of records

The Vue Grid component is client-server based. So, we send the data as JSON object between client and server. The reported issue occurs due to the serialization of the large-sized JSON object. We need to increase the maximum length for serializing the large-sized JSON object. You have to alter the [MaxJsonLength](#) property on your web.config file or in the place of deserialization.

##### **Solution: 1**

```
`csharp
<configuration>
<system.web.extensions>
<scripting>
<webServices>
<jsonSerialization maxJsonLength="50000000"/>
</webServices>
</scripting>
</system.web.extensions>
</configuration>
`
```

##### **Solution : 2**

```
`csharp
var serializer = new JavaScriptSerializer { MaxJsonLength = Int32.MaxValue };
`
```



### Microsoft excel limitation while exporting millions of records to excel file format

By default, Microsoft Excel supports only 1,048,576 records in an excel sheet. Hence it is not possible to export millions of records to excel. You can refer the [documentation](#) link for more details on Microsoft excel specifications and limits. So suggest to export the data in CSV (Comma-Separated Values) or other formats that can handle large datasets more efficiently than Excel.

## Columns

### Columns in Vue Grid component

In Syncfusion Vue Grid, Columns are fundamental elements that play a pivotal role in organizing and displaying data within your application. They serve as the building blocks for data presentation, allowing you to specify what data fields to show, how to format and style them, and how to enable various interactions within the grid.

### Column types

The Syncfusion Grid component allows you to specify the type of data that a column binds using the [columns.type](#) property. The `type` property is used to determine the appropriate [format](#), such as [number](#) or [date](#), for displaying the column data.

Grid supports the following column types:

- **string**: Represents a column that binds to string data. This is the default type if the `type` property is not defined.
- **number**: Represents a column that binds to numeric data. It supports formatting options for displaying numbers.
- **boolean**: Represents a column that binds to boolean data. It displays checkboxes for boolean values.
- **date**: Represents a column that binds to date data. It supports formatting options for displaying dates.
- **datetime**: Represents a column that binds to date and time data. It supports formatting options for displaying date and time values.
- **checkbox**: Represents a column that displays checkboxes.

Here is an example of how to specify column types in a grid using the types mentioned above.

### APP.VUE

```
<template>
  <div id="app">
    <ejs-grid ref='grid' :dataSource="data" height='315px'>
      <e-columns>
        <e-column field='OrderID' headerText='Order ID'
textAlign='Right' type='number' width=150></e-column>
        <e-column field='CustomerID' headerText='Customer ID'
type='string' width=120></e-column>
        <e-column field='Freight' headerText='Freight' textAlign='Right'
format='C2' type='number' width=150></e-column>
        <e-column field='OrderDate' headerText='Order Date'
textAlign='Right' format='yMd' type='date' width=120></e-column>
      </e-columns>
    </ejs-grid>
  </div>
</template>
```

```

<script>
import Vue from "vue";
import { GridPlugin } from "@syncfusion/ej2-vue-grids";
import { data } from './datasource.js';
Vue.use(GridPlugin);
export default {
  data() {
    return {
      data: data
    };
  },
  provide: {
    grid: []
  }
}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/grid/column/column-type-cs1" %}

\* If the [type](#) is not defined, then it will be determined from the first record of the [dataSource](#).

\* In case if the first record of the [dataSource](#) is null/blank value for a column then it is necessary to define the [type](#) for that column. This is because the grid uses the column type to determine which filter dialog to display for that column.

#### Difference between boolean type and checkbox type column

1. Use the **boolean** column type when you want to bind boolean values from your datasource and/or edit boolean property values from your type.
2. Use the **checkbox** column type when you want to enable selection/deselection of the whole row.
3. When the grid column **type** is a **checkbox**, the selection type of the grid **selectionSettings** will be multiple. This is the default behavior.
4. If you have more than one column with the column type as a **checkbox**, the grid will automatically enable the other column's checkbox when selecting one column checkbox.

To learn more about how to render boolean values as checkboxes in a Syncfusion GridColumn, please refer to the [Render Boolean Values as Checkbox](#) section.

#### Column width

To adjust the column width in a Grid, you can use the [width](#) property within the [columns](#) property of the Grid configuration. This property enables you to define the column width in pixels or as a percentage. You can set the width to a specific value, like **100** for 100 pixels, or as a percentage value, such as **25%** for 25% of the available width.

1. Grid column width is calculated based on the sum of column widths. For example, a grid container with 4 columns and a width of 800 pixels will have columns with a default width of 200 pixels each.

2. If you specify widths for some columns but not others, the Grid will distribute the available width equally among the columns without explicit widths. For example, if you have 3 columns with widths of 100px, 200px, and no width specified for the third column, the third column will occupy the remaining width after accounting for the first two columns.
3. Columns with percentage widths are responsive and adjust their width based on the size of the grid container. For example, a column with a width of 50% will occupy 50% of the grid width and will adjust proportionally when the grid container is resized to a smaller size.
4. When you manually resize columns in Syncfusion Grid, a minimum width is set to ensure that the content within the cells remains readable. By default, the minimum width is set to 10 pixels if not explicitly specified for a column.
5. If the total width of all columns exceeds the width of the grid container, a horizontal scrollbar will automatically appear to allow horizontal scrolling within the grid.
6. When the columns is hide using the column chooser, the width of the hidden columns is removed from the total grid width, and the remaining columns will be resized to fill the available space.
7. If the parent element has a fixed width, the grid component will inherit that width and occupy the available space. However, if the parent element does not have a fixed width, the grid component will adjust its width dynamically based on the available space.

### Supported types for column width

Syncfusion Grid supports the following three types of column width:

#### 1. Auto

The column width is automatically calculated based on the content within the column cells. If the content exceeds the width of the column, it will be truncated with an ellipsis (...) at the end. You can set the width for columns as **auto** in your Grid configuration as shown below:

```
`html
```

```
<e-column field='OrderID' headerText='Order ID' textAlign='Right' width='auto'></e-column>
```

```
,
```

#### 2. Percentage

The column width is specified as a percentage value relative to the width of the grid container. For example, a column width of 25% will occupy 25% of the total grid width. You can set the width for columns as **percentage** in your Grid configuration as shown below:

```
`html
```

```
<e-column field='OrderID' headerText='Order ID' textAlign='Right' width='25%'></e-column>
```

```
,
```

#### 3. Pixel

The column width is specified as an absolute pixel value. For example, a column width of 100px will have a fixed width of 100 pixels regardless of the grid container size. You can set the width for columns as **pixel** in your Grid configuration as shown below:

```
`html
```

```
<e-column field='OrderID' headerText='Order ID' textAlign='Right' width='100'></e-column>
```

**APP.VUE**

```

<template>
  <div id="app">
    <ejs-grid ref='grid' :dataSource="data" height='315px'>
      <e-columns>
        <e-column field='OrderID' headerText='Order ID'
textAlign='Right' width='auto'></e-column>
        <e-column field='CustomerID' headerText='Customer ID'
width=120></e-column>
        <e-column field='Freight' headerText='Freight'
textAlign='Right' format='C2' width='100px'></e-column>
        <e-column field='OrderDate' headerText='Order Date'
textAlign='Right' format='yMd' width='30%'></e-column>
      </e-columns>
    </ejs-grid>
  </div>
</template>
<script>
import Vue from "vue";
import { GridPlugin } from "@syncfusion/ej2-vue-grids";
import { data } from './datasource.js';
Vue.use(GridPlugin);
export default {
  data() {
    return {
      data: data
    };
  },
  provide: {
    grid: []
  }
}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/grid/column/column-width-cs1" %}

*Column formatting*

Column formatting is a powerful feature in Syncfusion Grid that allows you to customize the display of data in grid columns. You can apply different formatting options to columns based on your requirements, such as displaying numbers with specific formats, formatting dates according to a specific locale, and using templates to format column values.

You can use the [columns.format](#) property to specify the format for column values.

**APP.VUE**

```

<template>
  <div id="app">
    <ejs-grid :dataSource='data' height='315'>
      <e-columns>

```

```

        <e-column field='OrderID' headerText='Order ID'
textAlign='Right' width=100></e-column>
        <e-column field='CustomerID' headerText='Customer ID'
width=120></e-column>
        <e-column field='Freight' headerText='Freight' format = 'C2'
width=80></e-column>
        <e-column field='OrderDate' headerText='Order Date'
format='yMd' textAlign='Right' width=80></e-column>
    </e-columns>
</ejs-grid>
</div>
</template>
<script>
import Vue from "vue";
import { GridPlugin } from "@syncfusion/ej2-vue-grids";
import { data } from './datasource.js';
Vue.use(GridPlugin);
export default {
  data() {
    return {
      data: data
    };
  },
  methods: {
  }
}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/grid/column/column-format-cs1" %}

- \* The grid uses the [Internalization](#) library to format values based on the specified format and culture.
- \* By default, the [number](#) and [date](#) values are formatted in **en-US** locale. You can localize the currency and date in different locale as explained [here](#).
- \* The available format codes may vary depending on the data type of the column.
- \* You can also customize the formatting further by providing a custom function to the **format** property, instead of a format string.
- \* Make sure that the format string is valid and compatible with the data type of the column, to avoid unexpected results.

#### Number formatting

Number formatting allows you to customize the display of numeric values in grid columns. You can use standard numeric format strings or custom numeric format strings to specify the desired format. The [columns.format](#) property can be used to specify the number format for numeric columns. For example, you can use the following format strings to format numbers:

Format	Description	Remarks

| ----- | ----- | -----  
 ----- |

| N | Denotes numeric type. | The numeric format is followed by integer value as N2, N3. etc which denotes the number of precision to be allowed.

|

| C | Denotes currency type. | The currency format is followed by integer value as C2, C3. etc which denotes the number of precision to be allowed.

|

| P | Denotes percentage type | The percentage format expects the input value to be in the range of 0 to 1. For example the cell value **0.2** is formatted as **20%**. The percentage format is followed by integer value as P2, P3. etc which denotes the number of precision to be allowed. |

The following example code demonstrates the formatting of data for **Mark 1** and **Mark 2** using the '**N**' format, **Percentage of Marks** using the '**P**' format, and **Fees** using the '**C**' format.

### APP.VUE

```
<template>
  <div id="app">
    <ejs-grid :dataSource='data' height='182px' allowPaging=true>
      <e-columns>
        <e-column field='RollNo' headerText='Roll No'></e-column>
        <e-column field='Mark1' headerText='Mark 1'></e-column>
        <e-column field='Mark2' headerText='Mark 2' format='N'></e-
column>
        <e-column field='Average' headerText='Average'
format='N2'></e-column>
        <e-column field='Percentage' headerText='Percentage of Marks'
format='P'></e-column>
        <e-column field='Fees' headerText='Fees' textAlign='Right'
format='C'></e-column>
      </e-columns>
    </ejs-grid>
  </div>
</template>
<script>
import Vue from "vue";
import { GridPlugin, Page } from "@syncfusion/ej2-vue-grids";
import { data } from './datasource.js';
Vue.use(GridPlugin);
export default {
  data() {
    return {
      data: data
    };
  },
  methods: {
  },
  provide: {
    grid: [Page]
  }
}
</script>
```

```
<style>
  @import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
</style>
```

{% previewsample "page.domainurl/code-snippet/grid/column/column-format-cs2" %}

To learn more about number formatting, you can refer to the [number](#) section.

### Date formatting

Date formatting in grid columns allows you to customize how date values are displayed. You can use standard date format strings, such as “d,” “D,” “MMM dd, yyyy,” and more, or create your own custom format strings. To specify the desired date format, you can use the [format](#) property of the Grid columns. For example, you can set `columns.format` as a string like “yMd” for a built-in date format.

Additionally, you can use custom format strings to format date values, and examples of custom formats and formatted date values are provided in the table below.

Format | Formatted value

{ type:'date', format:'dd/MM/yyyy' } | 04/07/1996

{ type:'date', format:'dd.MM.yyyy' } | 04.07.1996

{ type:'date', skeleton:'short' } | 7/4/96

{ type: 'dateTime', format: 'dd/MM/yyyy hh:mm a' } | 04/07/1996 12:00 AM

{ type: 'dateTime', format: 'MM/dd/yyyy hh:mm:ss a' } | 07/04/1996 12:00:00 AM

### APP.VUE

```
<template>
  <div id="app">
    <ejs-grid :dataSource='data' height='182px' allowPaging=true>
      <e-columns>
        <e-column field='OrderID' headerText='Order ID'
textAlign='Right' width=90></e-column>
        <e-column field='Freight' headerText='Freight'
textAlign='Right' format='C2' width=90></e-column>
        <e-column field='OrderDate' :format='formatOptions'
headerText='Order Date' format='yMd' textAlign='Right' width=120></e-column>
        <e-column field='OrderDate' :format='shipFormat'
headerText='Ship Date' textAlign='Right' width=150></e-column>
      </e-columns>
    </ejs-grid>
  </div>
</template>
<script>
import Vue from "vue";
import { GridPlugin, Page } from "@syncfusion/ej2-vue-grids";
import { data } from './datasource.js';
Vue.use(GridPlugin);
export default {
  data() {
    return {
      data: data,
      formatOptions: { type: 'date', format: 'dd/MM/yyyy' },
      shipFormat: { type: 'dateTime', format: 'dd/MM/yyyy hh:mm a' }
    }
  }
}
```

```

    };
  },
  methods: {
  },
  provide: {
    grid: [Page]
  }
}
</script>
<style>
  @import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/grid/column/column-format-cs3" %}

To learn more about date formatting, you can refer to [Date formatting](#).

#### Format the date column based on localization

You can also format the date column based on the localization settings of the user's browser. You can use the [format](#) property of the Grid columns along with the [locale](#) property to specify the desired date format based on the locale.

In this example, the format property specifies the date format as "yyyy-MMM-dd", and the locale property specifies the locale as "de-DE" for German (Germany).

#### APP.VUE

```

<template>
  <div id="app">
    <ejs-grid :dataSource='data' locale='de-DE' height='315px'>
      <e-columns>
        <e-column field='OrderID' headerText='Order ID'
textAlign='Right' width=120></e-column>
        <e-column field='CustomerID' headerText='Customer ID'
width=150></e-column>
        <e-column field='Freight' headerText='Freight'
textAlign='Right' width=150></e-column>
        <e-column field='OrderDate' headerText='Order Date'
:format='dateFormatOptions' textAlign='Right' width=150></e-column>
        <e-column field='ShipName' headerText='Ship Name'
width=150></e-column>
      </e-columns>
    </ejs-grid>
  </div>
</template>
<script>
import Vue from "vue";
import { L10n, loadCldr, setCulture, setCurrencyCode } from
"@syncfusion/ej2-base";
import * as currencies from './currencies.json';
import * as cagregorian from './ca-gregorian.json';
import * as numbers from './numbers.json';
import * as timeZoneNames from './timeZoneNames.json';
import * as numberingSystems from './numberingSystems.json';
import { GridPlugin } from "@syncfusion/ej2-vue-grids";
import { data } from './datasource.js';

```



```

setCulture('de-DE');
setCurrencyCode('EUR');
L10n.load({
  'de-DE': {
    'grid': {
      'EmptyRecord': 'Keine Aufzeichnungen angezeigt',
      'GroupDropArea': 'Ziehen Sie einen Spaltenkopf hier, um die
Gruppe ihre Spalte',
      'UnGroup': 'Klicken Sie hier, um die Gruppierung aufheben',
      'EmptyDataSourceError': 'DataSource darf bei der Erstauslastung
nicht leer sein, da Spalten aus der dataSource im AutoGenerate
Spaltenraster',
      'Item': 'Artikel',
      'Items': 'Artikel'
    },
    'pager': {
      'currentPageInfo': '{0} von {1} Seiten',
      'totalItemsInfo': '({0} Beiträge)',
      'firstPageTooltip': 'Zur ersten Seite',
      'lastPageTooltip': 'Zur letzten Seite',
      'nextPageTooltip': 'Zur nächsten Seite',
      'previousPageTooltip': 'Zurück zur letzten Seit',
      'nextPagerTooltip': 'Gehen Sie zu den nächsten Pager-Elementen',
      'previousPagerTooltip': 'Gehen Sie zu vorherigen Pager-
Elementen'
    }
  }
});
Vue.use(GridPlugin);
export default {
  data() {
    return {
      data: data,
      dateFormatOptions: {type: 'date', format: 'dd-MMM-yy'}
    };
  }
}
</script>
<style>
  @import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/grid/column/column-format-cs4" %}

#### Format template column value

In the Syncfusion Vue Grid, you have the ability to customize the display of data in a column through the use of template columns. Formatting template column values is essential for enhancing the visual representation of data in a web application. It allows the customization of the appearance of specific column data, such as dates and numbers, to improve readability and user understanding.

To illustrate how to format a template column value, consider the following example where the **OrderDate** column is formatted to display dates in the **'Jul 04, 1996'** format.

#### APP.VUE

```
<template>
```

```

<div id="app">
  <ejs-grid :dataSource='data' height='315px'>
    <e-columns>
      <e-column field='OrderID' headerText='Order ID'
textAlign='Right' width=120></e-column>
      <e-column field='CustomerID' headerText='Customer ID'
width=150></e-column>
      <e-column field='Freight' headerText='Freight'
textAlign='Right' width=150></e-column>
      <e-column field='OrderDate' headerText='Order Date'
:template='columnTemplate' textAlign='Right' width=150></e-column>
      <e-column field='ShipName' headerText='Ship Name'
width=150></e-column>
    </e-columns>
  </ejs-grid>
</div>
</template>
<script>
import Vue from "vue";
import { GridPlugin } from "@syncfusion/ej2-vue-grids";
import { data } from './datasource.js';
Vue.use(GridPlugin);
export default {
  data() {
    return {
      data: data,
      columnTemplate: function () {
        return {
          template: Vue.component('columnTemplate', {
            template: `<div> {{formatDate(data.OrderDate)}}
</div>`,
            methods: {
              formatDate(date) {
                return new
Date(date).toLocaleDateString("en-US", { day: "2-digit", month: "short",
year: "numeric" });
            },
          },
        );
      },
    };
  },
};
</script>
<style>
@import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/grid/column/column-format-cs5" %}

### Custom formatting

Syncfusion Grid allows you to customize the formatting of data in the grid columns. You can apply custom formats to numeric or date columns to display data in a specific way according to the

requirements. To apply custom formatting to grid columns in Syncfusion Grid, you can use the [format](#) property. Here's an example of how you can use custom formatting for numeric and date columns:

In the below example, the **numberFormatOptions** object is used as the **format** property for the **'Freight'** column to apply a custom numeric format with four decimal places. Similarly, the **dateFormatOptions** object is used as the **format** property for the **'OrderDate'** column to apply a custom date format displaying the date in the format of day-of-the-week, month abbreviation, day, and 2-digit year (e.g. Sun, May 8, '23).

#### APP.VUE

```
<template>
  <div id="app">
    <ejs-grid :dataSource='data' height='315px'>
      <e-columns>
        <e-column field='OrderID' headerText='Order ID'
textAlign='Right' width=120></e-column>
        <e-column field='CustomerID' headerText='Customer ID'
width=150></e-column>
        <e-column field='Freight' headerText='Freight'
textAlign='Right' :format='numberFormatOptions' width=150></e-column>
        <e-column field='OrderDate' headerText='Order Date'
:format='dateFormatOptions' textAlign='Right' width=150></e-column>
        <e-column field='ShipName' headerText='Ship Name'
width=150></e-column>
      </e-columns>
    </ejs-grid>
  </div>
</template>
<script>
import Vue from "vue";
import { GridPlugin } from "@syncfusion/ej2-vue-grids";
import { data } from './datasource.js';
Vue.use(GridPlugin);
export default {
  data() {
    return {
      data:data,
      numberFormatOptions: {
        // Custom format for numeric columns
        format: '##.0000',
      },
      dateFormatOptions: {
        // Custom format for date columns
        type: 'Date',
        format: "EEE, MMM d, 'yyyy",
      }
    };
  }
}
</script>
<style>
  @import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
</style>
```

```
{% previewsample "page.domainurl/code-snippet/grid/column/column-format-cs6" %}
```

To learn more about custom formatting, you can refer to [Custom Date formatting](#) and [Custom Number formatting](#).

### *Align the text of content*

You can align the text in the content of a Grid column using the `textAlign` property. This property allows you to specify the alignment of the text within the cells of a particular column in the Grid. By default, the text is aligned to the left, but you can change the alignment by setting the value of the `textAlign` property to one of the following options:

*Left:* Aligns the text to the left (default). **Center:** Aligns the text to the center. *Right:* Aligns the text to the right. **Justify:** Align the text to the justify.

Here is an example of using the `textAlign` property to align the text of a Grid column:

### **APP.VUE**

```
<template>
<div id='app'>
  <div style='display: flex'>
    <label style='padding: 10px 10px 12px 0'> Align the text for
columns: </label>
    <ejs-dropdownlist ref='dropdown' id='dropdownlist' style='margin-
top:5px' index='0' width='150'
      :dataSource='alignmentData' :fields='fields'
      :change='change'></ejs-dropdownlist>
  </div>
  <ejs-grid ref='grid' id='grid' :dataSource='data' height='315'>
    <e-columns>
      <e-column field='OrderID' headerText='Order ID'
textAlign='Right' width=100></e-column>
      <e-column field='CustomerID' headerText='Customer ID'
width=120></e-column>
      <e-column field='Freight' headerText='Freight' format='C2'
width=80></e-column>
      <e-column field='OrderDate' headerText='Order Date' format='yMd'
textAlign='Right' width=80></e-column>
    </e-columns>
  </ejs-grid>
</div>
</template>
<script>
import Vue from "vue";
import { GridPlugin } from "@syncfusion/ej2-vue-grids";
import { DropDownListPlugin } from "@syncfusion/ej2-vue-dropdowns";
import { data } from './datasource.js';
Vue.use(GridPlugin);
Vue.use(DropDownListPlugin);
export default {
  data() {
    return {
      data: data,
      fields: { text: 'text', value: 'value' },
      alignmentData: [
        { text: 'Left', value: 'Left' },
        { text: 'Right', value: 'Right' },
```

```

        { text: 'Center', value: 'Center' },
        { text: 'Justify', value: 'Justify' },
    ],
    };
},
methods: {
    change: function (args) {
        let grid = this.$refs.grid.$el.ej2_instances[0];
        grid.columns.forEach(col => {
            col.textAlign = args.value;
        });
        grid.refreshColumns();
    }
}
}
</script>
<style>
@import '../node_modules/@syncfusion/ej2-vue-dropdowns/styles/material.css';
@import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/grid/column/column-align-cs1" %}

\* The `textAlign` property only changes the alignment content not the column header. If you want to align both the column header and content, you can use the [headerTextAlign](#) property.

#### Render boolean value as checkbox

The Grid component allows you to render boolean values as checkboxes in columns. This can be achieved by using the [displayAsCheckBox](#) property, which is available in the [columns](#). This property is useful when you have a boolean column in your Grid and you want to display the values as checkboxes instead of the default text representation of **true** or **false**.

To enable the rendering of boolean values as checkboxes, you need to set the `displayAsCheckBox` property of the `columns` to **true**.

#### APP.VUE

```

<template>
  <div id="app">
    <ejs-grid :dataSource='data' height='315'>
      <e-columns>
        <e-column field='OrderID' headerText='Order ID'
        textAlign='Right' width=100></e-column>
        <e-column field='CustomerID' headerText='Customer ID'
        width=120></e-column>
        <e-column field='Freight' headerText='Freight' textAlign=
        'Right' width=120 format= 'C2'></e-column>
        <e-column field='ShipCountry' headerText='Ship Country'
        width=150></e-column>
        <e-column field='Verified' headerText='Verified'
        displayAsCheckBox='true' width=150></e-column>
      </e-columns>
    </ejs-grid>
  </div>
</template>
<script>

```

```
import Vue from "vue";
import { GridPlugin, Page } from "@syncfusion/ej2-vue-grids";
import { data } from './datasource.js';
Vue.use(GridPlugin);
export default {
  data() {
    return {
      data: data,
    };
  },
  provide: {
    grid: [Page]
  }
}
</script>
<style>
  @import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
</style>
```

{% previewsample "page.domainurl/code-snippet/grid/column/default-cs22" %}

\* The `displayAsCheckBox` property is only applicable to boolean values in Grid columns.

\* When `displayAsCheckBox` is set to **true**, the boolean values will be rendered as checkboxes in the Grid column, with checked state indicating **true** and unchecked state indicating **false**.

#### *How to prevent checkbox in the blank row*

To prevent the checkbox in the blank row of the Grid, even if the `displayAsCheckBox` property is set to true for that column, you can use the `rowDataBound` event and check for empty or null values in the row data. If all the values in the row are empty or null, you can set the inner HTML of the corresponding cell to an empty string to hide the checkbox.

Here is an example of how you can prevent a checkbox from being displayed in a blank row in a Grid:

#### **APP.VUE**

```
<template>
  <div id="app">
    <ejs-grid id="grid" :dataSource="data" :rowDataBound='rowDataBound'>
      <e-columns>
        <e-column field='OrderID' headerText='Order ID'
textAlign='Right' width=90></e-column>
        <e-column field='CustomerID' headerText='Customer ID'
width=120></e-column>
        <e-column field='Freight' headerText='Freight' textAlign='Right'
format='C2' width=90></e-column>
        <e-column field='Verified' headerText='Verified'
displayAsCheckBox='true' width=120> </e-column>
      </e-columns>
    </ejs-grid>
  </div>
</template>
<script>
import Vue from "vue";
import { GridPlugin } from "@syncfusion/ej2-vue-grids";
import { data } from './datasource.js';
```

```

Vue.use(GridPlugin);
export default {
  data() {
    return {
      data: data
    };
  },
  methods: {
    rowDataBound(args) {
      let grid = document.getElementById('grid').ej2_instances[0];
      let count = 0;
      let keys = Object.keys(args.data);
      for (let i = 0; i < keys.length; i++) {
        if (args.data[keys[i]] == null || args.data[keys[i]] == '' ||
args.data[keys[i]] == undefined) {
          count++;
        }
      }
      if (count == keys.length) {
        for (let i = 0; i < grid.columns.length; i++) {
          if (grid.columns[i].displayAsCheckBox) {
            args.row.children[i].innerHTML = '';
          }
        }
      }
    }
  }
}
</script>
<style>
  @import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/grid/column/blank-row-cs1" %}

### AutoFit columns

The Grid has a feature that allows it to automatically adjust column widths based on the maximum content width of each column when you double-click on the resizer symbol located in a specific column header. This feature ensures that all data in the grid rows is displayed without wrapping. To use this feature, you need to inject the **Resize** module in the provide section and enable the resizer symbol in the column header by setting the [allowResizing](#) property to true in the grid.

### Resizing a column to fit its content using autoFit method

The [autoFitColumns](#) method resizes the column to fit the widest cell's content without wrapping. You can autofit a specific column at initial rendering by invoking the `autoFitColumns` method in [dataBound](#) event.

To use the `autoFitColumns` method, inject the **Resize** module in the provide section.

### APP.VUE

```

<template>
  <div id="app">
    <ejs-grid ref='grid' :dataSource="data" :dataBound='dataBound'
height='315px'>
      <e-columns>

```

```

        <e-column field='OrderID' headerText='Order ID'
textAlign='Right' width=150></e-column>
        <e-column field='CustomerID' headerText='Customer ID'
width=120></e-column>
        <e-column field='Freight' headerText='Freight' textAlign='Right'
format='C2' width=150></e-column>
        <e-column field='OrderDate' headerText='Order Date'
textAlign='Right' format='yMd' type='date' width=120></e-column>
    </e-columns>
</ejs-grid>
</div>
</template>
<script>
import Vue from "vue";
import { GridPlugin, Resize } from "@syncfusion/ej2-vue-grids";
import { data } from './datasource.js';
Vue.use(GridPlugin);
export default {
  data() {
    return {
      data: data
    };
  },
  methods: {
    dataBound: function() {
      this.$refs.grid.autoFitColumns(['OrderDate', 'CustomerID']);
    }
  },
  provide: {
    grid: [Resize]
  }
}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/grid/column/default-cs1" %}

You can autofit all the columns by invoking the `autoFitColumns` method without specifying column names.

#### AutoFit columns with empty space

The Autofit feature is utilized to display columns in a grid based on the defined width specified in the columns declaration. If the total width of the columns is less than the width of the grid, this means that white space will be displayed in the grid instead of the columns auto-adjusting to fill the entire grid width.

You can enable this feature by setting the `autoFit` property set to true. This feature ensures that the column width is rendered only as defined in the Grid column definition.

#### APP.VUE

```

<template>
  <div id="app">

```



```

    <ejs-grid ref='grid' :dataSource="data" :allowResizing='true'
height='400px' width='850px' :autoFit='true'>
    <e-columns>
        <e-column field='OrderID' headerText='Order ID' minWidth='100'
width='150' maxWidth='200' textAlign='Right'></e-column>
        <e-column field='CustomerID' headerText='Customer ID'
minWidth='8' width='150'></e-column>
        <e-column field='Freight' headerText='Freight' minWidth='8'
width='120' format='C2' textAlign='Right'></e-column>
        <e-column field='ShipCity' headerText='Ship City' :allowResizing
= 'false' width='150' textAlign='Right'></e-column>
        <e-column field='ShipCountry' headerText='Ship Country'
minWidth='8' width='150'></e-column>
    </e-columns>
    </ejs-grid>
</div>
</template>
<script>
import Vue from "vue";
import { GridPlugin, Resize } from "@syncfusion/ej2-vue-grids";
import { data } from './datasource.js';
Vue.use(GridPlugin);
export default {
  data() {
    return {
      data: data
    };
  },
  provide: {
    grid: [Resize]
  }
}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/grid/column/autofit-cs1" %}

If any one of the column width is undefined, then the particular column will automatically adjust to fill the entire width of the grid table, even if you have enabled the `autoFit` property of grid.

#### AutoFit columns when changing column visibility using column chooser

In Syncfusion Grid, you can auto-fit columns when the column visibility is changed using the column chooser. This can be achieved by calling the `autoFitColumns` method in the `actionComplete` event. By using the `requestType` property in the event arguments, you can differentiate between different actions, and then call the `autoFitColumns` method when the request type is `columnState`.

Here's an example code snippet in Vue that demonstrates how to auto fit columns when changing column visibility using column chooser:

#### APP.VUE

```

<template>
  <div id="app">

```

```

<ejs-grid ref='grid' :dataSource="data" :allowResizing='true'
:toolbar='toolbarOptions' :showColumnChooser='true' height='400px'
width='600px' :autoFit='true' :actionComplete='actionComplete'>
  <e-columns>
    <e-column field='OrderID' headerText='Order ID' textAlign='Right'
width=100></e-column>
    <e-column field='CustomerID' headerText='Customer ID'
width=120></e-column>
    <e-column field='ShipName' headerText='Ship Name' width=80></e-
column>
    <e-column field='ShipAddress' headerText='Ship Address'
width=120></e-column>
    <e-column field='ShipCity' headerText='Ship City' width=100></e-
column>
  </e-columns>
</ejs-grid>
</div>
</template>
<script>
import Vue from "vue";
import { GridPlugin, Resize, ColumnChooser, Toolbar } from
"@syncfusion/ej2-vue-grids";
import { data } from './datasource.js';
Vue.use(GridPlugin);
export default {
  data() {
    return {
      data: data,
      toolbarOptions: ['ColumnChooser']
    };
  },
  methods:{
    actionComplete: function(args){
      if (args.requestType === 'columnstate') {
        let grid =this.$refs.grid.$el.ej2_instances[0];
        grid.autoFitColumns();
      }
    }
  },
  provide: {
    grid: [Resize, ColumnChooser, Toolbar]
  }
}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/grid/column/autofit-cs2" %}

### Locked columns

The Syncfusion Grid allows you to lock columns, which prevents them from being reordered and moves them to the first position. This functionality can be achieved by setting the [column.lockColumn](#) property to **true**, which locks the column and moves it to the first position in the grid.

Here's an example of how you can use the `lockColumn` property to lock a column in the Syncfusion Grid:

#### APP.VUE

```
<template>
  <div id="app">
    <ejs-grid :dataSource="data" :allowReordering='true'
:allowSelection='false' height='315px'>
      <e-columns>
        <e-column field='OrderID' headerText='Order ID'
textAlign='Right' width=90></e-column>
        <e-column field='CustomerID' headerText='Customer ID'
width=120></e-column>
        <e-column field='ShipCity' width=90 :lockColumn='true'
:customAttributes="customAttributes"></e-column>
        <e-column field='ShipName' width=120></e-column>
        <e-column field='ShipCountry' width=120></e-column>
        <e-column field='ShipRegion' width=120></e-column>
      </e-columns>
    </ejs-grid>
  </div>
</template>
<script>
import Vue from "vue";
import { GridPlugin, Reorder } from "@syncfusion/ej2-vue-grids";
import { data } from './datasource.js';
Vue.use(GridPlugin);
export default {
  data() {
    return {
      data: data,
      customAttributes : {class: 'customcss'}
    };
  },
  provide: {
    grid: [Reorder]
  }
}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
.e-grid .e-rowcell.customcss{
  background-color: #ecedee;
}
.e-grid .e-headercell.customcss{
  background-color: #ecedee;
}
</style>
```

{% previewsample "page.domainurl/code-snippet/grid/column/default-cs24" %}

#### Show or hide columns

The Syncfusion Grid control allows you to show or hide columns dynamically by using property or methods available in the grid. This feature can be useful when you want to customize the visibility of columns in the Grid based on the requirements.

### Using property

You can show or hide columns in the Vue Grid using the [visible](#) property of each column. By setting the **visible** property to **true** or **false**, you can control whether the column should be visible or hidden in the grid. Here's an example of how to show or hide a column in the Vue Grid using the visible property:

In the below example, the **ShipCity** column is defined with **visible** property set to **false**, which will hide the column in the rendered grid.

#### APP.VUE

```
<template>
  <div id="app">
    <ejs-grid :dataSource="data" height='315px'>
      <e-columns>
        <e-column field='OrderID' headerText='Order ID'
textAlign='Right' width=90></e-column>
        <e-column field='CustomerID' headerText='Customer ID'
width=120></e-column>
        <e-column field='Freight' headerText='Freight' textAlign='Right'
width=90></e-column>
        <e-column field='ShipCity' :visible=false headerText='Ship City'
width=120></e-column>
      </e-columns>
    </ejs-grid>
  </div>
</template>
<script>
import Vue from "vue";
import { GridPlugin } from "@syncfusion/ej2-vue-grids";
import { data } from './datasource.js';
Vue.use(GridPlugin);
export default {
  data() {
    return {
      data: data
    };
  }
}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-buttons/styles/material.css";
</style>
```

{% previewsample "page.domainurl/code-snippet/grid/column/default-cs23" %}

\* Hiding a column using the **visible** property only affects the UI representation of the grid. The data for the hidden column will still be available in the underlying data source, and can be accessed or modified programmatically.

\* When a column is hidden, its width is not included in the calculation of the total grid width.

\* To hide a column permanently, you can set its visible property to false in the column definition, or remove the column definition altogether.

### Using methods

You can also show or hide columns in the Vue Grid using the [showColumns](#) and [hideColumns](#) methods of the grid component. These methods allow you to show or hide columns based on either the `headerText` or the `field` of the column.

#### Based on header text

You can dynamically show or hide columns in the Grid based on the header text by invoking the `showColumns` or `hideColumns` methods. These methods take an array of column header texts as the first parameter, and the value `headerText` as the second parameter to specify that you are showing or hiding columns based on the header text.

Here's an example of how to show or hide a column based on the HeaderText in the Vue Grid:

#### APP.VUE

```
<template>
  <div id="app">
    <ejs-button id='show' cssClass="e-info" v-on:click.native='show'> Show
  </ejs-button>
    <ejs-button id='hide' cssClass="e-info" v-on:click.native='hide'> Hide
  </ejs-button>
    <ejs-grid ref='grid' id='grid' :dataSource="data" height='315px'>
      <e-columns>
        <e-column field='OrderID' headerText='Order ID'
textAlign='Right' width=90></e-column>
        <e-column field='CustomerID' headerText='Customer ID'
width=120></e-column>
        <e-column field='Freight' headerText='Freight' textAlign='Right'
width=90></e-column>
        <e-column field='ShipCity' headerText='Ship City'
width=120></e-column>
        <e-column field='ShipCountry' headerText='Ship Country'
width=120></e-column>
      </e-columns>
    </ejs-grid>
  </div>
</template>
<script>
import Vue from "vue";
import { GridPlugin } from "@syncfusion/ej2-vue-grids";
import { ButtonPlugin } from "@syncfusion/ej2-vue-buttons";
import { data } from './datasource.js';
Vue.use(GridPlugin);
Vue.use(ButtonPlugin);
export default {
  data() {
    return {
      data: data
    };
  },
  methods: {
    show: function () {
      let grid = this.$refs.grid.$el.ej2_instances[0];
      grid.showColumns('Customer ID', 'headerText');
    },
    hide: function () {
```

```

        let grid = this.$refs.grid.$el.ej2_instances[0];
        grid.hideColumns('Customer ID', 'headerText');
    }
}
</script>
<style>
    @import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/grid/column/default-cs32" %}

### Based on field

You can dynamically show or hide columns in the Grid using external buttons based on the field by invoking the `showColumns` or `hideColumns` methods. These methods take an array of column fields as the first parameter, and the value `field` as the second parameter to specify that you are showing or hiding columns based on the field.

Here's an example of how to show or hide a column based on the field in the Vue Grid:

### APP.VUE

```

<template>
    <div id="app">
        <ejs-button id='show' cssClass="e-info" v-on:click.native='show'> Show
    </ejs-button>
        <ejs-button id='hide' cssClass="e-info" v-on:click.native='hide'> Hide
    </ejs-button>
        <ejs-grid ref='grid' id='grid' :dataSource="data" height='315px'>
            <e-columns>
                <e-column field='OrderID' headerText='Order ID'
                    textAlign='Right' width=90></e-column>
                <e-column field='CustomerID' headerText='Customer ID'
                    width=120></e-column>
                <e-column field='Freight' headerText='Freight' textAlign='Right'
                    width=90></e-column>
                <e-column field='ShipCity' headerText='Ship City'
                    width=120></e-column>
                <e-column field='ShipCountry' headerText='Ship Country'
                    width=120></e-column>
            </e-columns>
        </ejs-grid>
    </div>
</template>
<script>
import Vue from "vue";
import { GridPlugin } from "@syncfusion/ej2-vue-grids";
import { ButtonPlugin } from "@syncfusion/ej2-vue-buttons";
import { data } from './datasource.js';
Vue.use(GridPlugin);
Vue.use(ButtonPlugin);
export default {
    data() {
        return {
            data: data

```

```

    };
  }
  methods: {
    show: function () {
      let grid = this.$refs.grid.$el.ej2_instances[0];
      grid.showColumns('CustomerID', 'field');
    },
    hide: function () {
      let grid = this.$refs.grid.$el.ej2_instances[0];
      grid.hideColumns('CustomerID', 'field');
    }
  }
}
</script>
<style>
  @import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/grid/column/default-cs33" %}

### Controlling grid actions

You can control various actions such as filtering, grouping, sorting, resizing, reordering, editing, and searching for specific columns in the Syncfusion Vue Grid using the following properties:

- [allowEditing](#): Enables or disables editing for a column.
- [allowFiltering](#): Enables or disables filtering for a column.
- [allowGrouping](#): Enables or disables grouping for a column.
- [allowSorting](#): Enables or disables sorting for a column.
- [allowReordering](#): Enables or disables reordering for a column.
- [allowResizing](#): Enables or disables resizing for a column.
- [allowSearching](#): Enables or disables searching for a column.

Here is an example code that demonstrates how to control grid actions for specific columns:

### APP.VUE

```

<template>
  <div id="app">
    <ejs-grid :dataSource='data' :allowReordering="true"
    :editSettings='editSettings' :allowSorting="true" :allowFiltering="true"
    :allowGrouping="true" height="220px">
      <e-columns>
        <e-column field='OrderID' headerText='Order ID'
        textAlign='Right' width=90 :allowReordering="false"></e-column>
        <e-column field='CustomerID' headerText='Customer ID'
        :allowEditing="false" width=120></e-column>
        <e-column field='Freight' headerText='Freight'
        textAlign='Right' format='C2' width=90 :allowFiltering="false"></e-column>
        <e-column field='OrderDate' headerText='Order Date'
        textAlign='Right' type='date' format='yMd' :allowGrouping="false" width=120
        :allowSorting="false"></e-column>
      </e-columns>
    </ejs-grid>
  </div>

```

```

</template>
<script>
import Vue from "vue";
import { GridPlugin, Sort, Group, Filter, Reorder, Edit } from
"@syncfusion/ej2-vue-grids";
import { data } from './datasource.js';
Vue.use(GridPlugin);
export default {
  data() {
    return {
      data: data,
      editSettings: { allowEditing: true, allowAdding: true, allowDeleting:
true },
    };
  },
  provide: {
    grid: [Sort, Group, Filter, Reorder, Edit]
  }
}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/grid/column/default-cs25" %}

### *Customize column styles*

Customizing the grid column styles allows you to modify the appearance of columns in the Grid control to meet your design requirements. You can customize the font, background color, and other styles of the columns. To customize the columns styles in the grid, you can use grid event, css, property or method support.

For more information check on this [documentation](#).

### *Manipulating columns*

The Syncfusion Grid for Vue provides powerful features for manipulating columns in a grid. This section explains how to access columns, update column definitions, and add/remove columns using Syncfusion Grid properties, methods, and events.

### *Accessing columns*

To access columns in the Syncfusion Grid, you can use the following methods in the grid.

- [\*\*getColumns:\*\*](#)

This method returns the array of columns defined in the grid.

```
`ts
```

```
let columns = grid.getColumns();
```

```
,
```

- [\*\*getColumnByField:\*\*](#)

This method returns the column object that matches the specified field name.



```
`ts
```

```
let column = grid.getColumnByField('ProductName');
```

```
,
```

- [getColumnByUid:](#)

This method returns the column object that matches the specified UID.

```
`ts
```

```
let column = grid.getColumnByUid();
```

```
,
```

- [getVisibleColumns:](#)

This method returns the array of visible columns.

```
`ts
```

```
let visibleColumns = grid.getVisibleColumns();
```

```
,
```

- [getForeignKeyColumns:](#)

This method returns the array of foreignkey columns.

```
`ts
```

```
let foreignKeyColumns = grid.getForeignKeyColumns();
```

```
,
```

- [getColumnFieldNames](#)

This method returns an array of field names of all the columns in the Grid.

```
`ts
```

```
let fieldNames = grid.getColumnFieldNames()
```

```
,
```

For a complete list of column methods and properties, refer to this [section](#).

#### Updating column definitions

You can update the column definitions in the Grid using the [columns](#) property. You can modify the properties of the column objects in the columns array to update the columns dynamically. For example, you can change the headerText, width, visible, and other properties of a column to update its appearance and behavior in the grid and then call the [refreshColumns](#) method to apply the changes to the grid.

#### **APP.VUE**

```

<template>
  <div id="app">
    <ejs-button @click.native="updateColumns"> Update Columns </ejs-
button>
    <ejs-grid ref='grid' :dataSource='data' :height='280'>
      <e-columns>
        <e-column field='OrderID' headerText='Order ID'
textAlign='Right' width=90></e-column>
        <e-column field='CustomerID' headerText='Customer ID'
width=120></e-column>
        <e-column field='Freight' headerText='Freight'
textAlign='Right' format='C2' width=90></e-column>
        <e-column field='OrderDate' headerText='Order Date'
textAlign='Right' format='yMd' width=120 type='date'></e-column>
      </e-columns>
    </ejs-grid>
  </div>
</template>
<script>
import Vue from "vue";
import { GridPlugin } from "@syncfusion/ej2-vue-grids";
import { ButtonPlugin } from "@syncfusion/ej2-vue-buttons";
import { data } from './datasource.js';
Vue.use(GridPlugin);
Vue.use(ButtonPlugin);
export default {
  data() {
    return {
      data: data
    };
  },
  methods: {
    updateColumns: function() {
      let grid = this.$refs.grid.$el.ej2_instances[0];
      grid.columns[0].textAlign = 'Center';
      grid.columns[0].width = '100';
      grid.columns[2].visible = false;
      grid.columns[1].customAttributes = { class: 'customcss' };
      // Applying changes to the grid
      grid.refreshColumns();
    },
  }
}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/grid/column/default-cs26" %}

### [Adding/removing columns](#)

The Grid component allows you to dynamically add or remove columns to and from the grid using the [columns](#) property, which can be accessed through the instance of the Grid.

To add a new column to the Grid, you can directly **push** the new column object to the columns property. To remove a column from the Grid, you can use the **pop** method, which removes the last

element from the columns array of the Grid. Alternatively, you can use the splice method to remove a specific column from the columns array.

Here's an example of how you can add and remove a column from the grid:

#### APP.VUE

```
<template>
  <div id="app">
    <ejs-button @click.native="AddColumns"> Add Columns </ejs-button>
    <ejs-button @click.native="DeleteColumns"> Delete Columns </ejs-button>
    <ejs-grid ref='grid' :dataSource='data' :height='280'>
      <e-columns>
        <e-column field='OrderID' headerText='Order ID'
textAlign='Right' width=90></e-column>
        <e-column field='CustomerID' headerText='Customer ID'
width=120></e-column>
        <e-column field='Freight' headerText='Freight'
textAlign='Right' format='C2' width=90></e-column>
        <e-column field='OrderDate' headerText='Order Date'
textAlign='Right' format='yMd' width=120 type='date'></e-column>
      </e-columns>
    </ejs-grid>
  </div>
</template>
<script>
import Vue from "vue";
import { GridPlugin } from "@syncfusion/ej2-vue-grids";
import { ButtonPlugin } from "@syncfusion/ej2-vue-buttons";
import { data } from './datasource.js';
Vue.use(GridPlugin);
Vue.use(ButtonPlugin);
export default {
  data() {
    return {
      data: data
    };
  },
  methods: {
    AddColumns: function () {
      let grid = this.$refs.grid.$el.ej2_instances[0];
      var newColumns = [
        { field: 'ShipCity', headerText: 'Ship City', width: 120 },
        { field: 'ShipCountry', headerText: 'Ship Country', width: 120 },
      ];
      newColumns.forEach((col) => {
        grid.columns.push(col);
      });
      grid.refreshColumns();
    },
    DeleteColumns: function () {
      let grid = this.$refs.grid.$el.ej2_instances[0];
      grid.columns.pop();
      grid.refreshColumns();
    }
  }
}
```

```

</script>
<style>
  @import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/grid/column/default-cs34" %}

### How to refresh columns

You can use the [refreshColumns](#) method of the Syncfusion Grid to refresh the columns in the grid. This method can be used when you need to update the grid columns dynamically based on user actions or data changes.

```
`ts
```

```
grid.refreshColumns();
```

```
,
```

### Responsive columns

The Syncfusion Vue Grid provides a built-in feature to toggle the visibility of columns based on media queries using the [hideAtMedia](#) property of the column object. The `hideAtMedia` accepts valid [Media Queries](#).

In this example, we have a Grid that displays data with three columns: **Order ID**, **Customer ID**, and **Freight**. We have set the `hideAtMedia` property of the **OrderID** column to (min-width: 700px) which means that this column will be hidden when the browser screen width is less than or equal to 700px.

### APP.VUE

```

<template>
  <div id="app">
    <ejs-grid :dataSource='data' height='315px'>
      <e-columns>
        <e-column field='OrderID' headerText='Order ID'
textAlign='Right' width=120 hideAtMedia='(min-width: 700px)'>
        </e-column> // column visibility hide when browser screen
width lessthan 700px;
        <e-column field='CustomerID' headerText='Customer ID'
width=140 hideAtMedia='(max-width: 700px)'>
        </e-column> // column Visibility show when browser screen
width 500px or less;
        <e-column field='Freight' headerText='Freight'
textAlign='Right' format='C' width=120
hideAtMedia='(min-width: 500px)'>
        </e-column> // column visibility hide when browser screen
width lessthan 500px;
        <e-column field='OrderDate' headerText='Order Date'
textAlign='Right' format='yMd' type='date' width=140>
        </e-column> // it always shown
      </e-columns>
    </ejs-grid>
  </div>
</template>
<script>
import Vue from "vue";
import { GridPlugin } from "@syncfusion/ej2-vue-grids";

```

```
import { data } from './datasource.js';
Vue.use(GridPlugin);
export default {
  data() {
    return {
      data: data
    };
  }
}
</script>
<style>
  @import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
</style>
```

{% previewsample "page.domainurl/code-snippet/grid/column/default-cs30" %}

*See also*

- [Group Column by Format](#)
- [How to set complex column as Foreignkey column](#)
- [Complex Data Binding with list of Array Of Objects](#)
- [Custom tooltip for the header in Vue Grid](#)
- [Display only multiple field name in one column in Vue Grid](#)
- [How to change the data source or columns dynamically](#)

Auto generated columns in Vue Grid component

The [columns](#) are automatically generated when [columns](#) declaration is empty or undefined while initializing the grid. All the columns in the [dataSource](#) are bound as grid columns.

#### **APP.VUE**

```
<template>
  <div id="app">
    <ejs-grid :dataSource="data"> </ejs-grid>
  </div>
</template>
<script>
import Vue from 'vue';
import { GridPlugin } from '@syncfusion/ej2-vue-grids';
Vue.use(GridPlugin);
export default {
  data () {
    return {
      data: [
        { OrderID: 10248, CustomerID: 'VINET', EmployeeID: 5 },
        { OrderID: 10249, CustomerID: 'TOMSP', EmployeeID: 6 },
        { OrderID: 10250, CustomerID: 'HANAR', EmployeeID: 4 }
      ]
    }
  }
}
</script>
<style>
  @import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
</style>
```

```
{% previewsample "page.domainurl/code-snippet/grid/column/default-cs2" %}
```

When the columns are auto-generated then the column [type](#) will be determined from the first record of the [dataSource](#).

#### *How to set isPrimaryKey for auto generated columns when editing is enabled*

Primary key can be defined in the declaration of column object of the grid. When we didn't declare the columns, the grid will generate the columns automatically. For these auto generated columns, you can set `isPrimaryKey` column property as true by using the following ways,

If Primary key "column index" is known then refer the following code example

#### **APP.VUE**

```
<template>
  <div id="app">
    <ejs-grid ref='grid' :dataSource="data" :editSettings='editSettings'
    :dataBound='dataBound'> </ejs-grid>
  </div>
</template>
<script>
import Vue from 'vue';
import { GridPlugin, Edit } from '@syncfusion/ej2-vue-grids';
Vue.use(GridPlugin);
export default {
  data () {
    return {
      data: [
        { OrderID: 10248, CustomerID: 'VINET', EmployeeID: 5 },
        { OrderID: 10249, CustomerID: 'TOMSP', EmployeeID: 6 },
        { OrderID: 10250, CustomerID: 'HANAR', EmployeeID: 4 }],
      editSettings: { allowEditing: true, allowAdding: true, allowDeleting:
true },
    }
  },
  provide: {
    grid: [Edit]
  },
  methods: {
    dataBound: function() {
      this.$refs.grid.ej2Instances.columns[0].isPrimaryKey = true;
    }
  }
}
</script>
<style>
  @import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
</style>
```

```
{% previewsample "page.domainurl/code-snippet/grid/column/default-cs3" %}
```

If Primary key "column fieldname" is known then you can get the column by using `var column = grid.getColumnByField('OrderID')` and then set primary key by defining `column.isPrimaryKey = 'true'`

### Set column options to auto generated columns

You can set column options such as **format**, **width** to the auto generated columns by using **dataBound** event of the grid.

In the below example, **width** is set for **OrderID** column, **date** type is set for **OrderDate** column and **numeric** type is set for **Freight** column.

#### APP.VUE

```
<template>
  <div id="app">
    <ejs-grid ref='grid' :dataSource="data" :dataBound='dataBound'> </ejs-
grid>
  </div>
</template>
<script>
import Vue from 'vue';
import { GridPlugin } from '@syncfusion/ej2-vue-grids';
Vue.use(GridPlugin);
export default {
  data () {
    return {
      data: [
        { OrderID: 10248, CustomerID: 'VINET', Freight: 32.3800, OrderDate:
"1996-07-02T00:00:00.000Z" },
        { OrderID: 10249, CustomerID: 'TOMSP', Freight: 32.3800, OrderDate:
"1996-07-19T00:00:00.000Z" },
        { OrderID: 10250, CustomerID: 'HANAR', Freight: 32.3800, OrderDate:
"1996-07-22T00:00:00.000Z" } ]
    },
    methods: {
      dataBound: function() {
        for (var i = 0; i < this.$refs.grid.ej2Instances.columns.length;
i++) {
          this.$refs.grid.ej2Instances.columns[0].width = 120;
          if(this.$refs.grid.ej2Instances.columns[i].field ===
"OrderDate"){
            this.$refs.grid.ej2Instances.columns[i].type="date";
          }
          if (this.$refs.grid.ej2Instances.columns[i].type === "date") {
            this.$refs.grid.ej2Instances.columns[i].format = { type:
"date", format: "dd/MM/yyyy" };
          }
          this.$refs.grid.ej2Instances.columns[2].format = "P2";
          this.$refs.grid.ej2Instances.refreshColumns();
        }
      }
    }
  }
</script>
<style>
  @import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
</style>
```

{% previewsample "page.domainurl/code-snippet/grid/column/default-cs4" %}

### *How to auto-generate columns from complex column fields to flat columns*

By default, the auto-generated columns will create multiple column fields when grid has complex datasource. Hereby using [ValueAccessor](#) property, complex columns data are combined and generated in a single column.

In the below example, we set the **ValueAccessor** property to those columns whose field type is **Object**.

#### **APP.VUE**

```
<template>
  <div id="app">
    <ejs-grid
      :dataSource="data"
      id="Grid"
      ref="grid"
      :pageSettings="pageSettings"
      :allowPaging="true"
      :editSettings="editSettings"
      :toolbar="toolbar"
      :dataBound="dataBound"
    ></ejs-grid>
  </div>
</template>
<script>
import Vue from "vue";
import { GridPlugin, Edit, Toolbar, Page } from "@syncfusion/ej2-vue-grids";
import { data } from "../datasource.js";
Vue.use(GridPlugin);
export default {
  data() {
    return {
      data: data,
      flag: true,
      editSettings: {
        allowEditing: true,
        allowAdding: true,
        allowDeleting: true
      },
      pageSettings: { pageSize: 5 },
      toolbar: ["Add", "Edit", "Delete", "Update", "Cancel"]
    };
  },
  methods: {
    dataBound: function() {
      if (this.flag) {
        this.flag = false;
        var cols = Object.keys(this.$refs.grid.getCurrentViewRecords()[0]);
        var length = cols.length;
        var col = [];
        for (var i = 0; i < length; i++) {
          var field = cols[i];
          var obj = {};
          obj["field"] = field;
          var mon = this.$refs.grid.getCurrentViewRecords()[i][field];
          var type = typeof mon;
          if (type === "object") {
            obj["valueAccessor"] = (field, data, column) => {
```



```

        return (
            data[field].Name +
            " , " +
            data[field].Unit +
            " , " +
            data[field].VSetMax
        );
    };
    }
    col.push(obj);
}
this.$refs.grid.setProperties({ columns: col });
}
},
provide: {
    grid: [Edit, Toolbar, Page]
}
};
</script>
<style>
@import "https://cdn.syncfusion.com/ej2/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/grid/column/generate-cs1" %}

## Headers in Vue Grid component

### Header text

By default, column header title is displayed from column [field](#) value. To override the default header title, you have to define the [headerText](#) value.

### APP.VUE

```

<template>
  <div id="app">
    <ejs-grid :dataSource="data" height='315px'>
      <e-columns>
        <e-column field='OrderID' headerText='Order ID'
textAlign='Right' width=90></e-column>
        <e-column field='CustomerID' headerText='Customer ID'
width=120></e-column>
        <e-column field='Freight' headerText='Freight' textAlign='Right'
width=90></e-column>
        <e-column field='ShipCity' headerText='Ship City' width=120></e-
column>
      </e-columns>
    </ejs-grid>
  </div>
</template>
<script>
import Vue from "vue";
import { GridPlugin } from "@syncfusion/ej2-vue-grids";
import { data } from './datasource.js';
Vue.use(GridPlugin);
export default {
  data() {

```

```

    return {
      data: data
    };
  }
}
</script>
<style>
  @import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/grid/column/default-cs28" %}

If both the [field](#) and [headerText](#) are not defined in the column, the column renders with “empty” header text.

#### Header template

Customize the header element using the [headerTemplate](#) property. In this demo, the custom element is rendered for both the CustomerID and OrderDate column headers.

#### APP.VUE

```

<template>
  <div id="app">
    <ejs-grid ref="grid" :dataSource="data" :allowPaging="true"
height="273px">
      <e-columns>
        <e-column field="OrderID" headerText="Order ID" width="120"
textAlign="Right"></e-column>
        <e-column field="CustomerID" headerText="Customer ID"
:headerTemplate="orderTemplate" width="150"></e-column>
        <e-column field="ShipCity" headerText="Ship City" width="150"></e-
column>
        <e-column field="OrderDate" headerText="Order Date" width="135"
format="yMd" type="date" :headerTemplate="dateTemplate"
textAlign="Right"></e-column>
        <e-column field="ShipName" headerText="Ship Name" width="150"></e-
column>
      </e-columns>
    </ejs-grid>
  </div>
</template>
<script>
import Vue from "vue";
import {
  GridPlugin,
  Page,
  Filter,
  Sort,
  Toolbar,
  Edit,
} from "@syncfusion/ej2-vue-grids";
import { data } from "../datasource.js";
Vue.use(GridPlugin);
export default {
  data() {
    return {

```

```

        data: data,
        dateTemplate: function () {
            return {
                template: Vue.component("columnTemplate", {
                    template: `<div><span class="e-icon-calender e-icons
headericon"></span> Order Date</div>`,
                    data: function () {
                        return {
                            data: {},
                        };
                    },
                    computed: {},
                }),
            };
        },
        orderTemplate: function () {
            return {
                template: Vue.component("columnTemplate", {
                    template: `<div><span class="e-icon-userlogin e-icons
employee"></span> Customer ID</div>`,
                    data: function () {
                        return {
                            data: {},
                        };
                    },
                    computed: {},
                }),
            };
        },
        methods: {},
        provide: {
            grid: [Page, Filter, Sort, Edit, Toolbar],
        },
    };
</script>
<style>
@import "https://cdn.syncfusion.com/ej2/material.css";
@import "./style.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/grid/column/default-cs29" %}

#### *Change the orientation of header text*

You can change the orientation of the header text by using the [customAttributes](#) property.

To change the Orientation of Header Text, Ensure the following steps:

**Step 1:** Create a css class with orientation style for grid header cell.

`

```

.orientationcss .e-headercelldiv {
transform: rotate(90deg);
}

```

**Step 2:** Add the custom css class to particular column by using [customAttributes](#) property.

```
<e-column field='Freight' headerText='Freight' textAlign='Center' format='C2'
:customAttributes='customAttributes' width=80></e-column>
```

**Step 3:** Resize the header cell height by using the following code.

```
`ts
setHeaderHeight(args) {
let textWidth = document.querySelector(".orientationcss > div").scrollWidth;//Obtain the width of the
headerText content.
let headerCell = document.querySelectorAll(".e-headercell");
for(let i = 0; i < headerCell.length; i++) {
(headerCell.item(i)).style.height = textWidth + 'px'; //Assign the obtained textWidth as the height of the
headerCell.
}
}
```

### APP.VUE

```
<template>
  <div id="app">
    <ejs-grid :dataSource='data' :created='setHeaderHeight'
height='240px' >
      <e-columns>
        <e-column field='OrderID' headerText='Order ID'
textAlign='Right' width=100></e-column>
        <e-column field='CustomerID' headerText='Customer ID'
width=120></e-column>
        <e-column field='Freight' headerText='Freight'
textAlign='Center' format='C2' :customAttributes='customAttributes'
width=80></e-column>
        <e-column field='ShipCity' headerText='Ship City'
width=100></e-column>
      </e-columns>
    </ejs-grid>
  </div>
</template>
<script>
import Vue from "vue";
import { GridPlugin } from "@syncfusion/ej2-vue-grids";
import { data } from './datasource.js';
Vue.use(GridPlugin);
export default {
  data: () => {
    return {
```

```

        data: data,
        customAttributes : {class : 'orientationcss'},
    };
},
methods: {
    setHeaderHeight: function(args) {
        let textWidth = document.querySelector(".orientationcss >


").scrollWidth; //Obtain the width of the headerText content.
        let headerCell = document.querySelectorAll(".e-headercell");
        for (let i = 0; i < headerCell.length; i++) {
            (headerCell.item(i)).style.height = textWidth + 'px'; //Assign the
            obtained textWidth as the height of the headerCell.
        }
    }
}
}
</script>
<style>
    @import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
    .orientationcss .e-headercelldiv {
        transform: rotate(90deg);
        padding-top: 5px;
    }
</style>


```

{% previewsample "page.domainurl/code-snippet/grid/how-to/default-cs4" %}

### Column template in Vue Grid component

#### Render image in a column

The column [template](#) has options to display custom element instead of a field value in the column.

The `template` property should be a function which returns an object. The object should contain a Vue component which should be assigned to the `template` property. The grid will look for the template property and render it as new Vue instance.

### APP.VUE

```

<template>
    <div id="app">
        <ejs-grid :dataSource="data" height=310>
            <e-columns>
                <e-column headerText='Employee Image' width='150'
                textAlign='Center' :template='cTemplate'></e-column>
                <e-column field='EmployeeID' headerText='Employee ID'
                width='125' textAlign='Right'></e-column>
                <e-column field='FirstName' headerText='Name'
                width='120'></e-column>
                <e-column field='Title' headerText='Title' width='170'></e-
                column>
            </e-columns>
        </ejs-grid>
    </div>
</template>
<script>
import Vue from "vue";
import { GridPlugin } from "@syncfusion/ej2-vue-grids";

```

```

import { employeeData } from "../datasource.js";
Vue.use(GridPlugin);
export default {
  data: () => {
    return {
      data: employeeData,
      cTemplate: function () {
        return { template : Vue.component('columnTemplate',{
          template: `<div class="image">
            
          </div>`,
          data: function() {
            return {
              data: {}
            }
          },
          computed: {
            image: function() {
              return '../images/' + this.data.EmployeeID + '.png';
            },
            altImage: function() {
              return this.data.EmployeeID;
            }
          }
        })
      }
    }
  }
};
}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
.image img {
  height: 55px;
  width: 55px;
  border-radius: 50px;
  box-shadow: inset 0 0 1px #e0e0e0, inset 0 0 14px rgba(0,0,0,0.2);
}
</style>

```

{% previewsample "page.domainurl/code-snippet/grid/column/template-cs1" %}

*Render other components in a column*

You can render any component in a grid column using the [template](#) property.

#### APP.VUE

```

<template>
  <div id="app">
    <ejs-grid :dataSource='data' height='315px' >
      <e-columns>
        <e-column headerText='Order Status'
:template="dropdownTemplate" width=200></e-column>
        <e-column field='OrderID' headerText='Order ID'
textAlign='Right' width=90></e-column>
      </e-columns>
    </ejs-grid>
  </div>
</template>

```

```

        <e-column field='CustomerID' headerText='Customer ID'
width=120></e-column>
        <e-column field='Freight' headerText='Freight'
textAlign='Right' format='C2' width=90></e-column>
        <e-column field='ShipName' headerText='Ship Name'
width=120></e-column>
    </e-columns>
</ejs-grid>
</div>
</template>
<script>
import Vue from "vue";
import { GridPlugin } from "@syncfusion/ej2-vue-grids";
import { DropDownListPlugin } from "@syncfusion/ej2-vue-dropdowns";
import { data } from './datasource.js';
Vue.use(GridPlugin);
Vue.use(DropDownListPlugin);
export default {
  data() {
    return {
      data: data,
      dropdownTemplate: function () {
        return {
          template: Vue.component('bindDropdown', {
            template: `<ejs-dropdownlist id='dropdownlist'
:dataSource='dropData'> </ejs-dropdownlist>`,
            data() {
              return {
                dropData: ['Order Placed', 'Processing', 'Delivered']
              }
            }
          })
        }
      })
    }
  }
};
}
</script>
<style>
  @import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/grid/how-to/dropdown-cs1" %}

### Using condition template

You can render the template elements based on condition.

In the following code, checkbox is rendered based on **Discontinued** field value.

### APP.VUE

```

<template>
  <div id="app">
    <ejs-grid :dataSource="data" height=310>
      <e-columns>
        <e-column headerText='Discontinued' width='150'
textAlign='Center' :template='cTemplate'></e-column>

```

```

        <e-column field='ProductID' headerText='Product ID'
width='125'></e-column>
        <e-column field='ProductName' headerText='Name'
width='120'></e-column>
        <e-column field='SupplierID' headerText='Supplier ID'
width='170'></e-column>
    </e-columns>
</ejs-grid>
</div>
</template>
<script>
import Vue from "vue";
import { GridPlugin } from "@syncfusion/ej2-vue-grids";
import { productData } from "../datasource.js";
Vue.use(GridPlugin);
export default {
  data: () => {
    return {
      data: productData,
      cTemplate: function () {
        return { template : Vue.component('columnTemplate',{
          template: `<div v-if=cData class="template_checkbox">
            <input type="checkbox" checked />
          </div>
          <div v-else class="template_checkbox">
            <input type="checkbox" />
          </div>`,
          data: function() {
            return {
              data: {}
            }
          },
          computed: {
            cData: function() {
              return this.data.Discontinued;
            }
          }
        })}
      }
    };
  }
}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/grid/column/condition-template-cs1" %}

[How to get the row object by clicking on the template element](#)

You can get the row object without selecting the row and achieve it using the column template feature and the `getRowObjectFromUID` method of the Grid.



In the following sample, the button element is rendered in the Employee Data column. By clicking the button, you can get the row object using the `getRowObjectFromUID` method of the Grid and display it in the console.

#### APP.VUE

```
<template>
  <div id="app">
    <ejs-grid ref="grid" :dataSource="data" height=315 :recordClick =
    'recordClick'>
      <e-columns>
        <e-column headerText='Employee Data' width='150'
        textAlign='Right' :template='cTemplate' isPrimaryKey='true'></e-column>
        <e-column field='EmployeeID' headerText='Employee ID'
        width='130' textAlign='Right'></e-column>
        <e-column field='FirstName' headerText='Name'
        width='120'></e-column>
        <e-column field='Title' headerText='Title' width='170'></e-
        column>
      </e-columns>
    </ejs-grid>
  </div>
</template>
<script>
import Vue from "vue";
import { GridPlugin, RecordClickEventArgs } from "@syncfusion/ej2-vue-
grids";
import { employeeData } from "../datasource.js";
import { closest } from "@syncfusion/ej2-base";
Vue.use(GridPlugin);
export default {
  data: () => {
    return {
      data: employeeData,
      cTemplate: function () {
        return { template : Vue.component('columnTemplate',{
          template: `<div class="image">
            <button class="empData">Employee Data</button>
          </div>`
        }}}
      }
    };
  },
  methods: {
    recordClick(args) {
      if (args.target.classList.contains('empData')) {
        var rowObj =
        this.$refs.grid.ej2Instances.getRowObjectFromUID(closest(args.target, '.e-
        row').getAttribute('data-uid')
        );
        console.log(rowObj);
      }
    }
  },
}
```

```
@import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
</style>
```

{% previewsample "page.domainurl/code-snippet/grid/column/template-cs2" %}

### Complex data binding in Vue Grid component

You can achieve complex data binding in the grid by using the dot(.) operator in the [column.field](#).

#### APP.VUE

```
<template>
  <div id="app">
    <ejs-grid :dataSource='data' :height='315'>
      <e-columns>
        <e-column field='EmployeeID' headerText='Employee ID'
textAlign='Right' width=120></e-column>
        <e-column field='Name.FirstName' headerText='First Name'
width=120></e-column>
        <e-column field='Name.LastName' headerText='Last Name'
width=120></e-column>
        <e-column field='Title' headerText='Title' width=150></e-
column>
      </e-columns>
    </ejs-grid>
  </div>
</template>
<script>
import Vue from "vue";
import { GridPlugin } from "@syncfusion/ej2-vue-grids";
import { data } from './datasource.js';
Vue.use(GridPlugin);
export default {
  data() {
    return {
      data: data
    };
  }
}
</script>
<style>
  @import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
</style>
```

{% previewsample "page.domainurl/code-snippet/grid/column/complex-cs1" %}

### Foreign key column in Vue Grid component

Foreign key column can be enabled by using [column.dataSource](#), [column.foreignKeyField](#) and [column.foreignKeyValue](#) properties.

- [column.dataSource](#) - Defines the foreign data.
- [column.foreignKeyField](#) - Defines the mapping column name to the foreign data.
- [column.foreignKeyValue](#) - Defines the display field from the foreign data.

In the following example, **Employee Name** is a foreign column which shows **FirstName** column from foreign data.

#### APP.VUE

```
<template>
  <div id="app">
    <ejs-grid :dataSource='data' height='315'>
      <e-columns>
        <e-column field='OrderID' headerText='Order ID'
textAlign='Right' width=100></e-column>
        <e-column field='EmployeeID' headerText='Employee Name'
width=120 foreignKeyValue='FirstName' :dataSource='employeeData'></e-column>
        <e-column field='Freight' headerText='Freight'
textAlign='Right' width=80></e-column>
        <e-column field='ShipCity' headerText='Ship City' width=130
></e-column>
      </e-columns>
    </ejs-grid>
  </div>
</template>
<script>
import Vue from "vue";
import { GridPlugin, ForeignKey } from "@syncfusion/ej2-vue-grids";
import { data, employeeData } from '../datasource.js';
Vue.use(GridPlugin);
export default {
  data() {
    return {
      data: data,
      employeeData: employeeData
    };
  },
  provide: {
    grid: [ForeignKey]
  }
}
</script>
<style>
  @import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
</style>
```

{% previewsample "page.domainurl/code-snippet/grid/column/foreigncolumn-cs1" %}

\* For remote data, the sorting and grouping is done based on [column.foreignKeyField](#) instead of [column.foreignKeyValue](#).

\* If [column.foreignKeyField](#) is not defined, then the column uses [column.field](#).

*Use edit template in foreignkey column*

By default, foreign key column uses dropdown component for editing. You can render other than the dropdown by using the [column.edit](#) property.

The following example demonstrates the way of using edit template in foreign column.

In the following example, The **Employee Name** is a foreign key column and while editing, AutoComplete component is rendered instead of DropDownList.

#### APP.VUE

```
<template>
  <div id="app">
    <ejs-grid ref='grid' :dataSource='data' :editSettings='editoption'
:Toolbar='toolbar' height='270px' >
      <e-columns>
        <e-column field='OrderID' headerText='Order ID'
textAlign='Right' width=100></e-column>
        <e-column field='EmployeeID' headerText='Employee Name'
:dataSource='employeeData' foreignKeyValue='FirstName' :edit='edit'
width=120></e-column>
        <e-column field='Freight' headerText='Freight'
textAlign='Center' format='C2' width=80></e-column>
        <e-column field='ShipCity' headerText='Ship City'
width=130></e-column>
      </e-columns>
    </ejs-grid>
  </div>
</template>
<script>
import Vue from "vue";
import { createElement } from '@syncfusion/ej2-base';
import { GridPlugin, Edit, Toolbar, ForeignKey } from "@syncfusion/ej2-vue-grids";
import { AutoComplete } from "@syncfusion/ej2-dropdowns";
import { DataManager, Query } from '@syncfusion/ej2-data';
import { data, fEmployeeData } from './datasource.js';
Vue.use(GridPlugin);
export default {
  data: () => {
    return {
      data: data,
      employeeData: fEmployeeData,
      toolbar: ['Add', 'Edit', 'Delete', 'Update', 'Cancel'],
      editoption: { allowEditing: true },
      edit: {
        create: () => { // to create input element
          return createElement('input');
        },
        read: () => { // return edited value to update data source
          let value = new DataManager(fEmployeeData).executeLocal(new
Query().where('FirstName', 'equal', this.autoComplete.value));
          return value.length && value[0]['EmployeeID']; // to convert
foreign key value to local value.
        },
        destroy: () => { // to destroy the custom component.
          this.autoComplete.destroy();
        },
        write: (args) => { // to show the value for date picker
          this.autoComplete = new AutoComplete({
            dataSource: fEmployeeData,
            fields: { value: args.column.foreignKeyValue },
            value: args.foreignKeyData[0][args.column.foreignKeyValue]
          });
        }
      }
    };
  }
};
```

```

        });
        this.autoComplete.appendTo(args.element);
    },
    },
    };
},
provide: {
    grid: [Edit, ForeignKey]
},
}
</script>
<style>
    @import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/grid/how-to/foreignKey-cs1" %}

#### *Customize filter UI in foreignkey column*

You can create your own filtering UI by using [column.filter](#) property. The following example demonstrates the way to create a custom filtering UI in the foreign column.

In the following example, The **Employee Name** is a foreign key column. DropDownList is rendered using Filter UI.

#### **APP.VUE**

```

<template>
    <div id="app">
        <ejs-grid ref='grid' :dataSource='data' :allowFiltering='true'
        :filterSettings='filteroption' height='270px' >
            <e-columns>
                <e-column field='OrderID' headerText='Order ID'
                textAlign='Right' width=100></e-column>
                <e-column field='EmployeeID' headerText='Employee Name'
                :dataSource='employeeData' foreignKeyValue='FirstName' :filter='filter'
                width=120></e-column>
                <e-column field='Freight' headerText='Freight'
                textAlign='Center' format='C2' width=80></e-column>
                <e-column field='ShipCity' headerText='Ship City'
                width=130></e-column>
            </e-columns>
        </ejs-grid>
    </div>
</template>
<script>
import Vue from "vue";
import { createElement } from '@syncfusion/ej2-base';
import { GridPlugin, Edit, Toolbar, ForeignKey, Filter } from
"@syncfusion/ej2-vue-grids";
import { DropDownList } from "@syncfusion/ej2-dropdowns";
import { DataManager } from '@syncfusion/ej2-data';
import { data, fEmployeeData } from './datasource.js';
let dropInstance;
Vue.use(GridPlugin);
export default {
    data: () => {

```

```

    return {
      data: data,
      employeeData: fEmployeeData,
      filteroption: { type: 'Menu' },
      filter: {
        ui: {
          create: (args) => {
            let flValInput = createElement('input', { className: 'flm-
input' });
            args.target.appendChild(flValInput);
            dropInstance = new DropDownList({
              dataSource: new DataManager(fEmployeeData),
              fields: { text: 'FirstName', value: 'EmployeeID' },
              placeholder: 'Select a value',
              popupHeight: '200px'
            });
            dropInstance.appendTo(flValInput);
          },
          write: (args) => {
            dropInstance.text = args.filteredValue || '';
          },
          read: (args) => {
            args.fltrObj.filterByColumn(args.column.field,
args.operator, dropInstance.text);
          }
        }
      },
      provide: {
        grid: [Filter, ForeignKey, Edit, Toolbar]
      }
    }
  }
</script>
<style>
  @import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/grid/how-to/foreignKey-cs2" %}

#### [Perform aggregation in foreignkey column](#)

Default aggregations are not supported in a foreign key column. You can achieve aggregation for the foreign key column by using custom the aggregates. The following example demonstrates the way to achieve aggregation in foreign key column.

In the following example, The **Employee Name** is a foreign key column and the aggregation for the foreign column was calculated in customAggregateFn.

#### **APP.VUE**

```

<template>
  <div id="app">
    <ejs-grid ref='grid' :dataSource='data' :allowFiltering='true'
height='260px' >
      <e-columns>

```

```

        <e-column field='OrderID' headerText='Order ID'
textAlign='Right' width=100></e-column>
        <e-column field='EmployeeID' headerText='Employee Name'
:dataSource='employeeData' foreignKeyValue='FirstName' width=120></e-column>
        <e-column field='Freight' headerText='Freight'
textAlign='Center' format='C2' width=80></e-column>
        <e-column field='ShipCity' headerText='Ship City'
width=130></e-column>
    </e-columns>
    <e-aggregates>
        <e-aggregate>
            <e-columns>
                <e-column field="EmployeeID" type="Custom"
:customAggregate='customAggregateFn' :footerTemplate='footerTemplate'></e-
column>
            </e-columns>
        </e-aggregate>
    </e-aggregates>
</ejs-grid>
</div>
</template>
<script>
import Vue from "vue";
import { GridPlugin, Aggregate ,ForeignKey, Filter } from "@syncfusion/ej2-
vue-grids";
import { getValue } from "@syncfusion/ej2-base";
import { getForeignData } from "@syncfusion/ej2-grids";
import { data,fEmployeeData } from './datasource.js';
Vue.use(GridPlugin);
export default {
    data: () => {
        return {
            data: data,
            employeeData: fEmployeeData,
            footerTemplate: function () {
                return {
                    template: Vue.component('customTemplate', {
                        template: `<span>Count of Margaret:
{{data.Custom}}</span>`,
                        data() { return { data: { data: {} } }; }
                    })
                }
            },
        };
    },
    methods: {
        customAggregateFn: function (data, column) {
            return data.result.filter((dObj) => {
                return getValue('FirstName',
getForeignData(this.$refs.grid.getColumnByField(column.field), dObj)[0]) ===
'Margaret';
            }).length;
        },
    },
    provide: {
        grid: [Aggregate, ForeignKey, Filter],
    },

```

```

    }
  </script>
  <style>
    @import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
  </style>

```

{% previewsample "page.domainurl/code-snippet/grid/how-to/foreignKey-cs3" %}

### Enable multiple foreign key columns

Multiple foreign key columns with editing options are enabled for the Vue Grid component.

In the following example, **Customer Name** and **Ship City** are foreign key columns that display the **ContactName** and **City** columns from foreign data.

### APP.VUE

```

<template>
  <div id="app">
    <ejs-grid :dataSource='orderDetails' :editSettings='editOptions'
    :toolbar='toolbarItems' height='315'>
      <e-columns>
        <e-column field='OrderID' headerText='Order ID'
        textAlign='Right' :validationRules='orderidrules' :isPrimaryKey='true'
        width=100></e-column>
        <e-column field='CustomerID' headerText='Customer Name'
        width=120 foreignKeyValue='ContactName' foreignKeyField="CustomerID"
        :dataSource='customerData' :validationRules='orderidrules'></e-column>
        <e-column field='Freight' headerText='Freight'
        textAlign='Right' editType='numericedit' width=80 format='C2'></e-column>
        <e-column field='EmployeeID' headerText='Ship City'
        width=120 foreignKeyValue='City' foreignKeyField="EmployeeID"
        :dataSource='employeeData' :validationRules='orderidrules'></e-column>
        <e-column field='ShipCountry' headerText='Ship Country'
        editType='dropdownedit' width=130></e-column>
      </e-columns>
    </ejs-grid>
  </div>
</template>
<script>
import Vue from "vue";
import { GridPlugin, Edit, Toolbar, ForeignKey } from "@syncfusion/ej2-vue-
grids";
import { orderDetails, customerData, employeeData } from './datasource.js';
Vue.use(GridPlugin);
export default {
  data() {
    return {
      orderDetails: orderDetails,
      customerData: customerData,
      employeeData: employeeData,
      toolbarItems: ['Add', 'Edit', 'Delete', 'Update', 'Cancel'],
      editOptions: { allowEditing: true, allowAdding: true, allowDeleting:
true },
      orderidrules: { required: true },
    };
  },
  provide: {

```



```

        grid: [Edit, Toolbar, ForeignKey]
    }
}
</script>
<style>
    @import "https://cdn.syncfusion.com/ej2/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/grid/column/foreigncolumn-cs2" %}

### Column reorder in Vue Grid component

Reordering can be done by drag and drop of a particular column header from one index to another index within the grid. To enable reordering, set the [allowReordering](#) to true.

To use Reordering, you need to inject **Reorder** module in the **provide** section.

### APP.VUE

```

<template>
    <div id="app">
        <ejs-grid :dataSource="data" :allowReordering='true' height='315px'>
            <e-columns>
                <e-column field='OrderID' headerText='Order ID'
                textAlign='Right' width=90></e-column>
                <e-column field='CustomerID' headerText='Customer ID'
                width=120></e-column>
                <e-column field='Freight' headerText='Freight' textAlign='Right'
                format='C2' width=90></e-column>
                <e-column field='OrderDate' headerText='Order Date'
                textAlign='Right' format='yMd' type='date' width=120></e-column>
            </e-columns>
        </ejs-grid>
    </div>
</template>
<script>
import Vue from "vue";
import { GridPlugin, Reorder } from "@syncfusion/ej2-vue-grids";
import { data } from './datasource.js';
Vue.use(GridPlugin);
export default {
    data() {
        return {
            data: data
        };
    },
    provide: {
        grid: [Reorder]
    }
}
</script>
<style>
    @import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/grid/column/default-cs11" %}

\* You can disable reordering a particular column by setting the [columns.allowReordering](#) to false.

\* In RTL mode, you can click and drag the left edge of the header cell to resize the column.

### Reorder single column

Grid have option to reorder Columns either by Interaction or by using the `reorderColumns` method. In the below sample, `ShipCity` column is reordered to last column position by using the method.

In the below sample, `Ship City` and `Ship Region` column is reordered to last column position.

### APP.VUE

```
<template>
  <div id="app">
    <ejs-button id='reorderSingleCol' @click.native='reorder'>Reorder
    Ship City to Last</ejs-button>
    <ejs-grid ref='grid' :dataSource="data" :allowReordering='true'
    height='315px'>
      <e-columns>
        <e-column field='OrderID' headerText='Order ID'
        textAlign='Right' width=90></e-column>
        <e-column field='CustomerID' headerText='Customer ID'
        width=120></e-column>
        <e-column field='ShipCity' headerText='Ship City' width=100></e-
        column>
        <e-column field='ShipRegion' headerText='Ship Region'
        width=80></e-column>
        <e-column field='ShipName' headerText='Ship Name' width=80></e-
        column>
      </e-columns>
    </ejs-grid>
  </div>
</template>
<script>
import Vue from "vue";
import { GridPlugin, Reorder } from "@syncfusion/ej2-vue-grids";
import { ButtonPlugin } from "@syncfusion/ej2-vue-buttons";
import { data } from './datasource.js';
Vue.use(GridPlugin);
Vue.use(ButtonPlugin);
export default {
  data() {
    return {
      data: data
    };
  },
  provide: {
    grid: [Reorder]
  },
  methods: {
    reorder: function() {
      this.$refs.grid.reorderColumns('ShipCity', 'ShipName');
    }
  }
}
</script>
<style>
```

```
@import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
#reorderSingleCol {
    text-transform: none;
}
</style>
```

{% previewsample "page.domainurl/code-snippet/grid/column/default-cs12" %}

### Reorder multiple columns

User can reorder a single column at a time by Interaction. Sometimes we need to have reorder multiple columns at the same time, It can be achieved through programmatically by using `reorderColumns` method.

In the below sample, `Ship City` and `Ship Region` column is reordered to last column position.

### APP.VUE

```
<template>
  <div id="app">
    <ejs-button id='reorderMultipleCols' @click.native='reorder'>Reorder
    Ship City and Ship Region to Last</ejs-button>
    <ejs-grid ref='grid' :dataSource="data" :allowReordering='true'
    height='315px'>
      <e-columns>
        <e-column field='OrderID' headerText='Order ID'
        textAlign='Right' width=90></e-column>
        <e-column field='CustomerID' headerText='Customer ID'
        width=120></e-column>
        <e-column field='ShipCity' headerText='Ship City' width=100></e-
        column>
        <e-column field='ShipRegion' headerText='Ship Region'
        width=80></e-column>
        <e-column field='ShipName' headerText='Ship Name' width=80></e-
        column>
      </e-columns>
    </ejs-grid>
  </div>
</template>
<script>
import Vue from "vue";
import { GridPlugin, Reorder } from "@syncfusion/ej2-vue-grids";
import { ButtonPlugin } from "@syncfusion/ej2-vue-buttons";
import { data } from './datasource.js';
Vue.use(GridPlugin);
Vue.use(ButtonPlugin);
export default {
  data() {
    return {
      data: data
    };
  },
  provide: {
    grid: [Reorder]
  },
  methods: {
    reorder: function() {
```

```

    this.$refs.grid.reorderColumns(['ShipCity', 'ShipRegion'], 'ShipName');
  }
}
</script>
<style>
  @import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
  #reorderMultipleCols {
    text-transform: none;
  }
</style>

```

{% previewsample "page.domainurl/code-snippet/grid/column/default-cs13" %}

### Reorder events

During the reorder action, the grid component triggers the below three events.

1. The `columnDragStart` event triggers when column header element drag (move) starts.
2. The `columnDrag` event triggers when column header element is dragged (moved) continuously.
3. The `columnDrop` event triggers when a column header element is dropped on the target column.

### APP.VUE

```

<template>
  <div id="app">
    <ejs-grid ref='grid' :dataSource="data" :allowReordering='true'
      :columnDrag='columnDrag' :columnDrop='columnDrop'
      :columnDragStart='columnDragStart' height='315px'>
      <e-columns>
        <e-column field='OrderID' headerText='Order ID'
          textAlign='Right' width=90></e-column>
        <e-column field='CustomerID' headerText='Customer ID'
          width=120></e-column>
        <e-column field='ShipCity' headerText='Ship City' width=100></e-
column>
        <e-column field='ShipRegion' headerText='Ship Region'
          width=80></e-column>
        <e-column field='ShipName' headerText='Ship Name' width=80></e-
column>
      </e-columns>
    </ejs-grid>
  </div>
</template>
<script>
import Vue from "vue";
import { GridPlugin, Reorder } from "@syncfusion/ej2-vue-grids";
import { ButtonPlugin } from "@syncfusion/ej2-vue-buttons";
import { data } from './datasource.js';
Vue.use(GridPlugin);
Vue.use(ButtonPlugin);
export default {
  data() {

```

```

    return {
      data: data
    };
  },
  provide: {
    grid: [Reorder]
  },
  methods: {
    columnDragStart: function() {
      alert('columnDragStart event is Triggered');
    },
    columnDrag: function() {
      alert('columnDrag event is Triggered');
    },
    columnDrop: function() {
      alert('columnDrop event is Triggered');
    }
  }
}
</script>
<style>
  @import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
  #reorderMultipleCols {
    text-transform: none;
  }
</style>

```

{% previewsample "page.domainurl/code-snippet/grid/column/default-cs14" %}

### Column resizing in Vue Grid component

Grid component provides an intuitive user interface for resizing columns to fit their content. This feature allows users to easily adjust the width of the columns to improve readability and aesthetics of the data presented. To enable column resizing, set the [allowResizing](#) property of the grid to **true**.

Once column resizing is enabled, columns width can be resized by clicking and dragging at the right edge of the column header. While dragging the column, the width of the respective column will be resized immediately.

To use the column resize, inject **ResizeService** in the provider section of **AppModule**.

### APP.VUE

```

<template>
  <div id="app">
    <ejs-grid :dataSource="data" :allowResizing='true' height='315px'>
      <e-columns>
        <e-column field='OrderID' headerText='Order ID'
textAlign='Right' width=100></e-column>
        <e-column field='CustomerID' headerText='Customer ID'
width=120></e-column>
        <e-column field='ShipCity' headerText='Ship City' width=100></e-
column>
        <e-column field='ShipName' headerText='Ship Name' width=80></e-
column>
        <e-column field='ShipCountry' headerText='Ship Country'
textAlign='Right' width=100></e-column>
      </e-columns>
    </ejs-grid>
  </div>
</template>

```

```

        <e-column field='ShipAddress' headerText='Ship Address'
width=120></e-column>
        <e-column field='Freight' headerText='Freight' width=80></e-
column>
    </e-columns>
</ejs-grid>
</div>
</template>
<script>
import Vue from "vue";
import { GridPlugin, Resize } from "@syncfusion/ej2-vue-grids";
import { data } from '../datasource.js';
Vue.use(GridPlugin);
export default {
  data() {
    return {
      data: data
    };
  },
  provide: {
    grid: [Resize]
  }
}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/grid/column/resize-column" %}

- \* You can disable Resizing for a particular column, by specifying [columns.allowResizing](#) to **false**.
- \* In RTL mode, you can click and drag the left edge of header cell to resize the column.
- \* The **width** property of the column can be set initially to define the default width of the column. However, when column resizing is enabled, you can override the default width by manually resizing the columns.

#### *Restrict the resizing based on minimum and maximum width*

The Grid component allows you to restrict the column width resizing between a minimum and maximum width. This can be useful when you want to ensure that your grid's columns stay within a certain range of sizes.

To enable this feature, you can define the [columns.minWidth](#) and [columns.maxWidth](#) properties of the columns directive for the respective column.

In the below code, **OrderID**, **Ship Name** and **Ship Country** columns are defined with minimum and maximum width. The **OrderID** column is set to have a minimum width of 100 pixels and a maximum width of 200 pixels. Similarly, the **ShipName** column is set to have a minimum width of 150 pixels and a maximum width of 300 pixels. The **ShipCountry** column is set to have a minimum width of 120 pixels and a maximum width of 280 pixels.

#### **APP.VUE**

```

<template>
  <div id="app">

```

```

        <ejs-grid :dataSource="data" :allowResizing='true' height='315px'>
            <e-columns>
                <e-column field='OrderID' headerText='Order ID'
textAlign='Right' minWidth= 100 width=150 maxWidth=250 ></e-column>
                <e-column field='CustomerID' headerText='Customer ID'
width=120></e-column>
                <e-column field='ShipCity' headerText='Ship City' width=100></e-
column>
                <e-column field='ShipName' headerText='Ship Name' minWidth= 150
width=200 maxWidth=300></e-column>
                <e-column field='ShipCountry' headerText='Ship Country'
textAlign='Right' minWidth= 120 width=150 maxWidth=280></e-column>
                <e-column field='ShipAddress' headerText='Ship Address'
width=120></e-column>
                <e-column field='Freight' headerText='Freight' width=80></e-
column>
            </e-columns>
        </ejs-grid>
    </div>
</template>
<script>
import Vue from "vue";
import { GridPlugin, Resize } from "@syncfusion/ej2-vue-grids";
import { data } from './datasource.js';
Vue.use(GridPlugin);
export default {
    data() {
        return {
            data: data
        };
    },
    provide: {
        grid: [Resize]
    }
}
</script>
<style>
    @import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/grid/column/resize-min-max" %}

\* The [columns.minWidth](#) and [columns.maxWidth](#) properties will be considered only when the user resizes the column. When resizing the window, these properties will not be considered. This is because columns cannot be re-rendered when resizing the window.

\* When setting the `minWidth` and `maxWidth` properties, ensure that the values are appropriate for your data and layout requirements.

\* The specified `minWidth` and `maxWidth` values take precedence over any user-initiated resizing attempts that fall outside the defined range.

#### *Prevent resizing for particular column*

The Grid component provides the ability to prevent resizing for a particular column. This can be useful if you want to maintain a consistent column width or prevent users from changing the width of a column.

You can disable resizing for a particular column by setting the [allowResizing](#) property of the column to **false**. The following example demonstrates, how to disabled resize for **Customer ID** column.

#### APP.VUE

```
<template>
  <div id="app">
    <ejs-grid :dataSource="data" :allowResizing='true' height='315px'>
      <e-columns>
        <e-column field='OrderID' headerText='Order ID'
textAlign='Right' width=100></e-column>
        <e-column field='CustomerID' headerText='Customer ID'
:allowResizing="false" width=120></e-column>
        <e-column field='ShipCity' headerText='Ship City' width=100></e-
column>
        <e-column field='Freight' headerText='Freight' width=80></e-
column>
      </e-columns>
    </ejs-grid>
  </div>
</template>
<script>
import Vue from "vue";
import { GridPlugin, Resize } from "@syncfusion/ej2-vue-grids";
import { data } from './datasource.js';
Vue.use(GridPlugin);
export default {
  data() {
    return {
      data: data
    };
  },
  provide: {
    grid: [Resize]
  }
}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
</style>
```

{% previewsample "page.domainurl/code-snippet/grid/column/prevent-resize" %}

You can also prevent resizing by setting `args.cancel` to **true** in the [resizeStart](#) event.

#### Resizing modes

The Syncfusion Grid component provides a [ResizeSettingsModel](#) interface for configuring the resizing behavior of grid columns. The interface includes a property named [mode](#) which is of the type [ResizeMode](#). The [ResizeMode](#) is an enum that determines the available resizing modes for the grid columns. There are two resizing modes available for grid columns in Grid:

1. **Normal Mode**: This mode does not adjust the columns to fit the remaining space. When the sum of column width is less than the grid's width, empty space will be present to the right of the last column. When the sum of column width is greater than the grid's width, columns will overflow, and a horizontal scrollbar will appear.



2. **Auto Mode:** This mode automatically resizes the columns to fill the remaining space. When the sum of column width is less than the grid's width, the columns will be automatically expanded to fill the empty space. Conversely, when the sum of column width is greater than the grid's width, the columns will be automatically contracted to fit within the available space.

The following example demonstrates how to set the [resizeSettings.mode](#) property to **Normal** and **Auto** on changing the dropdown value using the [change](#) event of the DropDownList component.

#### APP.VUE

```
<template>
  <div id="app">
    <div style="display: flex">
      <label style="padding: 10px 10px 15px 0"> Change the resize mode:
    </label>
    <ejs-dropdownlist ref='dropdown' id='dropdownlist' style="margin-top:5px" index="0"
      width="100" :dataSource="ddlData" :change="change"
    ></ejs-dropdownlist>
    </div>
    <ejs-grid ref='grid' style="padding: 5px 5px" :dataSource='data'
      height='315' :allowResizing='true'>
      <e-columns>
        <e-column field='OrderID' headerText='Order ID'
          textAlign='Right' width=100></e-column>
        <e-column field='CustomerID' headerText='Customer ID'
          width=120></e-column>
        <e-column field='ShipCity' headerText='Ship City' width=100></e-
          column>
        <e-column field='Freight' headerText='Freight' width=80></e-
          column>
      </e-columns>
    </ejs-grid>
  </div>
</template>
<script>
import Vue from "vue";
import { GridPlugin, Resize } from "@syncfusion/ej2-vue-grids";
import { DropDownListPlugin } from "@syncfusion/ej2-vue-dropdowns";
import { data } from './datasource.js';
Vue.use(GridPlugin);
Vue.use(DropDownListPlugin);
export default {
  data() {
    return {
      data: data,
      ddlData : [
        { text: 'Normal', value: 'Normal' },
        { text: 'Auto', value: 'Auto' },
      ],
    };
  },
  methods: {
    change: function(args) {
      this.$refs.grid.resizeSettings = {mode: args.value}
    }
  }
}
```

```

    },
    provide: {
      grid: [Resize]
    }
  }
</script>
<style>
  @import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/grid/column/resize-mode" %}

### *Resize stacked header column*

Grid component allows to resize stacked columns by clicking and dragging the right edge of the stacked column header. During the resizing action, the width of the child columns is resized at the same time. You can disable resize for any particular stacked column by setting [allowResizing](#) as **false** to its columns.

In this below code, we have disabled resize for **Ship City** column.

### **APP.VUE**

```

<template>
  <div id="app">
    <ejs-grid :dataSource="data" :allowResizing='true'>
      <e-columns>
        <e-column field='OrderID' headerText='Order ID' width='100'
textAlign="Center" minWidth='10'></e-column>
        <e-column headerText='Order Details'
:columns='orderColumns'></e-column>
        <e-column headerText='Ship Details'
:columns='shipColumns'></e-column>
      </e-columns>
    </ejs-grid>
  </div>
</template>
<script>
import Vue from "vue";
import { GridPlugin, Resize } from "@syncfusion/ej2-vue-grids";
import { data } from './datasource.js';
Vue.use(GridPlugin);
export default {
  data() {
    return {
      data: data,
      orderColumns: [
        {
          field: 'OrderDate',
          headerText: 'Order Date',
          type: 'date',
          format: 'yMd',
          width: 120,
          textAlign: 'Right',
          minWidth: 10
        },
        {
          field: 'Freight',

```

```

        headerText: 'Freight ($)',
        width: 100,
        format: 'C1',
        textAlign: 'Right',
        minWidth: 10
    },
    ],
    shipColumns: [
        {
            field: 'ShipCity',
            headerText: 'Ship City',
            width: 100,
            minWidth: 10,
            allowResizing: false
        },
        {
            field: 'ShipCountry',
            headerText: 'Ship Country',
            width: 120,
            minWidth: 10
        }
    ]
    };
},
provide: {
    grid: [Resize]
}
}
</script>
<style>
    @import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/grid/column/resize-stack-column" %}

When the [autoFit](#) property is set to **true**, the Grid will automatically adjust its column width based on the content inside them. In **normal** resize mode, if the [autoFit](#) property is set to **true**, the Grid will maintain any empty space that is left over after resizing the columns. However, in **auto** resize mode, the Grid will ignore any empty space.

#### Touch interaction

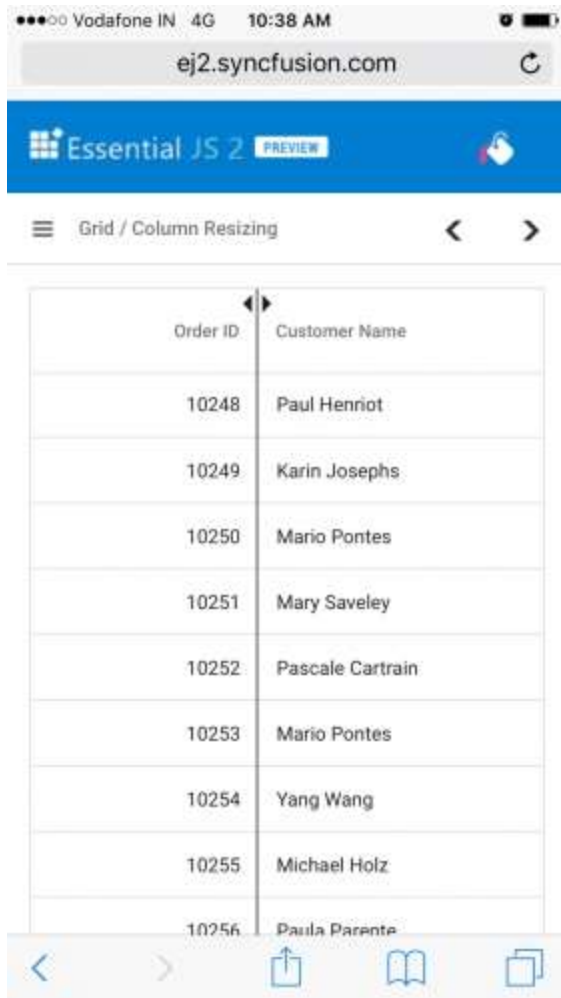
Grid component provides support for touch interactions to enable users to interact with the grid using their mobile devices. Users can resize columns in the grid by tapping and dragging the floating handler, and can also use the Column menu to autofit columns.

#### Resizing Columns on Touch Devices

To resize columns on a touch device:

1. Tap on the right edge of the header cell of the column that you want to resize. 2. A floating handler will appear over the right border of the column. 3. Tap and drag the floating handler to resize the column to the desired width.

The following screenshot represents the column resizing on the touch device.



Order ID	Customer Name
10248	Paul Henriot
10249	Karin Josephs
10250	Mario Pontes
10251	Mary Saveley
10252	Pascale Cartrain
10253	Mario Pontes
10254	Yang Wang
10255	Michael Holz
10256	Paula Parente

### Resizing column externally

Grid provides the ability to resize columns using an external button click. This can be achieved by changing the [width](#) property of the column and refreshing the grid using the [refreshColumns](#) method in the external button click function.

The following example demonstrates how to resize the columns in a grid. This is done by using the [change](#) event of the DropDownList component by change the [width](#) property of the selected column. This is accomplished using the [getColumnByField](#) on external button click. Then, the [refreshColumns](#) method is called on the grid component to update the displayed columns based on interaction.

### APP.VUE

```
<template>
  <div id="app">
    <div style="display: flex">
      <label style='padding: 10px 10px 15px 0'> Change the field:
    </label>
    <ejs-dropdownlist ref='dropdown' id='dropdownlist' style="margin-top:3px" index="0"
      width="100" :dataSource="ddlData"></ejs-dropdownlist>
    </div>
    <div>
      <label style="padding: 30px 17px 0 0">Enter the width: </label>
    </div>
  </div>
</template>
```

```

    <ejs-textbox ref='textbox' type="textbox" placeholder="Enter new
width" width="120"></ejs-textbox>
    <ejs-button ref='button' cssClass='e-outline' v-
on:click.native="onExternalResize">Resize</ejs-button>
  </div>
  <ejs-grid ref='grid' style="padding: 5px 5px" :dataSource='data'
height='315' :allowResizing='true'>
    <e-columns>
      <e-column field='OrderID' headerText='Order ID' textAlign='Right'
width=100></e-column>
      <e-column field='CustomerID' headerText='Customer ID'
width=120></e-column>
      <e-column field='Freight' headerText='Freight' width=80></e-
column>
      <e-column field='ShipCountry' headerText='Ship Country'
width=100></e-column>
    </e-columns>
  </ejs-grid>
</div>
</template>
<script>
import Vue from "vue";
import { GridPlugin, Resize } from "@syncfusion/ej2-vue-grids";
import { DropDownListPlugin } from "@syncfusion/ej2-vue-dropdowns";
import { ButtonPlugin } from '@syncfusion/ej2-vue-buttons';
import { TextBoxPlugin } from '@syncfusion/ej2-vue-inputs';
import { data } from './datasource.js';
Vue.use(GridPlugin);
Vue.use(DropDownListPlugin);
Vue.use(ButtonPlugin);
Vue.use(TextBoxPlugin);
export default {
  data() {
    return {
      data: data,
      ddlData : [
        { text: 'OrderID', value: 'OrderID' },
        { text: 'CustomerID', value: 'CustomerID' },
        { text: 'Freight', value: 'Freight' },
        { text: 'ShipCountry', value: 'ShipCountry' },
      ],
    };
  },
  methods: {
    onExternalResize: function() {
      this.$refs.grid.getColumnByField(this.$refs.dropdown.$el.value).width
= this.$refs.textbox.$el.value;
      this.$refs.grid.refreshColumns();
    }
  },
  provide: {
    grid: [Resize]
  }
}
</script>
<style>
  @import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";

```

```
</style>
```

```
{% previewsample "page.domainurl/code-snippet/grid/column/resize-externally" %}
```

The [refreshColumns](#) method is used to refresh the grid after the column widths are updated. Column resizing externally is useful when you want to provide a custom interface to the user for resizing columns.

### Resizing events

During the resizing action, the grid component triggers the below three events.

1.The [resizeStart](#) event triggers when column resize starts. This event can be used to perform actions when the user begins to resize a column. 2.The [resizing](#) event triggers when column header element is dragged (moved) continuously. This event is useful when you want to perform certain actions during the column resize process. 3.The [resizeStop](#) event triggers when column resize ends. This event can be used to perform actions after the column is resized.

The following is an example of using the resizing events, the [resizeStart](#) event is used to cancel the resizing of the **OrderID** column. The [resizeStop](#) event is used to apply custom CSS attributes to the resized column.

### APP.VUE

```
{% raw %}
<template>
<div id="app">
<div style="margin-left:180px"><p style="color:red;" id="message">{{ message
}}</p></div>
<ejs-grid ref='grid' :dataSource="data" :allowResizing='true' height='315px'
:resizeStart='resizeStart' :resizing='resizing' :resizeStop='resizeStop'>
<e-columns>
<e-column field='OrderID' headerText='Order ID' textAlign='Right'
width=100></e-column>
<e-column field='CustomerID' headerText='Customer ID' width=120></e-column>
<e-column field='ShipCity' headerText='Ship City' width=100></e-column>
<e-column field='ShipName' headerText='Ship Name' width=80></e-column>
<e-column field='ShipCountry' headerText='Ship Country' textAlign='Right'
width=100></e-column>
<e-column field='ShipAddress' headerText='Ship Address' width=120></e-
column>
<e-column field='Freight' headerText='Freight' width=80></e-column>
</e-columns>
</ejs-grid>
</div>
</template>
<script>
import Vue from "vue";
import { GridPlugin, Resize } from "@syncfusion/ej2-vue-grids";
import { data } from './datasource.js';
Vue.use(GridPlugin);
export default {
data() {
return {
data: data
};
},

```

```

provide: {
  grid: [Resize]
},
methods: {
  resizeStart: function (args) {
    this.message = `resizeStart event triggered`;
    if (args.column.field === 'OrderID') {
      args.cancel = true;
    }
  },
  resizing: function (args) {
    this.message = `resizing event triggered`;
  },
  resizeStop: function (args) {
    this.message = `resizeStop event triggered`;
    const headerCell =
    this.$refs.grid.getColumnHeaderByField(args.column.field);
    headerCell.classList.add('customcss');
    const columnCells = this.$refs.grid.getContentTable()
    .querySelectorAll(`[data-colindex="${args.column.index}"]`);
    columnCells.forEach(cell => {
      cell.style.backgroundColor = 'rgb(43, 195, 226)';
    });
  }
}
</script>
<style>
.e-grid .customcss {
  background-color: rgb(43, 195, 226);
}
@import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
</style>
{% endraw %}

```

{% previewsample "page.domainurl/code-snippet/grid/column/resize-event" %}

The ResizeArgs object passed to the events contains information such as the current column width, new column width, column index, and the original event. The [resizing](#) event is triggered multiple times during a single resize operation, so be careful when performing heavy operations in this event.

### Column chooser in Vue Grid component

The column chooser has options to show or hide columns dynamically. It can be enabled by defining the [showColumnChooser](#) as true.

To use column chooser, inject **ColumnChooser** in the **provide** section.

### APP.VUE

```

<template>
  <div id="app">
    <ejs-grid :dataSource="data" :showColumnChooser='true'
    :toolbar='toolbarOptions' height='272px'>
      <e-columns>
        <e-column field='OrderID' headerText='Order ID' width='120'
        textAlign="Right"></e-column>

```

```

        <e-column field='CustomerID' headerText='Customer Name'
width='150' :showInColumnChooser='false'></e-column>
        <e-column field='Freight' headerText='Freight' width='120'
format='C2' textAlign="Right"></e-column>
        <e-column field='ShipCountry' headerText='Ship Country'
:visible='true' width='150'></e-column>
        <e-column field='ShipCity' headerText='Ship City'
:visible='false' width='150'></e-column>
    </e-columns>
</ejs-grid>
</div>
</template>
<script>
import Vue from "vue";
import { GridPlugin, ColumnChooser, Toolbar } from "@syncfusion/ej2-vue-
grids";
import { data } from './datasource.js';
Vue.use(GridPlugin);
export default {
  data() {
    return {
      data: data,
      toolbarOptions: ['ColumnChooser']
    };
  },
  provide: {
    grid: [ColumnChooser, Toolbar]
  }
}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/grid/column/default-cs5" %}

You can hide the column names in column chooser by defining the

[columns.showInColumnChooser](#) as false.

*Open column chooser by external button*

The Column chooser can be displayed on a page through external button by invoking

the [openColumnChooser](#) method with X and Y axis positions.

#### APP.VUE

```

<template>
  <div id="app">
    <ejs-button @click.native="show">open Column Chooser </ejs-button>
    <ejs-grid ref='grid' :dataSource="data" :showColumnChooser='true'
height='272px'>
      <e-columns>
        <e-column field='OrderID' headerText='Order ID' width='120'
textAlign="Right"></e-column>
        <e-column field='CustomerID' headerText='Customer Name'
width='150' :showInColumnChooser='false'></e-column>

```



```

        <e-column field='OrderDate' headerText='Order Date'
width='130' format='yMd' textAlign='Right' type='date'></e-column>
        <e-column field='Freight' headerText='Freight' width='120'
format='C2' textAlign='Right'></e-column>
        <e-column field='ShipCountry' headerText='Ship Country'
:visible='false' width='150'></e-column>
        <e-column field='ShipCity' headerText='Ship City'
:visible='false' width='150'></e-column>
    </e-columns>
</ejs-grid>
</div>
</template>
<script>
import Vue from "vue";
import { GridPlugin, ColumnChooser } from "@syncfusion/ej2-vue-grids";
import { ButtonPlugin } from "@syncfusion/ej2-vue-buttons";
import { data } from './datasource.js';
Vue.use(GridPlugin);
Vue.use(ButtonPlugin);
export default {
  data() {
    return {
      data: data
    };
  },
  methods: {
    show: function() {
      this.$refs.grid.ej2Instances.columnChooserModule.openColumnChooser(200, 50);
      // give X and Y axis
    }
  },
  provide: {
    grid: [ColumnChooser]
  }
};
</script>
<style>
  @import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/grid/column/default-cs6" %}

### Column menu in Vue Grid component

The column menu has options to integrate features like sorting, grouping, filtering, column chooser, and autofit. It will show a menu with the integrated feature when users click on multiple icon of the column. To enable column menu, you need to define the [showColumnMenu](#) property as true.

To use the column menu, inject the `ColumnMenu` in the `provide` section.

The default items are displayed in following table.

Item	Description
----- -----	
SortAscending	Sort the current column in ascending order.

- | **SortDescending** | Sort the current column in descending order. |
- | **Group** | Group the current column. |
- | **Ungroup** | Ungroup the current column. |
- | **AutoFit** | Auto fit the current column. |
- | **AutoFitAll** | Auto fit all columns. |
- | **ColumnChooser** | Choose the column visibility. |
- | **Filter** | Show the filter option as given in **filterSettings.type** |

### APP.VUE

```

<template>
  <div id="app">
    <ejs-grid :dataSource="data" id="gridcomp" :allowPaging='true'
:allowGrouping='true' :allowSorting='true' :showColumnMenu='true'
:groupSettings='groupOptions' :allowFiltering='true'
:filterSettings='filterSettings'>
      <e-columns>
        <e-column field='OrderID' headerText='Order ID' width='120'
textAlign='Right'></e-column>
        <e-column field='Freight' headerText='Freight' format='C2'
textAlign='Right' width='120'></e-column>
        <e-column field='ShippedDate' headerText='Shipped Date'
width='130' format='yMd' textAlign='Right' type='date'></e-column>
        <e-column field='ShipCountry' headerText='Ship Country'
:visible='false' width='150'></e-column>
        <e-column field='ShipCity' headerText='Ship City'
width='150'></e-column>
      </e-columns>
    </ejs-grid>
  </div>
</template>
<script>
import Vue from "vue";
import { GridPlugin, Group, Sort, Resize, ColumnMenu, Page } from
"@syncfusion/ej2-vue-grids";
import { data } from '../datasource.js';
Vue.use(GridPlugin);
export default {
  data() {
    return {
      data: data,
      groupOptions: { showGroupedColumn: true },
      filterSettings: { type: "CheckBox" }
    };
  },
  provide: {
    grid: [Group, Sort, Resize, ColumnMenu, Page]
  }
}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";

```

```
</style>
```

```
{% previewsample "page.domainurl/code-snippet/grid/column/default-cs7" %}
```

You can disable column menu for a particular column by defining the [columns.showColumnMenu](#) as false.

You can customize the default items by defining the [columnMenuItems](#) with required items.

#### Column menu events

During the resizing action, the grid component triggers the below two events.

1. The `columnMenuOpen` event triggers before the column menu opens.
2. The `columnMenuClick` event triggers when the user clicks the column menu of the grid.

#### APP.VUE

```
<template>
  <div id="app">
    <ejs-grid :dataSource="data" id="gridcomp" :allowPaging='true'
:allowGrouping='true' :allowSorting='true' :showColumnMenu='true'
:groupSettings='groupOptions' :allowFiltering='true'
:filterSettings='filterSettings'
:columnMenuClick='columnMenuClick' :columnMenuOpen='columnMenuOpen'>
      <e-columns>
        <e-column field='OrderID' headerText='Order ID' width='120'
textAlign='Right'></e-column>
        <e-column field='Freight' headerText='Freight' format='C2'
textAlign='Right' width='120'></e-column>
        <e-column field='ShippedDate' headerText='Shipped Date'
width='130' format="yMd" textAlign='Right' type='date'></e-column>
        <e-column field='ShipCountry' headerText='Ship Country'
:visible='false' width='150'></e-column>
        <e-column field='ShipCity' headerText='Ship City'
width='150'></e-column>
      </e-columns>
    </ejs-grid>
  </div>
</template>
<script>
import Vue from "vue";
import { GridPlugin, Group, Sort, Resize, ColumnMenu, Page } from
"@syncfusion/ej2-vue-grids";
import { data } from './datasource.js';
Vue.use(GridPlugin);
export default {
  data() {
    return {
      data: data,
      groupOptions: { showGroupedColumn: true },
      filterSettings: { type: "CheckBox" }
    };
  },
  provide: {
    grid: [Group, Sort, Resize, ColumnMenu, Page]
```

```

    },
    methods: {
      columnMenuOpen: function(){
        alert('columnMenuOpen event is Triggered');
      },
      columnMenuClick: function(){
        alert('columnMenuClick event is Triggered');
      }
    }
  }
</script>
<style>
  @import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/grid/column/default-cs8" %}

#### Custom column menu item

Custom column menu items can be added by defining the [columnMenuItems](#) as collection of the [columnMenuItemModel](#). Actions for this customized items can be defined in the [columnMenuClick](#) event.

#### APP.VUE

```

<template>
  <div id="app">
    <ejs-grid ref='grid' :dataSource="data" id="gridcomp"
:allowPaging='true' :allowSorting='true' :showColumnMenu='true'
:sortSettings='sortSettings' :columnMenuItems='columnMenuItems'
:columnMenuClick='columnMenuClick'>
      <e-columns>
        <e-column field='OrderID' headerText='Order ID' width='140'
textAlign="Right"></e-column>
        <e-column field='CustomerID' headerText='Customer Name'
:showInColumnChooser='false'></e-column>
        <e-column field='Freight' headerText='Freight' format='C2'
textAlign="Right"></e-column>
        <e-column field='ShipCountry' headerText='Ship Country'
:visible='false' width='150'></e-column>
        <e-column field='ShipCity' headerText='Ship City'
width='150'></e-column>
      </e-columns>
    </ejs-grid>
  </div>
</template>
<script>
import Vue from "vue";
import { GridPlugin, Sort, ColumnMenu, Page } from "@syncfusion/ej2-vue-
grids";
import { data } from './datasource.js';
Vue.use(GridPlugin);
export default {
  data() {
    return {
      data: data,
      columnMenuItems: [{text:'Clear Sorting', id:'gridclearsorting'}],

```

```

        sortSettings: {columns:[{direction: "Ascending", field: "OrderID"}]}
    };
},
methods: {
    columnMenuClick: function(args) {
        if(args.item.id === 'gridclearsorting'){
            this.$refs.grid.clearSorting();
        }
    }
},
provide: {
    grid: [Sort, ColumnMenu, Page]
}
}
</script>
<style>
    @import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/grid/column/default-cs9" %}

#### *Customize menu items for particular columns*

Sometimes, you have a scenario that to hide an item from column menu for particular columns. In that case, you need to define the [columnMenuOpenEventArgs.hide](#) as true in the [columnMenuOpen](#) event.

The following sample, **Filter** item was hidden in column menu when opens for the **OrderID** column.

#### **APP.VUE**

```

<template>
    <div id="app">
        <ejs-grid ref='grid' :dataSource="data" id="gridcomp"
        :allowPaging='true' :allowSorting='true' :showColumnMenu='true'
        :filterSettings='filterSettings' :columnMenuOpen='columnMenuOpen'
        :allowFiltering='true' :allowGrouping='true'>
            <e-columns>
                <e-column field='OrderID' headerText='Order ID' width='140'
                textAlign="Right"></e-column>
                <e-column field='CustomerID' headerText='Customer Name'
                :showInColumnChooser='false'></e-column>
                <e-column field='Freight' headerText='Freight' format='C2'
                textAlign="Right"></e-column>
                <e-column field='ShipCountry' headerText='Ship Country'
                :visible='false' width='150'></e-column>
                <e-column field='ShipCity' headerText='Ship City'
                width='150'></e-column>
            </e-columns>
        </ejs-grid>
    </div>
</template>
<script>
import Vue from "vue";
import { GridPlugin, Sort, ColumnMenu, Page, Group, Filter } from
"@syncfusion/ej2-vue-grids";
import { data } from './datasource.js';
Vue.use(GridPlugin);

```

```

export default {
  data() {
    return {
      data: data,
      filterSettings: {type: 'Menu'}
    };
  },
  methods: {
    columnMenuOpen: function (args) {
      for (let item of args.items) {
        if (item.text === 'Filter' && args.column.field === 'OrderID') {
          item.hide = true;
        } else {
          item.hide = false;
        }
      }
    }
  },
  provide: {
    grid: [Sort, ColumnMenu, Page, Group, Filter]
  }
}
</script>
<style>
  @import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/grid/column/default-cs10" %}

#### *Customize the icon of column menu*

You can customize the column menu icon by overriding the default grid class `.e-icons.e-columnmenu` with a custom property `content` as mentioned below,

```

.e-grid .e-columnheader .e-icons.e-columnmenu::before {
  content: "\e941";
}

```

In the below sample, grid is rendered with a customized column menu icon.

#### **APP.VUE**

```

<template>
  <div id="app">
    <ejs-grid ref='grid' :dataSource='data':showColumnMenu='true'
height='315px' >
      <e-columns>
        <e-column field='OrderID' headerText='Order ID'
width=90></e-column>
        <e-column field='CustomerID' headerText='Customer ID'
width=120></e-column>
        <e-column field='Freight' headerText='Freight' format='C2'
width=90></e-column>

```

```

        <e-column field='ShipName' headerText='Ship Name'
width=120></e-column>
    </e-columns>
</ejs-grid>
</div>
</template>
<script>
import Vue from "vue";
import { GridPlugin, ColumnMenu } from "@syncfusion/ej2-vue-grids";
import { data } from './datasource.js';
Vue.use(GridPlugin);
export default {
  data() {
    return {
      data: data
    };
  },
  provide: {
    grid: [ColumnMenu]
  }
}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
.e-grid .e-columnheader .e-icons.e-columnmenu::before {
  content: "\e941";
}
</style>

```

{% previewsample "page.domainurl/code-snippet/grid/how-to/default-cs1" %}

### Column spanning in Vue Grid component

The grid has option to span the adjacent cells. You need to define the [colSpan](#) attribute to span cells in the [QueryCellInfo](#) event.

In the following demo, Employee Davolio doing analysis from 9.00 AM to 10.00 AM, so that cells have spanned.

### APP.VUE

```

<template>
  <div id="app">
    <ejs-grid :dataSource="data" height='auto' width='auto'
gridLines='Both' :allowTextWrap='true' :queryCellInfo='queryCellInfoEvent'>
      <e-columns>
        <e-column field='EmployeeID' headerText='Employee ID'
width='150' textAlign='Right' isPrimaryKey={true}></e-column>
        <e-column field='EmployeeName' headerText='Employee Name'
width='200'></e-column>
        <e-column field='9:00' headerText='9:00 AM' width='120'></e-
column>
        <e-column field='9:30' headerText='9:30 AM' width='120'></e-
column>
        <e-column field='10:00' headerText='10:00 AM'
width='120'></e-column>

```

```

        <e-column field='10:30' headerText='10:30 AM'
width='120'></e-column>
        <e-column field='11:00' headerText='11:00 AM'
width='120'></e-column>
        <e-column field='11:30' headerText='11:30 AM'
width='120'></e-column>
        <e-column field='12:00' headerText='12:00 PM'
width='120'></e-column>
        <e-column field='12:30' headerText='12:30 PM'
width='120'></e-column>
        <e-column field='2:30' headerText='2:30 PM' width='120'></e-
column>
        <e-column field='3:00' headerText='3:00 PM' width='120'></e-
column>
        <e-column field='3:30' headerText='3:30 PM' width='120'></e-
column>
        <e-column field='4:00' headerText='4:00 PM' width='120'></e-
column>
        <e-column field='4:30' headerText='4:30 PM' width='120'></e-
column>
        <e-column field='5:00' headerText='5:00 PM' width='120'></e-
column>
    </e-columns>
</ejs-grid>
</div>
</template>
<script>
import Vue from 'vue';
import { GridPlugin } from '@syncfusion/ej2-vue-grids';
import { data } from './datasource.js';
Vue.use(GridPlugin);
export default {
    data: () => {
        return {
            data: data
        }
    },
    methods: {
        queryCellInfoEvent: function(args) {
            let data = args.data;
            switch (data.EmployeeID) {
                case 10001:
                    if (args.column.field === '9:00' || args.column.field ===
'2:30' || args.column.field === '4:30') {
                        args.colSpan = 2;
                    } else if (args.column.field === '11:00') {
                        args.colSpan = 3;
                    }
                    break;
                case 10002:
                    if (args.column.field === '9:30' || args.column.field ===
'2:30' ||
                        args.column.field === '4:30') {
                        args.colSpan = 3;
                    } else if (args.column.field === '11:00') {
                        args.colSpan = 4;
                    }
            }
        }
    }
}

```



```

        break;
    case 10003:
        if (args.column.field === '9:00' || args.column.field ===
'11:30') {
            args.colSpan = 3;
        } else if (args.column.field === '10:30' ||
args.column.field === '3:30' ||
            args.column.field === '4:30' || args.column.field ===
'2:30') {
            args.colSpan = 2;
        }
        break;
    case 10004:
        if (args.column.field === '9:00') {
            args.colSpan = 3;
        } else if (args.column.field === '11:00') {
            args.colSpan = 4;
        } else if (args.column.field === '4:00' || args.column.field
=== '2:30') {
            args.colSpan = 2;
        }
        break;
    case 10005:
        if (args.column.field === '9:00') {
            args.colSpan = 4;
        } else if (args.column.field === '11:30') {
            args.colSpan = 3;
        } else if (args.column.field === '3:30' || args.column.field
=== '4:30' || args.column.field === '2:30') {
            args.colSpan = 2;
        }
        break;
    case 10006:
        if (args.column.field === '9:00' || args.column.field ===
'4:30' ||
            args.column.field === '2:30' || args.column.field ===
'3:30') {
            args.colSpan = 2;
        } else if (args.column.field === '10:00' ||
args.column.field === '11:30') {
            args.colSpan = 3;
        }
        break;
    case 10007:
        if (args.column.field === '9:00' || args.column.field ===
'3:00' || args.column.field === '10:30') {
            args.colSpan = 2;
        } else if (args.column.field === '11:30' ||
args.column.field === '4:00') {
            args.colSpan = 3;
        }
        break;
    case 10008:
        if (args.column.field === '9:00' || args.column.field ===
'10:30' || args.column.field === '2:30') {
            args.colSpan = 3;
        } else if (args.column.field === '4:00') {

```

```

        args.colSpan = 2;
    }
    break;
    default:
        this.extendQueryCellEvent(args, data.EmployeeID);
    }
},
extendQueryCellEvent: function(args, value) {
    switch (value) {
        case 10009:
            if (args.column.field === '9:00' || args.column.field ===
'11:30') {
                args.colSpan = 3;
            } else if (args.column.field === '4:30' || args.column.field
=== '2:30') {
                args.colSpan = 2;
            }
            break;
        case 100010:
            if (args.column.field === '9:00' || args.column.field ===
'2:30' ||
            args.column.field === '4:00' || args.column.field ===
'11:30') {
                args.colSpan = 3;
            } else if (args.column.field === '10:30') {
                args.colSpan = 2;
            }
            break;
        }
    }
});
</script>
<style>
    @import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/grid/column/spanning-cs1" %}

### Limitations

- Column spanning is not compatible with the following features:
  - Virtual scrolling
  - Infinite scrolling
  - Lazy load grouping

### Row

#### Row in Vue Grid component

It represents the record details that are fetched from the data source.

*Row customization**Using event*

You can customize the appearance of the Row by using the [rowDataBound](#) event. The [rowDataBound](#) event triggers for every row. In that event handler, you can get [RowDataBoundEventArgs](#) which contain details of the row.

**APP.VUE**

```
<template>
  <div id="app">
    <ejs-grid :dataSource="data" :enableHover='false'
:allowSelection='false' height='315' :rowDataBound='rowDataBound'>
      <e-columns>
        <e-column field='OrderID' headerText='Order ID'
textAlign='Right' width=90></e-column>
        <e-column field='CustomerID' headerText='Customer ID'
width=120></e-column>
        <e-column field='Freight' headerText='Freight' textAlign='Right'
format='C2' width=90></e-column>
        <e-column field='OrderDate' headerText='Order Date'
textAlign='Right' format='yMd' type='date' width=120> </e-column>
      </e-columns>
    </ejs-grid>
  </div>
</template>
<script>
import Vue from "vue";
import { GridPlugin } from "@syncfusion/ej2-vue-grids";
import { data } from './datasource.js';
Vue.use(GridPlugin);
export default {
  data() {
    return {
      data: data
    };
  },
  methods: {
    rowDataBound: function(args) {
      if (args.data['Freight'] < 30) {
        args.row.classList.add('below-30');
      } else if (args.data['Freight'] < 80) {
        args.row.classList.add('below-80');
      } else {
        args.row.classList.add('above-80');
      }
    }
  }
}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
.below-30 {
  background-color: orangered;
}
.below-80 {
  background-color: yellow;
}
```

```

    }
    .above-80 {
        background-color: greenyellow
    }
</style>

```

{% previewsample "page.domainurl/code-snippet/grid/row/default-cs5" %}

Using CSS customize alternate rows

You can change the grid's alternative rows' background color by overriding the `.e-altrow` class.

```

.e-grid .e-altrow {
background-color: #fafafa;
}

```

Please refer the following example.

#### APP.VUE

```

<template>
  <div id="app">
    <ejs-grid :dataSource="data">
      <e-columns>
        <e-column field='OrderID' headerText='Order ID'
textAlign='Right' width=90></e-column>
        <e-column field='CustomerID' headerText='Customer ID'
width=120></e-column>
        <e-column field='Freight' headerText='Freight' textAlign='Right'
format='C2' width=90></e-column>
        <e-column field='OrderDate' headerText='Order Date'
textAlign='Right' format='yMd' type='date' width=120> </e-column>
      </e-columns>
    </ejs-grid>
  </div>
</template>
<script>
import Vue from "vue";
import { GridPlugin } from "@syncfusion/ej2-vue-grids";
import { data } from './datasource.js';
Vue.use(GridPlugin);
export default {
  data() {
    return {
      data: data
    };
  }
}
</script>
<style>
  @import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
  .e-grid .e-altrow {
    background-color: #fafafa;
  }
</style>

```

```
}
</style>
```

{% previewsample "page.domainurl/code-snippet/grid/row/default-cs6" %}

[Using CSS customize selected row](#)

The background color of the selected row can be changed by overriding the following CSS style.

```
.e-grid td.e-active {
background-color: #f9920b;
}
,
```

This is demonstrated in the following sample:

### **APP.VUE**

```
<template>
  <div id="app">
    <ejs-grid :dataSource="data">
      <e-columns>
        <e-column field='OrderID' headerText='Order ID'
textAlign='Right' width=90></e-column>
        <e-column field='CustomerID' headerText='Customer ID'
width=120></e-column>
        <e-column field='Freight' headerText='Freight' textAlign='Right'
format='C2' width=90></e-column>
        <e-column field='OrderDate' headerText='Order Date'
textAlign='Right' format='yMd' type='date' width=120> </e-column>
      </e-columns>
    </ejs-grid>
  </div>
</template>
<script>
import Vue from "vue";
import { GridPlugin } from "@syncfusion/ej2-vue-grids";
import { data } from './datasource.js';
Vue.use(GridPlugin);
export default {
  data() {
    return {
      data: data
    };
  }
}
</script>
<style>
  @import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
  .e-grid td.e-active {
    background-color: #f9920b;
  }
</style>
```

```
{% previewsample "page.domainurl/code-snippet/grid/row/default-cs7" %}
```

### [Adding a new row programmatically](#)

The Grid can add a new row between the existing rows using the [addRecord](#) method of the Grid.

This is demonstrated in the following sample:

#### **APP.VUE**

```
<template>
  <div id="app">
    <button id="add" @click="clickRow">Add Row</button>
    <br /><br />
    <ejs-grid ref="grid" :dataSource="data"
:editSettings='editSettings'>
      <e-columns>
        <e-column field='OrderID' headerText='Order ID'
textAlign='Right' width=90 isPrimaryKey="true"></e-column>
        <e-column field='CustomerID' headerText='Customer ID'
width=120></e-column>
        <e-column field='Freight' headerText='Freight' textAlign='Right'
format='C2' width=90></e-column>
        <e-column field='ShipCity' headerText='Ship City' width=120>
</e-column>
      </e-columns>
    </ejs-grid>
  </div>
</template>
<script>
import Vue from "vue";
import { GridPlugin, Page, Edit } from "@syncfusion/ej2-vue-grids";
import { data } from './datasource.js';
Vue.use(GridPlugin);
export default {
  data() {
    return {
      data: data,
      editSettings: { allowEditing: true, allowAdding: true, allowDeleting:
true }
    };
  },
  methods: {
    clickRow: function () {
      this.$refs.grid.ej2Instances.addRecord(
        { OrderID: 3232, CustomerID: 'ALKIT', Freight: 40, ShipCity:
'London'}, 2);
    }
  },
  provide: {
    grid: [Page, Edit]
  }
}
</script>
<style>
  @import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
</style>
```

```
{% previewsample "page.domainurl/code-snippet/grid/row/default-cs8" %}
```

When working with remote data, it is impossible to add a new row between the existing rows.

#### *How to get the row information when hovering over the cell*

It is possible to get the row information when hovering over the specific cell. This can be achieved by using the [rowDataBound](#) event and [getRowInfo](#) method of the Grid.

In the following sample, the `mouseover` event is bound to a grid row in the `rowDataBound` event, and when hovering over the specific cell, using the `getRowInfo` method, row information will be retrieved and displayed in the console.

#### **APP.VUE**

```
<template>
  <div id="app">
    <ejs-grid id="grid" :dataSource="data" :editSettings='editSettings'
:rowDataBound='rowDataBound'>
      <e-columns>
        <e-column field='OrderID' headerText='Order ID'
textAlign='Right' width=90></e-column>
        <e-column field='CustomerID' headerText='Customer ID'
width=120></e-column>
        <e-column field='Freight' headerText='Freight' textAlign='Right'
format='C2' width=90></e-column>
        <e-column field='OrderDate' headerText='Order Date'
textAlign='Right' format='yMd' type='date' width=120> </e-column>
      </e-columns>
    </ejs-grid>
  </div>
</template>
<script>
import Vue from "vue";
import { GridPlugin, Edit } from "@syncfusion/ej2-vue-grids";
import { data } from './datasource.js';
import { MouseEventArgs } from '@syncfusion/ej2-base';
Vue.use(GridPlugin);
export default {
  data() {
    return {
      data: data,
      editSettings: { allowEditing: true, allowAdding: true, allowDeleting:
true }
    };
  },
  methods: {
    rowDataBound(args) {
      let gridElement = document.getElementById('grid').ej2_instances[0];
      args.row.addEventListener('mouseover', function(e) {
        console.log(gridElement.getRowInfo(e.target))
      })
    }
  },
  provide: {
    grid: [Edit]
  }
}
```

```

</script>
<style>
  @import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/grid/row/default-cs9" %}

[See Also](#)

- [How to pass data to DetailTemplate in Vue Grid](#)

### Row height in Vue Grid component

You can customize the row height of grid rows through the [rowHeight](#) property. The `rowHeight` property is used to change the row height of entire grid rows.

In the below example, the `rowHeight` is set as '60px'.

#### APP.VUE

```

<template>
  <div id="app">
    <ejs-grid :dataSource="data" :enableHover='false'
    :allowSelection='false' height='295' rowHeight=60>
      <e-columns>
        <e-column field='OrderID' headerText='Order ID'
        textAlign='Right' width=90></e-column>
        <e-column field='CustomerID' headerText='Customer ID'
        width=120></e-column>
        <e-column field='Freight' headerText='Freight' textAlign='Right'
        format='C2' width=90></e-column>
        <e-column field='OrderDate' headerText='Order Date'
        textAlign='Right' format='yMd' type='date' width=120> </e-column>
      </e-columns>
    </ejs-grid>
  </div>
</template>
<script>
import Vue from "vue";
import { GridPlugin } from "@syncfusion/ej2-vue-grids";
import { data } from './datasource.js';
Vue.use(GridPlugin);
export default {
  data() {
    return {
      data: data
    };
  }
}
</script>
<style>
  @import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/grid/row/default-cs3" %}



### Customize row height for particular row

Grid row height for particular row can be customized using the [rowDataBound](#) event by setting the `rowHeight` in arguments for each row based on the requirement.

In the below example, the row height for the row with OrderID as '10249' is set as '90px' using the `rowDataBound` event.

#### APP.VUE

```
<template>
  <div id="app">
    <ejs-grid :dataSource="data" height='315'
    :rowDataBound='rowDataBound'>
      <e-columns>
        <e-column field='OrderID' headerText='Order ID'
        textAlign='Right' width=90></e-column>
        <e-column field='CustomerID' headerText='Customer ID'
        width=120></e-column>
        <e-column field='Freight' headerText='Freight' textAlign='Right'
        format='C2' width=90></e-column>
        <e-column field='OrderDate' headerText='Order Date'
        textAlign='Right' format='yMd' type='date' width=120 > </e-column>
      </e-columns>
    </ejs-grid>
  </div>
</template>
<script>
import Vue from "vue";
import { GridPlugin } from "@syncfusion/ej2-vue-grids";
import { data } from './datasource.js';
Vue.use(GridPlugin);
export default {
  data() {
    return {
      data: data
    };
  },
  methods: {
    rowDataBound: (args) {
      if (args.data['OrderID'] === 10249) {
        args.rowHeight = 90;
      }
    }
  }
};
</script>
<style>
  @import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
</style>
```

{% previewsample "page.domainurl/code-snippet/grid/row/default-cs4" %}

In virtual scrolling mode, it is not applicable to set the `rowHeight` using the `rowDataBound` event.

### Row template in Vue Grid component

The [rowTemplate](#) has options to display custom `<tr>` element.

The `rowTemplate` property should be a function which returns an object. The object should contain a Vue component which should be assigned to the `template` property. The grid will look for the template property and render it as new Vue instance.

In `rowTemplate`, the Vue component `template` should be a `<tr>` element.

#### APP.VUE

```
<template>
  <div id="app">
    <ejs-grid :dataSource="data" height=310 width='auto'
:rowTemplate='rowTemplate' >
      <e-columns>
        <e-column field='Employee Image' headerText='Employee Image'
width='150' textAlign='Center'></e-column>
        <e-column field='Employee Details' headerText='Employee Details'
width='300' textAlign='Left'></e-column>
      </e-columns>
    </ejs-grid>
  </div>
</template>
<script>
import Vue from "vue";
import { GridPlugin } from "@syncfusion/ej2-vue-grids";
import { employeeData } from "../datasource.js";
import { Internationalization } from '@syncfusion/ej2-base';
let instance = new Internationalization();
Vue.use(GridPlugin);
export default {
  data: () => {
    return {
      data: employeeData,
      rowTemplate: function () {
        return { template : Vue.component('rowTemplate',{
          template: `<tr>
<td class="rowphoto">
  
</td>
<td class="details">
  <table class="CardTable" cellpadding="3" cellspacing="2">
    <colgroup>
      <col width="50%">
      <col width="50%">
    </colgroup>
    <tbody>
      <tr>
        <td class="CardHeader">First Name </td>
        <td>{{data.FirstName}} </td>
      </tr>
      <tr>
        <td class="CardHeader">Last Name</td>
        <td>{{data.LastName}} </td>
      </tr>
      <tr>
        <td class="CardHeader">Title</td>
        <td>{{data.Title}} </td>
      </tr>
    </tbody>
  </table>
</td>
</tr>`
        }
      }
    }
  }
}
```

```

        <tr>
            <td class="CardHeader">Birth Date</td>
            <td>{{format(data.BirthDate)}} </td>
        </tr>
        <tr>
            <td class="CardHeader">Hire Date</td>
            <td>{{format(data.HireDate)}} </td>
        </tr>
    </tbody>
</table>
</td>
</tr>`,
    data: function() {
        return {
            data: {}
        }
    },
    methods: {
        format: function(value) {
            return instance.formatDate(value, { skeleton: 'yMd', type:
'date' });
        }
    },
    computed: {
        image: function() {
            return './images/' + this.data.EmployeeID + '.png';
        },
        altImage: function() {
            return this.data.EmployeeID;
        }
    }
    })}
}
};
}
}
</script>
<style>
    @import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
    .rowphoto img {
        width: 100px;
        height: 100px;
        border-radius: 50px;
        box-shadow: inset 0 0 1px #e0e0e0, inset 0 0 14px rgba(0,0,0,0.2);
    }
    @media screen and (max-width: 600px) and (min-width: 320px) {
        .rowphoto img {
            width: 50px;
            height: 50px;
        }
    }
    @media screen and (max-width: 800px) and (min-width: 600px) {
        .rowphoto img {
            width: 70px;
            height: 70px;
        }
    }
}

```

```

.rowphoto,
.details {
  border-color: #e0e0e0;
  border-style: solid;
}
.rowphoto {
  border-width: 1px 0px 0px 0px;
  text-align: center;
}
.details {
  border-width: 1px 0px 0px 0px;
  padding-left: 18px;
}
.e-bigger .details {
  padding-left: 25px;
}
.e-device .details {
  padding-left: 8px;
}
.details > table {
  width: 100%;
}
.CardHeader {
  font-weight: bolder;
}
td {
  padding: 2px 2px 3px 4px;
}
</style>

```

{% previewsample "page.domainurl/code-snippet/grid/column/template-cs5" %}

#### Detail template in Vue Grid component

The [detailTemplate](#) provides an additional information about a data row which can show or hide by clicking on expand or collapse button. The [detailTemplate](#) property accepts the template for the detail row.

The [detailTemplate](#) property should be a function which returns an object. The object should contain a Vue component which should be assigned to the `template` property. The grid will look for the `template` property and render it as new Vue instance.

To use `detailTemplate`, inject the `DetailRow` in the `provide` section.

#### APP.VUE

```

<template>
  <div id="app">
    <ejs-grid :dataSource="data" :detailTemplate = 'detailTemplate' >
      <e-columns>
        <e-column field='EmployeeID' headerText='Employee ID'
width='125' textAlign='Right'></e-column>
        <e-column field='FirstName' headerText='Name'
width='120'></e-column>
        <e-column field='Title' headerText='Title' width='170'></e-
column>

```

```

        <e-column field='HireDate' headerText='Hire Date'
width='135' textAlign='Right' format='yMd'></e-column>
        <e-column field='ReportsTo' headerText='Reports To'
width='120' textAlign='Right'></e-column>
    </e-columns>
</ejs-grid>
</div>
</template>
<script>
import Vue from "vue";
import { GridPlugin, DetailRow } from "@syncfusion/ej2-vue-grids";
import { employeeData } from "../datasource.js";
import { Internationalization } from '@syncfusion/ej2-base';
let instance = new Internationalization();
Vue.use(GridPlugin);
export default {
    data: () => {
        return {
            data: employeeData,
            detailTemplate: function () {
                return { template : Vue.component('detailTemplate',{
                    template: `
<table class="detailtable" width="100%">
    <colgroup>
        <col width="35%">
        <col width="35%">
        <col width="30%">
    </colgroup>
    <tbody>
        <tr>
            <td rowspan="4" style="text-align: center;">
                
            </td>
            <td>
                <span style="font-weight: 500;">First Name: </span>
                {{data.FirstName}}
            </td>
            <td>
                <span style="font-weight: 500;">Postal Code: </span>
                {{data.PostalCode}}
            </td>
        </tr>
        <tr>
            <td>
                <span style="font-weight: 500;">Last Name: </span>
                {{data.LastName}}
            </td>
            <td>
                <span style="font-weight: 500;">City: </span>
                {{data.City}}
            </td>
        </tr>
        <tr>
            <td>
                <span style="font-weight: 500;">Title: </span>
                {{data.Title}}
            </td>
        </tr>
    </tbody>
</table>
                    `
                })
            }
        }
    }
}

```

```

                <td>
                    <span style="font-weight: 500;">Phone: </span>
                </td>
            </tr>
            <tr>
                <td>
                    <span style="font-weight: 500;">Address: </span>
                </td>
                <td>
                    <span style="font-weight: 500;">HireDate: </span>
                </td>
            </tr>
        </tbody>
    </table>`,
    data: function() {
        return {
            data: {}
        },
        methods: {
            format: function(value) {
                return instance.formatDate(value, { skeleton: 'yMd', type:
'date' });
            },
            computed: {
                image: function() {
                    return './' + this.data.EmployeeID + '.png';
                },
                altImage: function() {
                    return this.data.EmployeeID;
                }
            }
        }
    })
}
};
},
provide: {
    grid:[DetailRow]
}
}
</script>
<style>
    @import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
    .detailtable td {
        font-size: 13px;
        padding: 4px;
        max-width: 0;
        overflow: hidden;
        text-overflow: ellipsis;
        white-space: nowrap;
    }
    .photo {
        width: 100px;

```

```

        height: 100px;
        border-radius: 50px;
        box-shadow: inset 0 0 1px #e0e0e0, inset 0 0 14px rgba(0,0,0,0.2);
    }
    @media screen and (max-width: 800px) and (min-width: 320px) {
        .photo {
            width: 70px;
            height: 70px;
        }
    }
</style>

```

{% previewsample "page.domainurl/code-snippet/grid/column/template-cs4" %}

#### [Limitations for detail template](#)

Detail template is not supported with the following features:

- Frozen rows and columns
- Immutable mode
- Infinite scrolling
- Virtual scrolling
- Print
- Row template
- Row spanning
- Column spanning
- Lazy load grouping
- State persistence
- Hierarchy Grid

#### [Row drag and drop in Vue Grid component](#)

The Grid Row drag and drop allows you to drag grid rows and drop to another Grid or custom component. To enable Row drag and drop in the Grid, set the [allowRowDragAndDrop](#) to true. The target component on which the Grid rows to be dropped can be set by using [targetID](#).

To use Row Drag and Drop, you need to inject **RowDD** in the **provide** section.

#### **APP.VUE**

```

<template>
  <div id="app">
    <ejs-grid id='Grid' :dataSource='srcData' :allowPaging="true"
    :pageSettings="pageOptions" :allowSelection="true"
    :allowRowDragAndDrop="true"
      :selectionSettings="selectionOptions"
    :rowDropSettings="srcDropOptions" width="49%">
      <e-columns>
        <e-column field='OrderID' headerText='Order ID' width='120'
        textAlign='Right'></e-column>
        <e-column field='CustomerID' headerText='Customer Name'
        width='130'></e-column>
        <e-column field='Freight' headerText='Freight' width='120'
        format='C2' textAlign='Right'></e-column>
      </e-columns>
    </ejs-grid>
  </div>
</template>

```

```

        </ejs-grid>
        <ejs-grid id='DestGrid' :dataSource='destData' :allowPaging="true"
:pageSettings="pageOptions" :allowSelection="true"
        :allowRowDragAndDrop="true"
:selectionSettings="selectionOptions" :rowDropSettings="destDropOptions"
width="49%">
            <e-columns>
                <e-column field='OrderID' headerText='Order ID' width='120'
textAlign='Right'></e-column>
                <e-column field='CustomerID' headerText='Customer Name'
width='130'></e-column>
                <e-column field='Freight' headerText='Freight' width='120'
format='C2' textAlign='Right'></e-column>
            </e-columns>
        </ejs-grid>
    </div>
</template>
<script>
import Vue from "vue";
import { GridPlugin, RowDD, Selection, Page } from "@syncfusion/ej2-vue-
grids";
import { data } from './datasource.js';
Vue.use(GridPlugin);
export default {
    data() {
        return {
            srcData: data,
            pageSettings: { pageSize: 7 },
            destData: [],
            pageOptions: { pageSize: 7 },
            selectionOptions: { type: "Multiple" },
            srcDropOptions: { targetID: "DestGrid" },
            destDropOptions: { targetID: "Grid" }
        };
    },
    provide: {
        grid: [RowDD, Page, Selection]
    }
}
</script>
<style>
    @import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
    #Grid {
        float: left;
    }
    #DestGrid {
        float: right;
    }
</style>

```

{% previewsample "page.domainurl/code-snippet/grid/row/default-cs1" %}

#### *Drag and drop rows without drag icon*

You can hide the drag and drop icon when performing a drag and drop operation within the grid. This can be achieved by setting the [targetID](#) of the [rowDropSettings](#) as the current Grid's ID.



By setting the [targetID](#), the Grid will render without a helper icon (for row drag). Now you can customize the drag and drop action. To control the drop action, you can bind the [rowDrop](#) event of the Grid. In the [rowDrop](#) event, you can prevent the default action(`args.cancel as true`) and reorder the rows using [reorderRows](#) method of the Grid.

### APP.VUE

```
<template>
  <div id="app">
    <ejs-grid ref='grid' id='Grid' :dataSource='data'
      :selectionSettings="selectionOptions" :rowDropSettings="dropOptions"
      height='273px' :rowDrop="rowDrop" :allowSelection="true"
      :allowRowDragAndDrop="true">
      <e-columns>
        <e-column field='OrderID' headerText='Order ID'
          isPrimaryKey={true} textAlign='Right' width=100></e-column>
        <e-column field='CustomerID' headerText='Customer ID'
          width=120></e-column>
        <e-column field='Freight' headerText='Freight' textAlign=
          'Right' width=120 format= 'C2'></e-column>
        <e-column field='ShipCountry' headerText='Ship Country'
          width=150></e-column>
      </e-columns>
    </ejs-grid>
  </div>
</template>
<script>
import Vue from "vue";
import { GridPlugin, RowDD, Selection } from "@syncfusion/ej2-vue-grids";
import { data } from '../datasource.js';
Vue.use(GridPlugin);
export default {
  data() {
    return {
      data: data,
      selectionOptions: { type: "Multiple" },
      dropOptions: { targetID: "Grid" },
    };
  },
  methods: {
    rowDrop(args) {
      args.cancel = true;
      var value = [];
      for (var r = 0; r < args.rows.length; r++) {
        value.push(args.fromIndex + r);
      }
      this.$refs.grid.reorderRows(value, args.dropIndex);
    }
  },
  provide: {
    grid: [RowDD, Selection]
  }
}
</script>
<style>
  @import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
</style>
```

```
{% previewsample "page.domainurl/code-snippet/grid/row/default-cs2" %}
```

\* Selection feature must be enabled for row drag and drop.

\* Multiple rows can be selected by clicking and dragging inside the grid. For multiple row selection, the [type](#) property must be set to **Multiple**.

#### *Limitations of row drag and drop*

- Multiple rows can be drag and drop in the row selections basis.
- Single row is able to drag and drop in same grid without enable the row selection.
- Row drag and drop feature is not having built in support with sorting, filtering, hierarchy grid and grouping features of grid.

#### Row spanning in Vue Grid component

The grid has option to span row cells. To achieve this, You need to define the [rowSpan](#) attribute to span cells in the [QueryCellInfo](#) event.

In the following demo, **Davolio** cell is spanned to two rows in the **EmployeeName** column.

Also Grid supports the spanning of rows and columns for same cells. **Lunch Break** cell is spanned to two rows and three columns in the **1:00** column.

#### **APP.VUE**

```
<template>
  <div id="app">
    <ejs-grid :dataSource="data" height='300' width='auto'
    gridLines='Both' :allowTextWrap='true' :queryCellInfo='queryCellInfoEvent'>
      <e-columns>
        <e-column field='EmployeeID' headerText='Employee ID'
width='150' textAlign='Right' isPrimaryKey={true}></e-column>
        <e-column field='EmployeeName' headerText='Employee Name'
width='200'></e-column>
        <e-column field='9:00' headerText='9:00 AM' width='120'></e-
column>
        <e-column field='9:30' headerText='9:30 AM' width='120'></e-
column>
        <e-column field='10:00' headerText='10:00 AM'
width='120'></e-column>
        <e-column field='10:30' headerText='10:30 AM'
width='120'></e-column>
        <e-column field='11:00' headerText='11:00 AM'
width='120'></e-column>
        <e-column field='11:30' headerText='11:30 AM'
width='120'></e-column>
        <e-column field='12:00' headerText='12:00 PM'
width='120'></e-column>
        <e-column field='12:30' headerText='12:30 PM'
width='120'></e-column>
        <e-column field='1:00' headerText='1:00 PM' width='120'></e-
column>
        <e-column field='1:30' headerText='1:30 PM' width='120'></e-
column>
      </e-columns>
    </ejs-grid>
  </div>
</template>
```

```

column>      <e-column field='2:00' headerText='2:00 PM' width='120'></e-
column>
column>      <e-column field='2:30' headerText='2:30 PM' width='120'></e-
column>
column>      <e-column field='3:00' headerText='3:00 PM' width='120'></e-
column>
column>      <e-column field='3:30' headerText='3:30 PM' width='120'></e-
column>
column>      <e-column field='4:00' headerText='4:00 PM' width='120'></e-
column>
column>      <e-column field='4:30' headerText='4:30 PM' width='120'></e-
column>
column>      <e-column field='5:00' headerText='5:00 PM' width='120'></e-
column>
      </e-columns>
    </ejs-grid>
  </div>
</template>
<script>
import Vue from 'vue';
import { GridPlugin } from '@syncfusion/ej2-vue-grids';
import { data } from './datasource.js';
Vue.use(GridPlugin);
export default {
  data: () => {
    return {
      data: data
    }
  },
  methods: {
    queryCellInfoEvent: function(args) {
      let data = args.data;
      switch (data.EmployeeID) {
        case 10001:
          if (args.column.field === '9:00' || args.column.field === '2:30'
|| args.column.field === '4:30') {
            args.colSpan = 2;
          } else if (args.column.field === '11:00') {
            args.colSpan = 3;
          } else if (args.column.field === 'EmployeeName') {
            args.rowSpan = 2;
          } else if (args.column.field === '1:00') {
            args.colSpan = 3;
            args.rowSpan = 2;
          }
          break;
        case 10002:
          if (args.column.field === '9:30' || args.column.field === '2:30'
||
            args.column.field === '4:30') {
            args.colSpan = 3;
          } else if (args.column.field === '11:00') {
            args.colSpan = 4;
          }
          break;
        case 10003:

```

```

        if (args.column.field === '9:00' || args.column.field ===
'11:30') {
            args.colSpan = 3;
        } else if (args.column.field === '10:30' || args.column.field
=== '3:30' ||
            args.column.field === '4:30' || args.column.field ===
'2:30') {
            args.colSpan = 2;
        }
        break;
    case 10004:
        if (args.column.field === '9:00') {
            args.colSpan = 3;
        } else if (args.column.field === '11:00') {
            args.colSpan = 4;
        } else if (args.column.field === '4:00' || args.column.field ===
'2:30') {
            args.colSpan = 2;
        }
        break;
    case 10005:
        if (args.column.field === '9:00') {
            args.colSpan = 4;
        } else if (args.column.field === '11:30') {
            args.colSpan = 3;
        } else if (args.column.field === '3:30' || args.column.field ===
'4:30' || args.column.field === '2:30') {
            args.colSpan = 2;
        }
        break;
    case 10006:
        if (args.column.field === '9:00' || args.column.field === '4:30'
||
            args.column.field === '2:30' || args.column.field ===
'3:30') {
            args.colSpan = 2;
        } else if (args.column.field === '10:00' || args.column.field
=== '11:30') {
            args.colSpan = 3;
        }
        break;
    case 10007:
        if (args.column.field === '9:00' || args.column.field === '3:00'
|| args.column.field === '10:30') {
            args.colSpan = 2;
        } else if (args.column.field === '11:30' || args.column.field
=== '4:00') {
            args.colSpan = 3;
        }
        break;
    default:
        this.extendQueryCellEvent(args, data.EmployeeID);
    }
},
extendQueryCellEvent: function(args, value) {
    switch (value) {
        case 10008:

```

```

        if (args.column.field === '9:00' || args.column.field ===
'10:30' || args.column.field === '2:30') {
            args.colSpan = 3;
        } else if (args.column.field === '4:00') {
            args.colSpan = 2;
        }
        break;
    case 10009:
        if (args.column.field === '9:00' || args.column.field ===
'11:30') {
            args.colSpan = 3;
        } else if (args.column.field === '4:30' || args.column.field ===
'2:30') {
            args.colSpan = 2;
        }
        break;
    case 10010:
        if (args.column.field === '9:00' || args.column.field === '2:30'
||
        args.column.field === '4:00' || args.column.field ===
'11:30') {
            args.colSpan = 3;
        } else if (args.column.field === '10:30') {
            args.colSpan = 2;
        }
        break;
    }
}
});
</script>
<style>
    @import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/grid/column/spanning-cs2" %}

To disable the spanning for particular grid page, we need to use `requestType` from [QueryCellInfo](#) event argument.

### Limitations

- Row spanning is not compatible with the following features:
  1. Virtual scrolling
  2. Infinite scrolling
  3. Lazy load grouping
  4. Row drag and drop

## Cell

### Editing

#### Edit in Vue Grid component

The Grid component has options to dynamically insert, delete and update records. Editing feature requires a primary key column for CRUD operations. To define the primary key, set [columns.isPrimaryKey](#) to **true** in particular column.

You can start the edit action either by double clicking the particular row or by selecting the required row and click on **Edit** button in the toolbar. Similarly, you can add a new record to grid either by clicking on **Add** button in the toolbar or on an external button which is bound to invoke the [addRecord](#) method of the grid, **Save** and **Cancel** while in edit mode is possible using respective toolbar icon in grid.

Deletion of the record is possible by selecting the required row and click on **Delete** button in the toolbar.

To use CRUD, inject the [Edit](#) module in the [Link to the Video](#) section.

To get start quickly with Edit Options, you can check on this video:

#### APP.VUE

```
<template>
  <div id="app">
    <ejs-grid :dataSource='data' :editSettings='editSettings'
height='315px'>
      <e-columns>
        <e-column field='OrderID' headerText='Order ID'
textAlign='Right' :isPrimaryKey='true' width=100></e-column>
        <e-column field='CustomerID' headerText='Customer ID'
width=120></e-column>
        <e-column field='Freight' headerText='Freight' textAlign=
'Right' width=120 format= 'C2'></e-column>
        <e-column field='ShipCountry' headerText='Ship Country'
width=150></e-column>
      </e-columns>
    </ejs-grid>
  </div>
</template>
<script>
import Vue from "vue";
import { GridPlugin, Page, Edit } from "@syncfusion/ej2-vue-grids";
import { data } from '../datasource.js';
Vue.use(GridPlugin);
export default {
  data() {
    return {
      data: data,
      editSettings: { allowEditing: true, allowAdding: true, allowDeleting:
true }
    };
  },
  provide: {
    grid: [Page, Edit]
  }
}
</script>
<style>
```

```
@import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
</style>
```

{% previewsample "page.domainurl/code-snippet/grid/edit/default-cs20" %}

\* If [columns.isIdentity](#) is enabled, then it will be considered as a read-only column when editing and adding a record.

\* You can disable editing for a particular column, by specifying [columns.allowEditing](#) to **false**.

#### *Toolbar with edit option*

The grid toolbar has the [built-in items](#) to execute Editing actions. You can define this by using the [toolbar](#) property.

#### **APP.VUE**

```
<template>
  <div id="app">
    <ejs-grid :dataSource='data' :editSettings='editSettings'
height='270' :toolbar='toolbar'>
      <e-columns>
        <e-column field='OrderID' headerText='Order ID'
textAlign='Right' :isPrimaryKey='true' width=100></e-column>
        <e-column field='CustomerID' headerText='Customer ID'
width=120></e-column>
        <e-column field='Freight' headerText='Freight' textAlign=
'Right' width=120 format= 'C2'></e-column>
        <e-column field='ShipCountry' headerText='Ship Country'
width=150></e-column>
      </e-columns>
    </ejs-grid>
  </div>
</template>
<script>
import Vue from "vue";
import { GridPlugin, Page, Toolbar, Edit } from "@syncfusion/ej2-vue-grids";
import { data } from './datasource.js';
Vue.use(GridPlugin);
export default {
  data() {
    return {
      data: data,
      editSettings: { allowEditing: true, allowAdding: true, allowDeleting:
true },
      toolbar: ['Add', 'Edit', 'Delete', 'Update', 'Cancel']
    };
  },
  provide: {
    grid: [Page, Edit, Toolbar]
  }
}
</script>
<style>
  @import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
</style>
```

```
{% previewsample "page.domainurl/code-snippet/grid/edit/default-cs21" %}
```

### *Disable editing for particular column*

You can disable editing for particular columns by using the [columns.allowEditing](#).

In the following demo, editing is disabled for the **CustomerID** column.

### **APP.VUE**

```
<template>
  <div id="app">
    <ejs-grid :dataSource='data' :editSettings='editSettings'
:toolbar='toolbar' height='273px'>
      <e-columns>
        <e-column field='OrderID' headerText='Order ID'
textAlign='Right' :isPrimaryKey='true' :validationRules='orderIDRules'
width=100></e-column>
        <e-column field='CustomerID' headerText='Customer ID'
:allowEditing='false' :validationRules='customerIDRules' width=120></e-
column>
        <e-column field='Freight' headerText='Freight' textAlign=
'Right' editType= 'numericedit' width=120 format= 'C2'></e-column>
        <e-column field='ShipCountry' headerText='Ship Country'
editType= 'dropdownedit' width=150></e-column>
      </e-columns>
    </ejs-grid>
  </div>
</template>
<script>
import Vue from "vue";
import { GridPlugin, Page, Toolbar, Edit } from "@syncfusion/ej2-vue-grids";
import { data } from '../datasource.js';
Vue.use(GridPlugin);
export default {
  data() {
    return {
      data: data,
      editSettings: { allowEditing: true, allowAdding: true, allowDeleting:
true, mode: 'Normal' },
      toolbar: ['Add', 'Edit', 'Delete', 'Update', 'Cancel'],
      orderIDRules: { required: true },
      customerIDRules: { required: true, minLength: 3 }
    };
  },
  provide: {
    grid: [Page, Edit, Toolbar]
  }
}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
</style>
```

```
{% previewsample "page.domainurl/code-snippet/grid/edit/default-cs22" %}
```



*Disable editing for a particular row or cell*

You can disable the editing for a particular row by using the [actionBegin](#) event of Grid based on `requestType` as `beginEdit`.

In the below demo, the rows which are having the value for `ShipCountry` column as "France" is prevented from editing.

**APP.VUE**

```
<template>
  <div id="app">
    <ejs-grid :dataSource='data' :editSettings='editSettings'
:toolbar='toolbar' height='273px' :actionBegin = 'actionBegin'>
      <e-columns>
        <e-column field='OrderID' headerText='Order ID'
textAlign='Right' :isPrimaryKey='true' width=100></e-column>
        <e-column field='CustomerID' headerText='Customer ID'
width=120></e-column>
        <e-column field='Freight' headerText='Freight' textAlign=
'Right' editType= 'numericedit' width=120 format= 'C2'></e-column>
        <e-column field='ShipCountry' headerText='Ship Country'
editType= 'dropdownedit' width=150></e-column>
      </e-columns>
    </ejs-grid>
  </div>
</template>
<script>
import Vue from "vue";
import { GridPlugin, Page, Toolbar, Edit } from "@syncfusion/ej2-vue-grids";
import { data } from './datasource.js';
Vue.use(GridPlugin);
export default {
  data() {
    return {
      data: data,
      editSettings: { allowEditing: true },
      toolbar: ['Edit', 'Update', 'Cancel']
    };
  },
  methods: {
    actionBegin(args) {
      if (args.requestType === 'beginEdit') {
        if (args.rowData.ShipCountry == "France") {
          args.cancel = true;
        }
      }
    }
  },
  provide: {
    grid: [Page, Edit, Toolbar]
  }
}
</script>
<style>
  @import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
</style>
```

```
{% previewsample "page.domainurl/code-snippet/grid/edit/default-cs23" %}
```

For batch mode of editing, you can use [cellEdit](#) event of Grid. In the below demo, the cells which are having the value as "France" is prevented from editing.

### APP.VUE

```
<template>
  <div id="app">
    <ejs-grid :dataSource='data' :editSettings='editSettings'
:toolbar='toolbar' height='273px' :cellEdit = 'cellEdit'>
      <e-columns>
        <e-column field='OrderID' headerText='Order ID'
textAlign='Right' :isPrimaryKey='true' width=100></e-column>
        <e-column field='CustomerID' headerText='Customer ID'
width=120></e-column>
        <e-column field='Freight' headerText='Freight' textAlign=
'Right' editType= 'numericedit' width=120 format= 'C2'></e-column>
        <e-column field='ShipCountry' headerText='Ship Country'
editType= 'dropdownedit' width=150></e-column>
      </e-columns>
    </ejs-grid>
  </div>
</template>
<script>
import Vue from "vue";
import { GridPlugin, Page, Toolbar, Edit } from "@syncfusion/ej2-vue-grids";
import { data } from './datasource.js';
Vue.use(GridPlugin);
export default {
  data() {
    return {
      data: data,
      editSettings: { allowEditing: true, mode: 'Batch' },
      toolbar: ['Edit', 'Update', 'Cancel']
    };
  },
  methods: {
    cellEdit(args) {
      if (args.value == "France") {
        args.cancel = true;
      }
    }
  },
  provide: {
    grid: [Page, Edit, Toolbar]
  }
}
</script>
<style>
  @import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
</style>
```

```
{% previewsample "page.domainurl/code-snippet/grid/edit/default-cs24" %}
```

*Editing template column*

You can edit template column value by defining **field** for that particular column.

In the below demo, the **ShipCountry** column is rendered with the template.

**APP.VUE**

```
<template>
  <div id="app">
    <ejs-grid :dataSource='data' :editSettings='editSettings'
:toolbar='toolbar' height='280px' >
      <e-columns>
        <e-column field='OrderID' headerText='Order ID'
:isPrimaryKey='true' textAlign='Right' width=100></e-column>
        <e-column field='CustomerID' headerText='Customer ID'
width=120></e-column>
        <e-column field='Freight' headerText='Freight'
textAlign='Center' format='C2' editType='numericedit' width=80></e-column>
        <e-column field='ShipCountry' headerText='Ship Country'
:template='editTemplate' editType='dropdownedit' width=120></e-column>
      </e-columns>
    </ejs-grid>
  </div>
</template>
<script>
import Vue from "vue";
import { GridPlugin, Toolbar, Edit } from "@syncfusion/ej2-vue-grids";
import { data } from './datasource.js';
Vue.use(GridPlugin);
export default {
  data: () => {
    return {
      data: data,
      editSettings: { allowEditing: true, allowAdding: true, allowDeleting:
true },
      toolbar: ['Add', 'Edit', 'Delete', 'Update', 'Cancel'],
      editTemplate: function () {
        return {
          template: Vue.component('editOption', {
            template: '<a href="#">{{data.ShipCountry}}</a>',
            data() { return { data: { data: {} } }; }
          })
        }
      },
    };
  },
  provide: {
    grid: [Edit, Toolbar]
  }
}
</script>
<style>
  @import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
</style>
```

{% previewsample "page.domainurl/code-snippet/grid/how-to/default-cs8" %}

*Troubleshoot editing works only for first row*

The Editing functionalities can be performed based upon the primary key value of the selected row. If **primaryKey** is not defined in the grid, then edit or delete action takes place the first row.

*How to make a Grid column always editable*

Make the Grid column always editable using the column template feature of the Grid.

In the following example, the textbox is rendered in the Freight column using a column template. The keyup event for the Grid is bound using the [created](#) event of the Grid, and the edited changes are saved in the data source using the [updateRow](#) method of the Grid.

**APP.VUE**

```
<template>
  <div id="app">
    <ejs-grid ref="grid" :dataSource='data' height='315px' :created =
'created'>
      <e-columns>
        <e-column field='OrderID' headerText='Order ID'
textAlign='Right' :isPrimaryKey='true' width=120></e-column>
        <e-column field='OrderDate' headerText='Order Date'
width=130 textAlign='Right' format='yMd'></e-column>
        <e-column field='ShipCountry' headerText='Ship Country'
width=120></e-column>
        <e-column field='Freight' headerText='Receipt Amount'
width=120 :template='cTemplate'></e-column>
      </e-columns>
    </ejs-grid>
  </div>
</template>
<script>
import Vue from "vue";
import { GridPlugin, parentsUntil } from "@syncfusion/ej2-vue-grids";
import { data } from './datasource.js';
Vue.use(GridPlugin);
export default {
  data() {
    return {
      data: data,
      cTemplate: function () {
        return { template : Vue.component('columnTemplate',{
          template: `<div class="image">
            <input :id='id' :value='value' class='custemp'
type='text' style='width: 100%' />
          </div>`,
          data: function() {
            return {
              data: {}
            }
          },
          computed: {
            id: function() {
              return this.data.OrderID;
            },
            value: function() {
              return this.data.Freight;
            }
          }
        });
      }
    };
  }
};
```

```

    }
    }
  }
};
},
methods: {
  created(args) {
    this.$refs.grid.ej2Instances.element.addEventListener('keyup',
function (e) { // Bind the keyup event for the grid.
  if (e.target.classList.contains('custemp')) { // Based on this
condition, you can find whether the target is an input element or not.
    var row = parentsUntil(e.target, 'e-row');
    var rowIndex = row.rowIndex; // Get the row index.
    var uid = row.getAttribute('data-uid');
    var grid = document.getElementsByClassName('e-
grid')[0].ej2_instances[0];
    var rowData = grid.getRowObjectFromUID(uid).data; // Get the
row data.
    rowData.Freight = e.target.value; // Update the new value
for the corresponding column.
    grid.updateRow(rowIndex, rowData); // Update the modified
value in the row data.
  }
});
}
},
}
}
</script>
<style>
  @import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/grid/edit/default-cs25" %}

[See Also](#)

- [Cascading DropDownList with Grid Editing](#)
- [Dialog editing for Vue Grid using WebAPI adaptor](#)
- [Is it possible to hide the tool bar or some of the CRUD operations in Vue Grid?](#)
- [How to pass a data from parent to child in Vue Grid](#)
- [Need to use different key value while performing delete in Vue Grid](#)
- [How to give custom Headertext name and validation in ChildData Vue Grid](#)
- [Calculate column value based on another column with filtering in Vue Grid](#)

Edit types in Vue Grid component

[Customize editors using params](#)

The [columns.editType](#) is used to define the editor component for any particular column. You can set the [columns.editType](#) based on data type of the column.

- [NumericTextBox](#) component for integers, double, and decimal data types.
- [TextBox](#) component for string data type.
- [DropDownList](#) component to show all unique values related to that field.

- [Checkbox](#) component for boolean type.
- [DatePicker](#) component for date type.

Also, you can customize model of the [columns.editType](#) component through the [columns.edit.params](#).

The following table describes cell edit type component and their corresponding edit params of the column.

Component | Example

[NumericTextBox](#) | params: { decimals: 2, value: 5 }

[TextBox](#) | -

[DropDownList](#) | params: { value: 'Germany' }

[DatePicker](#) | params: { value: new Date() }

[Checkbox](#) | params: { checked: true }

### APP.VUE

```
<template>
  <div id="app">
    <ejs-grid :dataSource='data' :editSettings='editSettings'
:toolbar='toolbar' height='273px'>
      <e-columns>
        <e-column field='OrderID' headerText='Order ID'
textAlign='Right' :isPrimaryKey='true' width=100></e-column>
        <e-column field='CustomerID' headerText='Customer ID'
width=120></e-column>
        <e-column field='Freight' headerText='Freight' textAlign=
'Right' editType= 'numericedit' :edit='numericParams' width=120 format=
'C2'></e-column>
        <e-column field='ShipCountry' headerText='Ship Country'
editType= 'dropdownedit' :edit='ddParams' width=150></e-column>
        <e-column field='ShippedDate' headerText='Shipped Date'
editType= 'datepickeredit' :edit='dpParams' width=150></e-column>
        <e-column field='Verified' headerText='Verified' editType=
'booleanedit' :edit='boolParams' width=150></e-column>
      </e-columns>
    </ejs-grid>
  </div>
</template>
<script>
import Vue from "vue";
import { GridPlugin, Page, Toolbar, Edit } from "@syncfusion/ej2-vue-grids";
import { data } from './datasource.js';
Vue.use(GridPlugin);
export default {
  data() {
    return {
      data: data,
      editSettings: { allowEditing: true, allowAdding: true, allowDeleting:
true },
      toolbar: ['Add', 'Edit', 'Delete', 'Update', 'Cancel'],
      numericParams: { params: { decimals: 2, value: 5 } },
      ddParams: { params: { value: 'Germany' } },
    }
  }
}
```

```

        dpParams: { params: {value: new Date() } },
        boolParams: { params: {checked: true } }
    };
},
provide: {
    grid: [Page, Edit, Toolbar]
}
}
</script>
<style>
    @import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/grid/edit/default-cs12" %}

If edit type is not defined in the column, then it will be considered as the **stringedit** type (Textbox component).

#### Restrict to type decimal points in a NumericTextBox while editing the numeric column

By default, the number of decimal places will be restricted to two in the NumericTextBox while editing the numeric column. We can restrict to type the decimal points in a NumericTextBox by using the **validateDecimalOnType** and **decimals** properties of NumericTextBox.

In the below demo, while editing the row we have restricted to type the decimal point value in the NumericTextBox of **Freight** column.

#### APP.VUE

```

<template>
    <div id="app">
        <ejs-grid :dataSource='data' :editSettings='editSettings'
        :toolbar='toolbar' height='265px' >
            <e-columns>
                <e-column field='OrderID' headerText='Order ID'
                :isPrimaryKey='true' textAlign='Right' width=100></e-column>
                <e-column field='CustomerID' headerText='Customer ID'
                width=120></e-column>
                <e-column field='Freight' headerText='Freight'
                textAlign='Center' editType='numericedit' :edit='numericParams'
                width=80></e-column>
                <e-column field='ShipCity' headerText='Ship City'
                editType='dropdownedit' width=120></e-column>
            </e-columns>
        </ejs-grid>
    </div>
</template>
<script>
import Vue from "vue";
import { GridPlugin, Toolbar, Edit } from "@syncfusion/ej2-vue-grids";
import { data } from './datasource.js';
Vue.use(GridPlugin);
export default {
    data: () => {
        return {
            data: data,

```

```

        editSettings: { allowEditing: true, allowAdding: true, allowDeleting:
true },
        toolbar: ['Add', 'Edit', 'Delete', 'Update', 'Cancel'],
        numericParams: { params: {
            validateDecimalOnType: true,
            decimals: 0,
            format: "N" }
        },
    };
},
provide: {
    grid: [Edit, Toolbar]
}
}
</script>
<style>
    @import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/grid/how-to/default-cs6" %}

[Provide custom data source and enabling filtering to DropDownList](#)

You can provide data source to the DropDownList by using the `columns.edit.params` property.

While setting new data source using edit params, you must specify a new `query` property too for the DropDownList as follows,

```

`ts
countryParams: {
  params: {
    allowFiltering: true,
    dataSource: country,
    fields: {text:"countryName",value:"countryName"},
    query: new Query(),
    actionComplete: () => false
  }
};
`

```

You can also enable filtering for the DropDownList by passing the `allowFiltering` as `true` to the edit params.

In the below demo, DropDownList is rendered with custom Datasource for the `ShipCountry` column and enabled filtering to search DropDownList items.

#### APP.VUE

```

<template>
  <div id="app">

```



```

    <ejs-grid ref='grid' :dataSource='data' :editSettings='editSettings'
:toolbar='toolbar' height='273px' >
        <e-columns>
            <e-column field='OrderID' headerText='Order ID'
:isPrimaryKey='true' textAlign='Right' width=100></e-column>
            <e-column field='CustomerID' headerText='Customer ID'
width=120></e-column>
            <e-column field='ShipCountry' headerText='ShipCountry'
editType='dropdownedit' :edit='countryParams' width=150></e-column>
        </e-columns>
    </ejs-grid>
</div>
</template>
<script>
import Vue from "vue";
import { createElement } from '@syncfusion/ej2-base';
import { GridPlugin, Toolbar, Edit } from "@syncfusion/ej2-vue-grids";
import { DropDownList } from "@syncfusion/ej2-dropdowns";
import { Query } from '@syncfusion/ej2-data';
import { cascadeData } from './datasource.js';
let country= [
    { countryName: 'United States', countryId: '1' },
    { countryName: 'Australia', countryId: '2' },
    { countryName: 'India', countryId: '3' }
];
Vue.use(GridPlugin);
export default {
    data: () => {
        return {
            data: cascadeData,
            toolbar: ['Add', 'Edit', 'Delete', 'Update', 'Cancel'],
            editSettings: { allowEditing: true, allowAdding: true,
allowDeleting: true },
            countryParams: {
                params: {
                    allowFiltering: true,
                    dataSource: country,
                    fields: {text:"countryName",value:"countryName"},
                    query: new Query(),
                    actionComplete: () => false
                }
            }
        };
    },
    provide: {
        grid: [Edit, Toolbar]
    }
}
</script>
<style>
    @import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/grid/how-to/default-cs7" %}

### Custom editors using template

The cell edit template is used to add a custom component for a particular column by invoking the following functions:

- **create** - It is used to create the element at the time of initialization.
- **write** - It is used to create custom component or assign default value at the time of editing.
- **read** - It is used to read the value from the component at the time of save.
- **destroy** - It is used to destroy the component.

### Render TimePicker component while editing

Use the cell edit template feature of the Grid to render the TimePicker component in the Grid edit form. In the below sample, we have rendered TimePicker component in the **OrderDate** column.

### APP.VUE

```
<template>
  <div id="app">
    <ejs-grid ref='grid' :dataSource='data' :allowPaging='true'
    :editSettings='editSettings' :pageSettings='pageSettings' :toolbar='toolbar'
    height='175px'>
      <e-columns>
        <e-column field='OrderID' headerText='Order ID'
        type='number' textAlign='Right' :isPrimaryKey='true'
        :validationRules='orderIDRules' width=100></e-column>
        <e-column field='CustomerID' headerText='Customer ID'
        type='string' width=120>
        </e-column>
        <e-column field='Freight' headerText='Freight' type='number'
        textAlign='Right' editType='numericedit' format='C2' width=120></e-column>
        <e-column field='OrderDate' headerText='Order Date'
        type='date' textAlign='Right' format="hh:mm" :edit='dpParams' width=120></e-
        column>
      </e-columns>
    </ejs-grid>
  </div>
</template>
<script>
import Vue from "vue";
import { GridPlugin, Page, Toolbar, Edit } from "@syncfusion/ej2-vue-grids";
import { TimePicker } from "@syncfusion/ej2-calendars";
import { enableRipple } from "@syncfusion/ej2-base";
import { purchaseData } from '../datasource.js';
Vue.use(GridPlugin);
let ddElem;
let timeObject;
function createOrderDateFn() {
  ddElem = document.createElement('input');
  return ddElem;
}
function destroyOrderDateFn() {
  timeObject.destroy();
}
function readOrderDateFn() {
  return timeObject.value;
}
</script>
```

```
function writeOrderDateFn(args) {
  enableRipple(true);
  timeObject = new TimePicker({
    value: args.rowData[args.column.field],
    step: 60,
  });
  timeObject.appendTo(ddElem);
}
export default {
  data() {
    return {
      data: purchaseData,
      pageSettings: { pageSizes: true, pageSize: 5 },
      editSettings: { allowEditing: true, allowAdding: true, allowDeleting:
true },
      dpParams: {
        create: createOrderDateFn,
        destroy: destroyOrderDateFn,
        read: readOrderDateFn,
        write: writeOrderDateFn,
      },
      toolbar: ['Add', 'Edit', 'Delete', 'Update', 'Cancel'],
      orderIDRules: { required: true },
    };
  },
  provide: {
    grid: [Page, Edit, Toolbar],
  },
};
</script>
<style>
@import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
</style>
```

{% previewsample "page.domainurl/code-snippet/grid/edit/default-cs13" %}

#### Render AutoComplete component while editing

Use the cell edit template feature of the Grid to render the AutoComplete component in the Grid edit form. In the below sample, we have rendered AutoComplete component in the **CustomerID** column.

#### APP.VUE

```
<template>
  <div id="app">
    <ejs-grid ref='grid' :dataSource='data' :allowPaging='true'
:editSettings='editSettings' :pageSettings='pageSettings' :toolbar='toolbar'
height='175px'>
      <e-columns>
        <e-column field='OrderID' headerText='Order ID'
type='number' textAlign='Right' :isPrimaryKey='true'
:validationRules='orderIDRules' width=100></e-column>
        <e-column field='CustomerID' headerText='Customer ID'
type='string' :edit='dpParams' width=140></e-column>
        <e-column field='Freight' headerText='Freight' type='number'
textAlign='Right' editType='numericedit' format='C2' width=120></e-column>
```

```

        <e-column field='OrderDate' headerText='Order Date'
type='date' editType='datepickeredit' textAlign='Right' format='yMd'
width=150></e-column>
    </e-columns>
</ejs-grid>
</div>
</template>
<script>
import Vue from "vue";
import { GridPlugin, Page, Toolbar, Edit } from "@syncfusion/ej2-vue-grids";
import { AutoComplete } from "@syncfusion/ej2-dropdowns";
import { purchaseData } from './datasource.js';
Vue.use(GridPlugin);
let inpuEle;
let autoCompleteIns;
let autoCompleteData = [
    { CustomerID: 'VINET', Id: '1' },
    { CustomerID: 'TOMSP', Id: '2' },
    { CustomerID: 'HANAR', Id: '3' },
    { CustomerID: 'VICTE', Id: '4' },
    { CustomerID: 'SUPRD', Id: '5' },
];
function createCustomerIDFn() {
    inpuEle = document.createElement('input');
    return inpuEle;
}
function destroyCustomerIDFn() {
    autoCompleteIns.destroy();
}
function readCustomerIDFn() {
    return autoCompleteIns.value;
}
function writeCustomerIDFn(args) {
    autoCompleteIns = new AutoComplete({
        allowCustom: true,
        value: args.rowData[args.column.field],
        dataSource: autoCompleteData,
        fields: { value: "CustomerID", text: "CustomerID" },
    });
    autoCompleteIns.appendTo(inpuEle);
}
export default {
    data() {
        return {
            data: purchaseData,
            pageSettings: { pageSizes: true, pageSize: 5 },
            editSettings: { allowEditing: true, allowAdding: true, allowDeleting:
true },
            daParams: {
                create: createCustomerIDFn,
                destroy: destroyCustomerIDFn,
                read: readCustomerIDFn,
                write: writeCustomerIDFn,
            },
            toolbar: ['Add', 'Edit', 'Delete', 'Update', 'Cancel'],
            orderIDRules: { required: true },
        };
    }
};

```

```

    },
    provide: {
      grid: [Page, Edit, Toolbar],
    },
  };
</script>
<style>
@import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/grid/edit/default-cs14" %}

[Render MultiSelect DropDown component while editing](#)

Use the cell edit template feature of the Grid to render the MultiSelect DropDown component in the Grid edit form. In the below sample, we have rendered MultiSelect DropDown component in the **ShipCity** column.

#### APP.VUE

```

<template>
  <div id="app">
    <ejs-grid ref='grid' :dataSource='data' :allowPaging='true'
    :editSettings='editSettings' :pageSettings='pageSettings' :toolbar='toolbar'
    height='175px'>
      <e-columns>
        <e-column field='OrderID' headerText='Order ID'
        type='number' textAlign='Right' :isPrimaryKey='true'
        :validationRules='orderIDRules' width=100></e-column>
        <e-column field='CustomerID' headerText='Customer ID'
        type='string' width=120></e-column>
        <e-column field='Freight' headerText='Freight' type='number'
        textAlign='Right' editType='numericedit' format='C2' width=120></e-column>
        <e-column field='ShipCity' headerText='Ship City'
        type='string' :edit='dsParams' width=150></e-column>
      </e-columns>
    </ejs-grid>
  </div>
</template>
<script>
import Vue from "vue";
import { GridPlugin, Page, Toolbar, Edit } from "@syncfusion/ej2-vue-grids";
import { MultiSelect } from "@syncfusion/ej2-dropdowns";
import { purchaseData } from './datasource.js';
Vue.use(GridPlugin);
let ddElem;
let multiSelectObj;
let multiSelectDatasource = [
  { ShipCity: 'Reims', Id: '1' },
  { ShipCity: 'Münster', Id: '2' },
  { ShipCity: 'Rio de Janeiro', Id: '3' },
  { ShipCity: 'Lyon', Id: '4' },
  { ShipCity: 'Charleroi', Id: '5' },
];
function createShipCityFn() {
  ddElem = document.createElement('input');

```

```

    return ddElem;
  }
  function readShipCityFn() {
    return multiSelectObj.value.join(",");
  }
  function destroyShipCityFn() {
    multiSelectObj.destroy();
  }
  function writeShipCityFn(args) {
    {
      let multiSelectVal = args.rowData[args.column.field]
        ? args.rowData[args.column.field].split(",")
        : [];
      multiSelectObj = new MultiSelect({
        value: multiSelectVal,
        dataSource: multiselectDatasource,
        fields: { value: "ShipCity", text: "ShipCity" },
        floatLabelType: "Never",
        mode: "Box",
      });
      multiSelectObj.appendTo(ddElem);
    }
  }
  export default {
    data() {
      return {
        data: purchaseData,
        pageSettings: { pageSizes: true, pageSize: 5 },
        editSettings: { allowEditing: true, allowAdding: true, allowDeleting:
true },
        dsParams: {
          create: createShipCityFn,
          destroy: destroyShipCityFn,
          read: readShipCityFn,
          write: writeShipCityFn,
        },
        toolbar: ['Add', 'Edit', 'Delete', 'Update', 'Cancel'],
        orderIDRules: { required: true },
      };
    },
    provide: {
      grid: [Page, Edit, Toolbar],
    },
  };
</script>
<style>
@import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/grid/edit/default-cs15" %}

#### Render MaskedTextBox component while editing

Use the cell edit template feature of the Grid to render the MaskedTextBox component in the Grid edit form. In the following sample, the MaskedTextBox component is rendered in the Mask column.

#### APP.VUE

```

<template>
  <div id="app">
    <ejs-grid ref='grid' :dataSource='data' :allowPaging='true'
:editSettings='editSettings' :toolbar='toolbar'>
      <e-columns>
        <e-column field='OrderID' headerText='Order ID'
type='number' textAlign='Right' :isPrimaryKey='true' width=100></e-column>
        <e-column field='CustomerID' headerText='Customer ID'
type='string' width=120>
          </e-column>
        <e-column field='ShipCountry' headerText='Ship Country'
type='string' width=120>
          </e-column>
        <e-column field='Mask' headerText='Mask' :edit='params'
width=140>
          </e-column>
        </e-columns>
      </ejs-grid>
    </div>
  </template>
  <script>
import Vue from "vue";
import { GridPlugin, Page, Toolbar, Edit } from "@syncfusion/ej2-vue-grids";
import { MaskedTextBox } from '@syncfusion/ej2-inputs';
import { employeeData } from './datasource.js';
Vue.use(GridPlugin);
let element;
let maskObj;
function create() {
  element = document.createElement('input');
  return element;
}
function destroy() {
  maskObj.destroy();
}
function read() {
  return maskObj.value;
}
function write(args) {
  maskObj = new MaskedTextBox({
    mask: "0-0-0-0",
    value: args.rowData.Mask
  });
  maskObj.appendTo(element);
}
export default {
  data() {
    return {
      data: employeeData,
      editSettings: { allowEditing: true, allowAdding: true, allowDeleting:
true },
      params: {
        create: create,
        destroy: destroy,
        read: read,
        write: write,
      },
    },
  },

```

```

        toolbar: ['Add', 'Edit', 'Delete', 'Update', 'Cancel']
    };
},
provide: {
    grid: [Page, Edit, Toolbar],
},
};
</script>
<style>
@import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/grid/edit/default-cs16" %}

#### Render RichTextEditor component while editing

Use the cell edit template feature of the Grid to render the RichTextEditor component in the Grid edit form. In the below sample, we have rendered RichTextEditor component in the **ShipAddress** column, so we use [allowTextWrap](#) property to true.

#### APP.VUE

```

<template>
  <div id="app">
    <ejs-grid ref='grid' :dataSource='data' :allowPaging='true'
    :allowTextWrap='true' :editSettings='editSettings'
    :pageSettings='pageSettings' :toolbar='toolbar' height='175px'
    :created='created'>
      <e-columns>
        <e-column field='OrderID' headerText='Order ID'
        type='number' textAlign='Right' :isPrimaryKey='true'
        :validationRules='orderIDRules' width=100></e-column>
        <e-column field='CustomerID' headerText='Customer ID'
        type='string' width=140>
        </e-column>
        <e-column field='Freight' headerText='Freight' type='number'
        textAlign='Right' editType='numericedit' format='C2' width=120></e-column>
        <e-column field='ShipAddress' headerText='Ship Address'
        type='string' :edit='ddParams' :valueAccessor='valueAccessor'
        :disableHtmlEncode='disableHtmlEncode' width=180>
        </e-column>
      </e-columns>
    </ejs-grid>
  </div>
</template>
<script>
import Vue from "vue";
import { purchaseData } from './datasource.js';
import { GridPlugin, Page, Toolbar, Edit } from "@syncfusion/ej2-vue-grids";
import { RichTextEditor, Toolbar as RTEToolbar, Link, Image, HtmlEditor,
QuickToolbar } from "@syncfusion/ej2-richtexteditor";
RichTextEditor.Inject(RTEToolbar, Link, Image, HtmlEditor, QuickToolbar);
Vue.use(GridPlugin);
let rteElem;
let richTextEditor;
function createShipAddressFn() {
  rteElem = document.createElement("textarea");

```



```

    return rteElem;
  }
  function destroyShipAddressFn() {
    richTextEditor.destroy();
  }
  function readShipAddressFn() {
    return richTextEditor.value;
  }
  function writeShipAddressFn(args) {
    richTextEditor = new RichTextEditor({
      value: args.rowData[args.column.field],
    });
    richTextEditor.appendTo(rteElem);
  }
  export default {
    data() {
      return {
        data: purchaseData,
        pageSettings: { pageSizes: true, pageSize: 5 },
        disableHtmlEncode: false,
        editSettings: { allowEditing: true, allowAdding: true, allowDeleting:
true },
        ddParams: {
          create: createShipAddressFn,
          destroy: destroyShipAddressFn,
          read: readShipAddressFn,
          write: writeShipAddressFn,
        },
        toolbar: ['Add', 'Edit', 'Delete', 'Update', 'Cancel'],
        orderIDRules: { required: true },
      };
    },
    methods: {
      created(args) {
        this.$refs.grid.ej2Instances.keyConfigs.enter = "";
      },
      valueAccessor: function (field, cdata, column) {
        var value = cdata[field];
        if (value !== undefined) {
          return value.split("\n").join("<br>");
        } else {
          return "";
        }
      },
    },
    provide: {
      grid: [Page, Edit, Toolbar],
    },
  };
</script>
<style>
@import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/grid/edit/default-cs17" %}

### Render multiple columns in DropDownList component while editing

Use the cell edit template feature of the Grid to render the DropDownList component in the Grid edit form.

The DropDownList has been provided with several options to customize each list item, group title, selected value, header, and footer element. By default, list items can be rendered as a single column in the DropDownList component. Instead of this, multiple columns can be rendered. This can be achieved by using the [headerTemplate](#) and [itemTemplate](#) properties of the DropDownList component.

This is demonstrated in the following sample.

#### APP.VUE

```
<template>
  <div id="app">
    <ejs-grid ref='grid' :dataSource='data' :allowPaging='true'
      :editSettings='editSettings' :toolbar='toolbar'>
      <e-columns>
        <e-column field='OrderID' headerText='Order ID'
          type='number' textAlign='Right' :isPrimaryKey='true' width=100></e-column>
        <e-column field='CustomerID' headerText='Customer ID'
          type='string' width=120></e-column>
        <e-column field='ShipCountry' headerText='Ship Country'
          type='string' :edit='params' width=300></e-column>
      </e-columns>
    </ejs-grid>
  </div>
</template>
<script>
import Vue from "vue";
import { GridPlugin, Page, Toolbar, Edit } from "@syncfusion/ej2-vue-grids";
import { Query } from "@syncfusion/ej2-data";
import { DropDownList } from "@syncfusion/ej2-dropdowns";
import { data } from './datasource.js';
Vue.use(GridPlugin);
let element;
let dropdownobj;
function create() {
  element = document.createElement('input');
  return element;
}
function destroy() {
  dropdownobj.destroy();
}
function read() {
  return dropdownobj.value;
}
function write(args) {
  dropdownobj = new DropDownList({
    dataSource:data,
    value: args.rowData[args.column.field],
    query: new Query().select(['EmployeeID', 'ShipCountry']).take(10),
    fields: { text: 'ShipCountry', value: 'ShipCountry' },
    placeholder: 'Select a Country',
  });
}
```

```

        headerTemplate:
'<table><tr><th>EmployeeID</th><th>ShipCountry</th></tr></table>',
        itemTemplate: '<div class="e-grid"><table class="e-
table"><tbody><tr><td class="e-rowcell">${EmployeeID}</td><td class="e-
rowcell">${ShipCountry}</td></tr> </tbody></table></div>'
    });
    dropdownobj.appendTo(element);
}
export default {
  data() {
    return {
      data: data,
      editSettings: { allowEditing: true, allowAdding: true, allowDeleting:
true },
      params: {
        create: create,
        destroy: destroy,
        read: read,
        write: write,
      },
      toolbar: ['Add', 'Edit', 'Delete', 'Update', 'Cancel']
    };
  },
  provide: {
    grid: [Page, Edit, Toolbar],
  },
};
</script>
<style>
@import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
.content {
  margin: 0 auto;
  width: 550px;
}
table{
  width:100%;
  border-collapse: separate;
  table-layout: fixed;
}
th,td{
  border-width: 1px 0 0 1px;
  border-color: #e0e0e0;
  text-align: left;
  border-style: solid;
  display: table-cell;
}
th{
  line-height: 36px;
  text-indent: 16px;
}
.e-ddl-header{
  padding-right: 17px;
  border-width: 1px 0px 1px 0px;
  border-color: #e0e0e0;
  border-style: solid;
}
.e-dropdownbase .e-list-item{

```

```
padding-right:0px;
}
</style>
```

{% previewsample "page.domainurl/code-snippet/grid/edit/default-cs18" %}

### Using template

The cell editor for a particular column can be specified using a Vue Component. The [column.editTemplate](#) property used to define the corresponding column editor.

### APP.VUE

```
<template>
  <div id="app">
    <ejs-grid :dataSource='data' :editSettings='editSettings'
:toolbar='toolbar' height='273px'>
      <e-columns>
        <e-column field='OrderID' headerText='Order ID'
textAlign='Right' :isPrimaryKey='true' width=100></e-column>
        <e-column field='CustomerID' headerText='Customer ID'
width=120></e-column>
        <e-column field='Freight' headerText='Freight' textAlign=
'Right' editType= 'numericedit' width=120 format= 'C2'></e-column>
        <e-column field='OrderDate' headerText='Order Date'
type='date' format= 'yMd' width=150 :editTemplate='editTemplate'></e-column>
      </e-columns>
    </ejs-grid>
  </div>
</template>
<script>
import Vue from "vue";
import { GridPlugin, Page, Toolbar, Edit } from "@syncfusion/ej2-vue-grids";
import { data } from './datasource.js';
import { DatePickerPlugin } from "@syncfusion/ej2-vue-calendars";
Vue.use(GridPlugin);
Vue.use(DatePickerPlugin);
export default {
  data() {
    return {
      data: data,
      editSettings: { allowEditing: true, allowAdding: true, allowDeleting:
true, mode: 'Normal' },
      toolbar: ['Add', 'Edit', 'Delete', 'Update', 'Cancel']
    };
  },
  methods: {
    editTemplate: function() {
      return {template: Vue.component('datePicker', {
        template: `<ejs-datepicker id="OrderDate" placeholder="Order
Date" v-model="data.OrderDate" floatLabelType='Never'></ejs-datepicker>`,
        data() {
          return {data:{}}
        }
      })}
    }
  },
},
```

```

    provide: {
      grid: [Page, Edit, Toolbar]
    }
  }
</script>
<style>
  @import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/grid/edit/default-cs19" %}

### In line editing in Vue Grid component

In Normal edit mode, when you start editing the currently selected record is changed to edit state. You can change the cell values and save edited data to the data source. To enable Normal edit, set the [editSettings.mode](#) as **Normal**.

### APP.VUE

```

<template>
  <div id="app">
    <ejs-grid :dataSource='data' :editSettings='editSettings'
:toolbar='toolbar' height='273px'>
      <e-columns>
        <e-column field='OrderID' headerText='Order ID'
textAlign='Right' :isPrimaryKey='true' width=100></e-column>
        <e-column field='CustomerID' headerText='Customer ID'
width=120></e-column>
        <e-column field='Freight' headerText='Freight' textAlign=
'Right' editType= 'numericedit' width=120 format= 'C2'></e-column>
        <e-column field='ShipCountry' headerText='Ship Country'
editType= 'dropdownedit' width=150></e-column>
      </e-columns>
    </ejs-grid>
  </div>
</template>
<script>
import Vue from "vue";
import { GridPlugin, Page, Toolbar, Edit } from "@syncfusion/ej2-vue-grids";
import { data } from './datasource.js';
Vue.use(GridPlugin);
export default {
  data() {
    return {
      data: data,
      editSettings: { allowEditing: true, allowAdding: true, allowDeleting:
true, mode: 'Normal' },
      toolbar: ['Add', 'Edit', 'Delete', 'Update', 'Cancel']
    };
  },
  provide: {
    grid: [Page, Edit, Toolbar]
  }
}
</script>
<style>
  @import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";

```

```
</style>
```

```
{% previewsample "page.domainurl/code-snippet/grid/edit/default-cs26" %}
```

**Normal** edit mode is default mode of editing.

*Automatically update the column based on another column edited value*

You can update the column value based on another column edited value by using the Cell Edit Template feature.

In the below demo, we have update the **TotalCost** column value based on the **UnitPrice** and **UnitInStock** column value while editing.

#### APP.VUE

```
<template>
  <div id="app">
    <ejs-grid id="grid" :dataSource="data" :editSettings="editSettings"
    :toolbar="toolbar" height="273px">
      <e-columns>
        <e-column field="ProductID" headerText="Product ID"
        textAlign="Right" :isPrimaryKey="true" width="100"></e-column>
        <e-column field="ProductName" headerText="Product Name"
        width="120"></e-column>
        <e-column field="UnitPrice" headerText="Unit Price"
        editType="numericedit" :edit="priceParams"
        width="150" format="C2" textAlign="Right" ></e-column>
        <e-column field="UnitsInStock" headerText="Units In Stock"
        editType="numericedit"
        :edit="stockParams" width="150" textAlign="Right"></e-column>
        <e-column field="TotalCost" headerText="Total Cost" width="150"
        :allowEditing="false" format="C2" textAlign="Right" ></e-column>
      </e-columns>
    </ejs-grid>
  </div>
</template>
<script>
import Vue from "vue";
import { GridPlugin, Toolbar, Edit } from "@syncfusion/ej2-vue-grids";
import { NumericTextBox } from "@syncfusion/ej2-inputs";
import { productData } from "../datasource.js";
let priceElem, stockElem, priceObj, stockObj;
Vue.use(GridPlugin);
export default {
  data: () => {
    return {
      data: productData,
      toolbar: ["Add", "Edit", "Delete", "Update", "Cancel"],
      editSettings: {
        allowEditing: true,
        allowAdding: true,
        allowDeleting: true,
      },
      priceParams: {
        create: () => {
          priceElem = document.createElement("input");
```

```

        return priceElem;
    },
    read: () => {
        return priceObj.value;
    },
    destroy: () => {
        priceObj.destroy();
    },
    write: (args) => {
        priceObj = new NumericTextBox({
            value: args.rowData[args.column.field],
            change: function (args) {
                let formEle = document
                    .getElementById("grid")
                    .querySelector("form").ej2_instances[0];
                var totalCostFieldEle = formEle.getInputElement("TotalCost");
                totalCostFieldEle.value = priceObj.value * stockObj.value;
            },
        });
        priceObj.appendTo(priceElem);
    },
},
stockParams: {
    create: () => {
        stockElem = document.createElement("input");
        return stockElem;
    },
    read: () => {
        return stockObj.value;
    },
    destroy: () => {
        stockObj.destroy();
    },
    write: (args) => {
        stockObj = new NumericTextBox({
            value: args.rowData[args.column.field],
            change: function (args) {
                let formEle = document
                    .getElementById("grid")
                    .querySelector("form").ej2_instances[0];
                var totalCostFieldEle = formEle.getInputElement("TotalCost");
                totalCostFieldEle.value = priceObj.value * stockObj.value;
            },
        });
        stockObj.appendTo(stockElem);
    },
},
};
},
provide: {
    grid: [Edit, Toolbar],
},
};
</script>
<style>
    @import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
</style>

```

```
{% previewsample "page.domainurl/code-snippet/grid/edit/default-cs27" %}
```

### Cancel edit based on condition

You can prevent the CRUD operations of the Grid by using condition in the [actionBegin](#) event with requestType as `beginEdit` for editing, `add` for adding and `delete` for deleting actions.

In the below demo, we prevent the CRUD operation based on the `Role` column value. If the Role Column is `Employee`, we are unable to edit/delete that row.

### APP.VUE

```
<template>
  <div id="app">
    <button v-on:click="btnClick">Grid is Addable</button>
    <ejs-grid :dataSource='data' :editSettings='editSettings'
:toolbar='toolbar' :actionBegin="actionBegin" height='240px'>
      <e-columns>
        <e-column field='OrderID' headerText='Order ID'
textAlign='Right' :isPrimaryKey='true' width=100></e-column>
        <e-column field='Role' headerText='Role' width=120></e-
column>
        <e-column field='Freight' headerText='Freight' textAlign=
'Right' editType= 'numericedit' width=120 format= 'C2'></e-column>
        <e-column field='ShipCountry' headerText='Ship Country'
editType= 'dropdownedit' width=150></e-column>
      </e-columns>
    </ejs-grid>
  </div>
</template>
<script>
import Vue from "vue";
import { GridPlugin, Page, Toolbar, Edit } from "@syncfusion/ej2-vue-grids";
import { employeeData } from './datasource.js';
Vue.use(GridPlugin);
export default {
  data() {
    return {
      data: employeeData,
      editSettings: { allowEditing: true, allowAdding: true, allowDeleting:
true, mode: 'Normal' },
      toolbar: ['Add', 'Edit', 'Delete', 'Update', 'Cancel'],
      isAddable: true
    };
  },
  methods: {
    actionBegin: function (args) {
      if (args.requestType === "beginEdit") {
        if (args.rowData["Role"].toLowerCase() === "employee") {
          args.cancel = true;
        }
      }
      if (args.requestType === "delete") {
        if (args.data[0]["Role"].toLowerCase() === "employee") {
          args.cancel = true;
        }
      }
    }
  }
}
```



```

    }
    if (args.requestType === "add") {
      if (!this.isAddable) {
        args.cancel = true;
      }
    }
  },
  btnClick: function (args) {
    args.target.innerText === "Grid is Addable" ? (args.target.innerText =
"Grid is Not Addable") : (args.target.innerText = "Grid is Addable");
    this.isAddable = !this.isAddable;
  },
},
provide: {
  grid: [Page, Edit, Toolbar]
}
}
</script>
<style>
  @import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/grid/edit/default-cs28" %}

#### *Perform CRUD action programmatically*

Grid methods can be used to perform CRUD operations programmatically. The [addRecord](#), [deleteRecord](#), and [startEdit](#) methods are used to perform CRUD operations in the following demo.

- To add a new record to the Grid, use the [addRecord](#) method. In this method, you can pass the data parameter to add a new record to the Grid, and the index parameter to add a record at a specific index. If you call this method with no parameters, it will create an empty row in the Grid.
- To change the selected row to the edit state, use the [startEdit](#) method.
- If you need to update the row data in the Grid's datasource, you can use the [updateRow](#) method. In this method, you need to pass the index value of the row to be updated along with the updated data.
- If you need to update the particular cell in the row, you can use the [setCellValue](#) method. In this method, you need to pass the primary key value of the data source, field name, and new value for the particular cell.
- To remove a selected row from the Grid, use the [deleteRecord](#) method. For both edit and delete operations, you must select a row first.

Note: In both normal and dialog editing modes, these methods can be used.

#### **APP.VUE**

```

<template>
  <div id="app">
    <button id="edit" @click="clickEdit">Edit</button>
    <button id="add" @click="clickAdd">Add</button>
    <button id="delete" @click="clickDelete">Delete</button>
    <button id="updaterow" @click="clickUpdateRow">Update Row</button>
  </div>
</template>

```

```

        <button id="updatecell" @click="clickUpdateCell">Update
        Cell</button>
        <br /><br />
        <ejs-grid ref="grid" :dataSource='data'
        :editSettings='editSettings' height='230px' id="Grid">
            <e-columns>
                <e-column field='OrderID' headerText='Order ID'
                textAlign='Right' width=120 :isPrimaryKey='true'></e-column>
                <e-column field='CustomerID' headerText='Customer ID'
                width=150></e-column>
                <e-column field='ShipCity' headerText='Ship City'
                width=150></e-column>
                <e-column field='ShipName' headerText='Ship Name'
                width=150></e-column>
            </e-columns>
        </ejs-grid>
    </div>
</template>
<script>
import Vue from "vue";
import { GridPlugin, Edit } from "@syncfusion/ej2-vue-grids";
import { data } from './datasource.js';
Vue.use(GridPlugin);
export default {
    data() {
        return {
            data: data,
            editSettings: { allowEditing: true, allowAdding: true, allowDeleting:
true },
        };
    },
    methods: {
        clickAdd: function () {
            this.$refs.grid.ej2Instances.addRecord({ "OrderID": "10248",
"CustomerID": "RTER", "ShipCity": "America", "ShipName": "Hanari" });
        },
        clickEdit: function () {
            this.$refs.grid.ej2Instances.startEdit();
        },
        clickDelete: function () {
            this.$refs.grid.ej2Instances.deleteRecord();
        },
        clickUpdateRow: function () {
            this.$refs.grid.ej2Instances.updateRow(0, { OrderID: 10248,
CustomerID: 'RTER', ShipCity: 'America', ShipName: 'Hanari' });
        },
        clickUpdateCell: function () {
            this.$refs.grid.ej2Instances.setCellValue((this.$refs.grid.ej2Instances.curr
entViewData[0] as any).OrderID, 'CustomerID', 'Value Changed');
        }
    },
    provide: {
        grid: [Edit]
    }
}
</script>

```

```
<style>
  @import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
</style>
```

{% previewsample "page.domainurl/code-snippet/grid/edit/default-cs29" %}

### Confirmation dialog

The delete confirm dialog can be shown when deleting a record by defining the

[showDeleteConfirmDialog](#) as **true**

### APP.VUE

```
<template>
  <div id="app">
    <ejs-grid :dataSource='data' :editSettings='editSettings'
:toolbar='toolbar' height='273px'>
      <e-columns>
        <e-column field='OrderID' headerText='Order ID'
textAlign='Right' :isPrimaryKey='true' width=100></e-column>
        <e-column field='CustomerID' headerText='Customer ID'
width=120></e-column>
        <e-column field='Freight' headerText='Freight' textAlign=
'Right' editType= 'numericedit' width=120 format= 'C2'></e-column>
        <e-column field='ShipCountry' headerText='Ship Country'
editType= 'dropdownedit' width=150></e-column>
      </e-columns>
    </ejs-grid>
  </div>
</template>
<script>
import Vue from "vue";
import { GridPlugin, Page, Toolbar, Edit } from "@syncfusion/ej2-vue-grids";
import { data } from './datasource.js';
Vue.use(GridPlugin);
export default {
  data() {
    return {
      data: data,
      editSettings: { showDeleteConfirmDialog: true, allowEditing: true,
allowAdding: true, allowDeleting: true, mode: 'Normal' },
      toolbar: ['Add', 'Edit', 'Delete', 'Update', 'Cancel']
    };
  },
  provide: {
    grid: [Page, Edit, Toolbar]
  }
}
</script>
<style>
  @import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
</style>
```

{% previewsample "page.domainurl/code-snippet/grid/edit/default-cs30" %}

The **showDeleteConfirmDialog** supports all type of edit modes.

### Default column values on add new row

The grid provides an option to set the default value for the columns when adding a new record in it. To set a default value for the particular column by defining the [columns.defaultValue](#).

#### APP.VUE

```
<template>
  <div id="app">
    <ejs-grid :dataSource='data' :editSettings='editSettings'
:toolbar='toolbar' height='273px'>
      <e-columns>
        <e-column field='OrderID' headerText='Order ID'
textAlign='Right' :isPrimaryKey='true' :validationRules='orderIDRules'
width=100></e-column>
        <e-column field='CustomerID' headerText='Customer ID'
defaultValue= 'HANAR' :validationRules='customerIDRules' width=120></e-
column>
        <e-column field='Freight' headerText='Freight' textAlign=
'Right' editType= 'numericedit' width=120 format= 'C2'></e-column>
        <e-column field='ShipCountry' headerText='Ship Country'
editType= 'dropdownedit' width=150></e-column>
      </e-columns>
    </ejs-grid>
  </div>
</template>
<script>
import Vue from "vue";
import { GridPlugin, Page, Toolbar, Edit } from "@syncfusion/ej2-vue-grids";
import { data } from './datasource.js';
Vue.use(GridPlugin);
export default {
  data() {
    return {
      data: data,
      editSettings: { allowEditing: true, allowAdding: true, allowDeleting:
true, mode: 'Normal' },
      toolbar: ['Add', 'Edit', 'Delete', 'Update', 'Cancel'],
      orderIDRules: { required: true },
      customerIDRules: { required: true, minLength: 3 }
    };
  },
  provide: {
    grid: [Page, Edit, Toolbar]
  }
}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
</style>
```

{% previewsample "page.domainurl/code-snippet/grid/edit/default-cs31" %}

### Adding a new row at the bottom of the Grid

By default, a new row will be added at the top of the grid. You can change it by setting [editSettings.newRowPosition](#) as **Bottom**.

**APP.VUE**

```

<template>
  <div id="app">
    <ejs-grid :dataSource='data' :editSettings='editSettings'
height='270' :toolbar='toolbar'>
      <e-columns>
        <e-column field='OrderID' headerText='Order ID'
textAlign='Right' :isPrimaryKey='true' width=100></e-column>
        <e-column field='CustomerID' headerText='Customer ID'
width=120></e-column>
        <e-column field='Freight' headerText='Freight' textAlign=
'Right' width=120 format= 'C2'></e-column>
        <e-column field='ShipCountry' headerText='Ship Country'
width=150></e-column>
      </e-columns>
    </ejs-grid>
  </div>
</template>
<script>
import Vue from "vue";
import { GridPlugin, Page, Toolbar, Edit } from "@syncfusion/ej2-vue-grids";
import { data } from './datasource.js';
Vue.use(GridPlugin);
export default {
  data() {
    return {
      data: data,
      editSettings: { allowEditing: true, allowAdding: true, allowDeleting:
true, newRowPosition: 'Bottom' },
      toolbar: ['Add', 'Edit', 'Delete', 'Update', 'Cancel']
    };
  },
  provide: {
    grid: [Page, Edit, Toolbar]
  }
}
</script>
<style>
  @import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/grid/edit/default-cs32" %}

Add newRowPosition is supported for **Normal** and **Batch** editing modes.

*Move the focus to a particular cell instead of first cell while editing a row*

The [recordDoubleClick](#) event allows you to move the focus to the corresponding cell (the cell that you double-clicked to edit a row) instead of the first cell in edit form. With the help of this event, you can focus the double-clicked column in inline edit mode.

**APP.VUE**

```

<template>
  <div id="app">

```

```

<ejs-grid ref="grid" :dataSource="data" :editSettings="editSettings"
height="220" :actionComplete="actionComplete"
:recordDoubleClick="recordDoubleClick">
  <e-columns>
    <e-column field="OrderID" headerText="Order ID"
textAlign="Right" :isPrimaryKey="true" width="120" type="number">
    </e-column>
    <e-column field="CustomerID" headerText="Customer ID"
width="140" type="string">
    </e-column>
    <e-column field="Freight" headerText="Freight"
editType="numERICedit" textAlign="Right" width="120" format="C2">
    </e-column>
    <e-column field="OrderDate" headerText="Order Date"
textAlign="Right" width="140" editType="datetimepickeredit" format="yMMM">
    </e-column>
    <e-column field="ShipCountry" headerText="Ship Country"
width="150" editType="dropdownedit" :edit="params">
    </e-column>
  </e-columns>
</ejs-grid>
</div>
</template>
<script>
import Vue from "vue";
import { GridPlugin, Page, Edit } from "@syncfusion/ej2-vue-grids";
import { data } from "../datasource.js";
Vue.use(GridPlugin);
var fieldName;
export default {
  data() {
    return {
      data: data,
      editSettings: {
        allowEditing: true,
        allowAdding: true,
        allowDeleting: true,
        mode: "Normal",
      },
      shipFormat: { type: "dateTime", format: "dd/MM/yyyy hh:mm a" },
      params: {
        params: {
          popupHeight: "300px",
        }
      }
    };
  },
  methods: {
    actionComplete(e) {
      if (e.requestType === "beginEdit") {
        // focus the column
        e.form.elements[this.$refs.grid.ej2Instances.element.getAttribute("id") +
fieldName].focus();
      }
    },
    recordDoubleClick(e) {

```

```

        var clickedColumnIndex = e.cell.getAttribute("data-colindex");
        fieldName =
this.$refs.grid.ej2Instances.columnModel[parseInt(clickedColumnIndex,
10)].field;
    }
    },
    provide: {
        grid: [Page, Edit]
    }
};
</script>
<style>
@import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/grid/edit/default-cs33" %}

### Dialog editing in Vue Grid component

In Dialog edit mode, when you start editing the currently selected row data will be shown on a dialog. You can change the cell values and save edited data to the data source. To enable Dialog edit, set the [editSettings.mode](#) as **Dialog**.

### APP.VUE

```

<template>
  <div id="app">
    <ejs-grid :dataSource='data' :editSettings='editSettings'
:toolbar='toolbar' height='273px'>
      <e-columns>
        <e-column field='OrderID' headerText='Order ID'
textAlign='Right' :isPrimaryKey='true' width=100></e-column>
        <e-column field='CustomerID' headerText='Customer ID'
width=120></e-column>
        <e-column field='Freight' headerText='Freight' textAlign=
'Right' editType= 'numericedit' width=120 format= 'C2'></e-column>
        <e-column field='ShipCountry' headerText='Ship Country'
editType= 'dropdownedit' width=150></e-column>
      </e-columns>
    </ejs-grid>
  </div>
</template>
<script>
import Vue from "vue";
import { GridPlugin, Page, Toolbar, Edit } from "@syncfusion/ej2-vue-grids";
import { data } from './datasource.js';
Vue.use(GridPlugin);
export default {
  data() {
    return {
      data: data,
      editSettings: { allowEditing: true, allowAdding: true, allowDeleting:
true, mode: 'Dialog' },
      toolbar: ['Add', 'Edit', 'Delete', 'Update', 'Cancel']
    };
  },
  provide: {

```

```

    grid: [Page, Edit, Toolbar]
  }
}
</script>
<style>
  @import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/grid/edit/default-cs8" %}

### Customize edit dialog

You can customize the appearance of the edit dialog in the [actionComplete](#) event based on `requestType` as `beginEdit` or `add`.

In the following example, the dialog's properties like header text, showCloseIcon, height have been changed while editing and adding the records.

Also the locale text for the `Save` and `Cancel` buttons has been changed by overriding the default locale strings.

You can refer the Grid [Default text](#) list for more localization.

### APP.VUE

```

<template>
  <div id="app">
    <ejs-grid :dataSource='data' :editSettings='editSettings'
    :toolbar='toolbar' height='273px' :actionComplete = 'actionComplete'>
      <e-columns>
        <e-column field='OrderID' headerText='Order ID'
        textAlign='Right' :isPrimaryKey='true' width=100></e-column>
        <e-column field='CustomerID' headerText='Customer ID'
        width=120></e-column>
        <e-column field='Freight' headerText='Freight' textAlign=
        'Right' editType= 'numericedit' width=120 format= 'C2'></e-column>
        <e-column field='ShipCountry' headerText='Ship Country'
        editType= 'dropdownedit' width=150></e-column>
      </e-columns>
    </ejs-grid>
  </div>
</template>
<script>
import Vue from "vue";
import { L10n } from '@syncfusion/ej2-base';
import { GridPlugin, Page, Toolbar, Edit } from "@syncfusion/ej2-vue-grids";
import { data } from './datasource.js';
Vue.use(GridPlugin);
L10n.load({
  'en-US': {
    'grid': {
      'SaveButton': 'Submit',
      'CancelButton': 'Discard'
    }
  }
});
export default {
  data() {

```



```

    return {
      data: data,
      editSettings: { allowEditing: true, allowAdding: true, allowDeleting:
true, mode: 'Dialog' },
      toolbar: ['Add', 'Edit', 'Delete', 'Update', 'Cancel']
    };
  },
  methods: {
    actionComplete(args) {
      if ((args.requestType === 'beginEdit' || args.requestType ===
'add')) {
        let dialog = args.dialog;
        dialog.showCloseIcon = false;
        dialog.height = 400;
        // change the header of the dialog
        dialog.header = args.requestType === 'beginEdit' ? 'Edit Record
of ' + args.rowData['CustomerID'] : 'New Customer';
      }
    }
  },
  provide: {
    grid: [Page, Edit, Toolbar]
  }
}
</script>
<style>
  @import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/grid/edit/default-cs9" %}

The Grid add or edit dialog element has the max-height property, which is calculated based on the available window height. So, in the normal window (1920 x 1080), it is possible to set the dialog's height up to 658px.

#### Show or hide columns in dialog editing

The Grid has the option to show hidden columns or hide visible columns while editing in the dialog edit mode by using the [actionBegin](#) event of the Grid.

In the `actionBegin` event, when the `requestType` is `beginEdit` or `add`, the column will be shown or hidden using the `column.visible` property. When the `requestType` is `save`, the properties will be reset to their original state.

In the following example, the `CustomerID` column is rendered as a hidden column, and the `ShipCountry` column is rendered as a visible column. In the edit mode, the `CustomerID` column will be changed to a visible state and the `ShipCountry` column will be changed to a hidden state.

#### APP.VUE

```

<template>
  <div id="app">
    <ejs-grid ref='grid' :dataSource='data' :editSettings='editSettings'
:toolbar='toolbar' height='273px' :actionBegin = 'actionBegin'>
      <e-columns>
        <e-column field='OrderID' headerText='Order ID'
textAlign='Right' :isPrimaryKey='true' width=100></e-column>

```

```

        <e-column field='CustomerID' :visible = 'false'
headerText='Customer ID' width=120></e-column>
        <e-column field='Freight' headerText='Freight' textAlign=
'Right' editType= 'numericedit' width=120 format= 'C2'></e-column>
        <e-column field='ShipCountry' headerText='Ship Country'
editType= 'dropdownedit' width=150></e-column>
    </e-columns>
</ejs-grid>
</div>
</template>
<script>
import Vue from "vue";
import { GridPlugin, Page, Toolbar, Edit } from "@syncfusion/ej2-vue-grids";
import { data } from './datasource.js';
Vue.use(GridPlugin);
export default {
    data() {
        return {
            data: data,
            editSettings: { allowEditing: true, allowAdding: true, allowDeleting:
true, mode: 'Dialog' },
            toolbar: ['Add', 'Edit', 'Delete', 'Update', 'Cancel']
        };
    },
    methods: {
        actionBegin(args) {
            if ((args.requestType === 'beginEdit' || args.requestType ===
'add')) {
                for (var i = 0; i < this.$refs.grid.getColumns().length; i++) {
                    if (this.$refs.grid.getColumns()[i].field == "CustomerID") {
                        this.$refs.grid.getColumns()[i].visible = true;
                    }
                    else if (this.$refs.grid.getColumns()[i].field ==
"ShipCountry") {
                        this.$refs.grid.getColumns()[i].visible = false;
                    }
                }
            }
            if ((args.requestType === 'save')) {
                for (var i = 0; i < this.$refs.grid.getColumns().length; i++) {
                    if (this.$refs.grid.getColumns()[i].field == "CustomerID") {
                        this.$refs.grid.getColumns()[i].visible = false;
                    }
                    else if (this.$refs.grid.getColumns()[i].field ==
"ShipCountry") {
                        this.$refs.grid.getColumns()[i].visible = true;
                    }
                }
            }
        },
        provide: {
            grid: [Page, Edit, Toolbar]
        }
    }
}
</script>
<style>

```

```
@import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
</style>
```

```
{% previewsample "page.domainurl/code-snippet/grid/edit/default-cs10" %}
```

### Customize Add/Edit Dialog footer

In dialog edit mode, a dialog will show up when editing the currently selected row or adding a new row. By default, you can save or cancel the edited changes by clicking the Save or Cancel button in the dialog's footer. Along with these buttons, it is possible to add a custom button in the footer section using the [actionComplete](#) event of the Grid.

In the following sample, using the `dialog` argument of the `actionComplete` event, the action for the custom button can be customized.

### APP.VUE

```
<template>
  <div id="app">
    <ejs-grid :dataSource='data' :editSettings='editSettings'
:toolbar='toolbar' height='273px' :actionComplete = 'actionComplete'>
      <e-columns>
        <e-column field='OrderID' headerText='Order ID'
textAlign='Right' :isPrimaryKey='true' width=100></e-column>
        <e-column field='CustomerID' headerText='Customer ID'
width=120></e-column>
        <e-column field='Freight' headerText='Freight' textAlign=
'Right' editType= 'numericedit' width=120 format= 'C2'></e-column>
        <e-column field='ShipCountry' headerText='Ship Country'
editType= 'dropdownedit' width=150></e-column>
      </e-columns>
    </ejs-grid>
  </div>
</template>
<script>
import Vue from "vue";
import { GridPlugin, Page, Toolbar, Edit } from "@syncfusion/ej2-vue-grids";
import { data } from './datasource.js';
Vue.use(GridPlugin);
export default {
  data() {
    return {
      data: data,
      editSettings: { allowEditing: true, allowAdding: true, allowDeleting:
true, mode: 'Dialog' },
      toolbar: ['Add', 'Edit', 'Delete', 'Update', 'Cancel']
    };
  },
  methods: {
    actionComplete(args) {
      if (args.requestType === 'beginEdit' || args.requestType === 'add')
      {
        let newFooterButton = {
          buttonModel: { content: 'custom' },
          click: this.onCustomButtonClick
        };
        args.dialog.buttons.push(newFooterButton);
      }
    }
  }
}
```

```

        args.dialog.refresh();
    },
    onCustomButtonClick() {
        alert('Add/Edit dialog custom footer button clicked');
    }
},
provide: {
    grid: [Page, Edit, Toolbar]
}
}
</script>
<style>
    @import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/grid/edit/default-cs11" %}

## Template editing in Vue Grid component

### *Inline or dialog template editing*

The dialog/inline template editing provides an option to customize the default behavior of dialog editing. Using the dialog template, you can render your own editors by defining the [editSettings.mode](#) as [Link to the Video](#) and [editSetting.template](#) as Vue component.

In some cases, you need to add the new field editors in the dialog which are not present in the column model. In that situation, the dialog template will help you to customize the default edit dialog.

To get start quickly with Dialog/Inline template edit Option, you can check on this video:

In the following sample, grid enabled with dialog template editing.

### **APP.VUE**

```

<template>
    <div id="app">
        <ejs-grid :dataSource='data' :editSettings='editSettings'
        :actionBegin="actionBegin" :actionComplete="actionComplete"
        :toolbar='toolbar' height='273px'>
            <e-columns>
                <e-column field='OrderID' headerText='Order ID'
                textAlign='Right' :isPrimaryKey='true' width=100></e-column>
                <e-column field='CustomerID' headerText='Customer ID'
                width=120></e-column>
                <e-column field='ShipCountry' headerText='Ship Country'
                width=150></e-column>
            </e-columns>
        </ejs-grid>
    </div>
</template>
<script>
import Vue from "vue";
import { GridPlugin, Page, Toolbar, Edit } from "@syncfusion/ej2-vue-grids";
import { data } from './datasource.js';
import DialogTemplate from './dialogtemp.vue';
import { DatePickerPlugin } from "@syncfusion/ej2-vue-calendars";
import { DropDownListPlugin } from "@syncfusion/ej2-vue-dropdowns";

```

```

import { NumericTextBox } from "@syncfusion/ej2-inputs";
import { NumericTextBoxPlugin } from "@syncfusion/ej2-vue-inputs";
import { DataUtil } from '@syncfusion/ej2-data';
Vue.use(GridPlugin);
Vue.use(DropDownListPlugin);
Vue.use(DatePickerPlugin);
Vue.use(NumericTextBoxPlugin)
export default {
  data() {
    return {
      data: data,
      editSettings: { allowEditing: true, allowAdding: true, allowDeleting:
true, mode: 'Dialog', template: function () {
        return { template : Vue.component('todo-item', {
          template: `<div formGroup="orderForm">
<div class="form-row">
  <div class="form-group col-md-6">
    <div class="e-float-input e-control-wrapper">
      <input id="OrderID" name="OrderID" v-
model='data.OrderID' type="text" :disabled="!data.isAdd">
      <span class="e-float-line"></span>
      <label class="e-float-text e-label-top" for="OrderID">
Order ID</label>
    </div>
  </div>
  <div class="form-group col-md-6">
    <div class="e-float-input e-control-wrapper">
      <input id="CustomerID" name="CustomerID" v-
model='data.CustomerID' type="text">
      <span class="e-float-line"></span>
      <label class="e-float-text e-label-top"
for="CustomerID">Customer Name</label>
    </div>
  </div>
</div>
<div class="form-row">
  <div class="form-group col-md-6">
    <ejs-numerictextbox id="Freight" placeholder="Freight" v-
model='data.Freight' floatLabelType='Always'></ejs-numerictextbox>
  </div>
  <div class="form-group col-md-6">
    <ejs-datepicker id="OrderDate" placeholder="Order Date" v-
model='data.OrderDate' floatLabelType='Always'></ejs-datepicker>
  </div>
</div>
<div class="form-row">
  <div class="form-group col-md-6">
    <ejs-dropdownlist id="ShipCountry" v-
model='data.ShipCountry' :dataSource='shipCountryDistinctData'
:fields="{text: 'ShipCountry', value: 'ShipCountry' }" placeholder="Ship
Country" popupHeight='300px' floatLabelType='Always'></ejs-dropdownlist>
  </div>
  <div class="form-group col-md-6">
    <ejs-dropdownlist id="ShipCity" v-model='data.ShipCity'
:dataSource='shipCityDistinctData' :fields="{text: 'ShipCity', value:
'ShipCity' }" placeholder="Ship City" popupHeight='300px'
floatLabelType='Always'></ejs-dropdownlist>

```

```

        </div>
    </div>
    <div class="form-row">
        <div class="form-group col-md-12">
            <div class="e-float-input e-control-wrapper">
                <textarea id="ShipAddress" name="ShipAddress"
type="text" v-model='data.ShipAddress'></textarea>
                <span class="e-float-line"></span>
                <label class="e-float-text e-label-top"
for="ShipAddress">Ship Address</label>
            </div>
        </div>
    </div>
</div>`,
data () {
    return {
        data: {}
    }
},
computed: {
    shipCityDistinctData: () => {
        return DataUtil.distinct(data, 'ShipCity', true);
    },
    shipCountryDistinctData: () => {
        return DataUtil.distinct(data, 'ShipCountry', true);
    }
}
}},
} },
    toolbar: ['Add', 'Edit', 'Delete', 'Update', 'Cancel']
};
},
provide: {
    grid: [Page, Edit, Toolbar]
},
methods: {
    actionBegin (args) {
        if (args.requestType === 'save') {
            // cast string to integer value.
            args.data['Freight'] =
parseFloat(args.form.querySelector("#Freight").value);
        }
    },
    actionComplete(args) {
        // Set initail Focus
        if (args.requestType === 'beginEdit') {
            (args.form.elements.namedItem('CustomerName')).focus();
        }
    }
}
}
</script>
<style>
    @import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
    @import
    "https://stackpath.bootstrapcdn.com/bootstrap/4.1.2/css/bootstrap.min.css";
    .form-group.col-md-6 {

```

```

        width: 250px;
        height: 54px;
    }
    .form-group.col-md-12 {
        height: 72px;
    }
    #ShipAddress {
        resize: vertical;
    }
</style>

```

{% previewsample "page.domainurl/code-snippet/grid/edit/dialogtemplate-cs1" %}

The Dialog/Inline template form editors should have **name** attribute.

#### Template context

The template should be a Vue Component. You can access the row information inside the Component and you can bind the attribute or value based on this row information.

The following properties will be available at the time of template execution.

Property Name	Usage
isAdd	A Boolean property; it defines whether the current row should be a new record or not.

In the following code example, the **OrderID** textbox has been disabled by using the **isAdd** property.

,

// The disabled attributes will be added based on the isAdd property.

```
<input id="OrderID" name="OrderID" v-model='data.OrderID' type="text" :disabled="!data.isAdd">
```

,

#### Get value from editor

You can read, format, and update the current editor value in the [actionBegin](#) event at the time of setting **requestType** to **save**.

In the following code example, the **Freight** value has been formatted and updated.

```
`ts
```

```
actionBegin(args) {
```

```
if (args.requestType === 'save') {
```

```
// cast string to integer value.
```

```
args.data['Freight'] = parseFloat(args.form.querySelector("#Freight").value);
```

```
}
```

```
},
```

```
,
```

*Set focus to editor*

By default, the first input element in the dialog will be focused while opening the dialog. If the first input element is in disabled or hidden state, focus the valid input element in the [actionComplete](#) event based on **requestType** as **beginEdit**.

```
`ts
actionComplete(args) {
  if ((args.requestType === 'beginEdit' || args.requestType === 'add')) {
    // Set initail Focus
    if (args.requestType === 'beginEdit') {
      (args.form.elements.namedItem('CustomerID')).focus();
    }
  }
}
```

*Adding validation rules for custom editors*

If you have used additional fields that are not present in the column model, then add the validation rules to the [actionComplete](#) event.

```
`ts
actionComplete(args) {
  if ((args.requestType === 'beginEdit' || args.requestType === 'add')) {
    // Add Validation Rules
    args.form.ej2_instances[0].addRules('Freight', {max: 500});
  }
}
```

*Batch editing in Vue Grid component*

In Batch edit mode, when you double-click on the grid cell, then the target cell changed to edit state. You can bulk save (added, changed and deleted data in the single request) to data source by click on the toolbar's **Update** button or by externally invoking the [batchSave](#) method. To enable Batch edit, set the [editSettings.mode](#) as **Batch**.

**APP.VUE**

```
<template>
  <div id="app">
    <ejs-grid :dataSource='data' :editSettings='editSettings'
    :toolbar='toolbar' height='273px'>
      <e-columns>
        <e-column field='OrderID' headerText='Order ID'
        textAlign='Right' :isPrimaryKey='true' width=100></e-column>
```



```

        <e-column field='CustomerID' headerText='Customer ID'
width=120 allowEditing='false'></e-column>
        <e-column field='Freight' headerText='Freight' textAlign=
'Right' editType= 'numericedit' width=120 format= 'C2'></e-column>
        <e-column field='ShipCountry' headerText='Ship Country'
editType= 'dropdownedit' width=150></e-column>
    </e-columns>
</ejs-grid>
</div>
</template>
<script>
import Vue from "vue";
import { GridPlugin, Page, Toolbar, Edit } from "@syncfusion/ej2-vue-grids";
import { data } from './datasource.js';
Vue.use(GridPlugin);
export default {
  data() {
    return {
      data: data,
      editSettings: { allowEditing: true, allowAdding: true, allowDeleting:
true, mode: 'Batch' },
      toolbar: ['Add', 'Edit', 'Delete', 'Update', 'Cancel']
    };
  },
  provide: {
    grid: [Page, Edit, Toolbar]
  }
}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/grid/edit/default-cs1" %}

If a column's [allowEditing](#) property is set to false, then the focus can be skipped in that non-editable column by clicking the tab or shift-tab key while in batch edit mode.

*Automatically update the column based on another column edited value in batch mode*

You can update the column value based on another column edited value in Batch mode by using the Cell Edit Template feature.

In the below demo, we have update the **TotalCost** column value based on the **UnitPrice** and **UnitInStock** column value while editing.

#### APP.VUE

```

<template>
  <div id="app">
    <ejs-grid id="Grid1" :dataSource="data" :editSettings="editSettings"
:toolbar="toolbar" height="273px" :cellEdit="cellEdit">
      <e-columns>
        <e-column field="ProductID" headerText="Product ID"
textAlign="Right" :isPrimaryKey="true" width="100"></e-column>
        <e-column field="ProductName" headerText="Product Name"
width="120"></e-column>

```

```

        <e-column field="UnitPrice" headerText="Unit Price"
editType="numericedit" :edit="priceParams"
        width="150" format="C2" textAlign="Right" ></e-column>
        <e-column field="UnitsInStock" headerText="Units In Stock"
editType="numericedit"
        :edit="stockParams" width="150" textAlign="Right"></e-column>
        <e-column field="TotalCost" headerText="Total Cost" width="150"
format="C2" textAlign="Right" ></e-column>
    </e-columns>
</ejs-grid>
</div>
</template>
<script>
import Vue from "vue";
import { GridPlugin, Toolbar, Edit } from "@syncfusion/ej2-vue-grids";
import { NumericTextBox } from "@syncfusion/ej2-inputs";
import { productData } from "../datasource.js";
import { getComponent } from "@syncfusion/ej2-base";
let priceElem, stockElem, priceObj, stockObj;
var grid;
Vue.use(GridPlugin);
export default {
  data: () => {
    return {
      data: productData,
      toolbar: ["Add", "Delete", "Update", "Cancel"],
      editSettings: {
        allowEditing: true,
        allowAdding: true,
        allowDeleting: true,
        mode: "Batch",
      },
    },
    priceParams: {
      create: () => {
        priceElem = document.createElement("input");
        return priceElem;
      },
      read: () => {
        return priceObj.value;
      },
      destroy: () => {
        priceObj.destroy();
      },
      write: (args) => {
        grid = new getComponent("Grid1", "grid");
        var rowData = args.rowData;
        var rowIndex = grid.getRowInfo(args.row).rowIndex;
        priceObj = new NumericTextBox({
          value: args.rowData[args.column.field],
          change: function (args) {
            var totalCostValue = args.value * rowData["UnitsInStock"];
            grid.updateCell(rowIndex, "TotalCost", totalCostValue);
          },
        });
        priceObj.appendTo(priceElem);
      },
    },
  },
};

```

```

stockParams: {
  create: () => {
    stockElem = document.createElement("input");
    return stockElem;
  },
  read: () => {
    return stockObj.value;
  },
  destroy: () => {
    stockObj.destroy();
  },
  write: (args) => {
    grid = new GetComponent("Grid1", "grid");
    var rowData = args.rowData;
    var rowIndex = grid.getRowInfo(args.row).rowIndex;
    stockObj = new NumericTextBox({
      value: args.rowData[args.column.field],
      change: function (args) {
        var totalCostValue = args.value * rowData["UnitPrice"];
        grid.updateCell(rowIndex, "TotalCost", totalCostValue);
      },
    });
    stockObj.appendTo(stockElem);
  },
},
};
},
methods: {
  cellEdit: function (args) {
    if (args.columnName === "TotalCost") {
      args.cancel = true;
    }
  },
},
provide: {
  grid: [Edit, Toolbar],
},
};
</script>
<style>
  @import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
</style>

```

```
{% previewsample "page.domainurl/code-snippet/grid/edit/default-cs2" %}
```

#### Cancel edit based on condition in batch mode

You can prevent the CRUD operations of the Batch edit Grid by using condition in the [cellEdit](#), [beforeBatchAdd](#) and [beforeBatchDelete](#) events for Edit, Add and Delete actions respectively.

In the below demo, we prevent the CRUD operation based on the **Role** column value. If the Role Column is **Employee**, we are unable to edit/delete that row.

#### APP.VUE

```

<template>
  <div id="app">

```

```

        <button v-on:click="btnClick">Grid is Addable</button>
        <ejs-grid :dataSource='data' :editSettings='editSettings'
:toolbar='toolbar' :cellEdit="cellEdit" :beforeBatchAdd="beforeBatchAdd"
:beforeBatchDelete="beforeBatchDelete" height='240px'>
            <e-columns>
                <e-column field='OrderID' headerText='Order ID'
textAlign='Right' :isPrimaryKey='true' width=100></e-column>
                <e-column field='Role' headerText='Role' width=120></e-
column>
                <e-column field='Freight' headerText='Freight' textAlign=
'Right' editType= 'numericedit' width=120 format= 'C2'></e-column>
                <e-column field='ShipCountry' headerText='Ship Country'
editType= 'dropdownedit' width=150></e-column>
            </e-columns>
        </ejs-grid>
    </div>
</template>
<script>
import Vue from "vue";
import { GridPlugin, Page, Toolbar, Edit } from "@syncfusion/ej2-vue-grids";
import { employeeData } from './datasource.js';
Vue.use(GridPlugin);
export default {
    data() {
        return {
            data: employeeData,
            editSettings: { allowEditing: true, allowAdding: true, allowDeleting:
true, mode: 'Batch' },
            toolbar: ['Add', 'Edit', 'Delete', 'Update', 'Cancel'],
            isAddable: true,
        };
    },
    methods: {
        cellEdit: function (args) {
            if (args.rowData["Role"] === "Employee") {
                args.cancel = true;
            }
        },
        beforeBatchAdd: function (args) {
            if (!this.isAddable) {
                args.cancel = true;
            }
        },
        beforeBatchDelete: function (args) {
            if (args.rowData["Role"] === "Employee") {
                args.cancel = true;
            }
        },
        btnClick: function (args) {
            args.target.innerText === "Grid is Addable" ? (args.target.innerText =
"Grid is Not Addable") : (args.target.innerText = "Grid is Addable");
            this.isAddable = !this.isAddable;
        },
    },
    provide: {
        grid: [Page, Edit, Toolbar]
    }
}

```

```

    }
  }
</script>
<style>
  @import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/grid/edit/default-cs3" %}

### Confirmation dialog

By default, grid will show the confirm dialog when saving or cancelling or performing any actions like filtering, sorting, etc.

### APP.VUE

```

<template>
  <div id="app">
    <ejs-grid :dataSource='data' :editSettings='editSettings'
:toolbar='toolbar' height='273px'>
      <e-columns>
        <e-column field='OrderID' headerText='Order ID'
textAlign='Right' :isPrimaryKey='true' width=100></e-column>
        <e-column field='CustomerID' headerText='Customer ID'
width=120></e-column>
        <e-column field='Freight' headerText='Freight' textAlign=
'Right' editType= 'numericedit' width=120 format= 'C2'></e-column>
        <e-column field='ShipCountry' headerText='Ship Country'
editType= 'dropdownedit' width=150></e-column>
      </e-columns>
    </ejs-grid>
  </div>
</template>
<script>
import Vue from "vue";
import { GridPlugin, Page, Toolbar, Edit } from "@syncfusion/ej2-vue-grids";
import { data } from './datasource.js';
Vue.use(GridPlugin);
export default {
  data() {
    return {
      data: data,
      editSettings: { showConfirmDialog: true, showDeleteConfirmDialog:
true, allowEditing: true, allowAdding: true, allowDeleting: true, mode:
'Batch' },
      toolbar: ['Add', 'Edit', 'Delete', 'Update', 'Cancel']
    };
  },
  provide: {
    grid: [Page, Edit, Toolbar]
  }
}
</script>
<style>
  @import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/grid/edit/default-cs4" %}

\* [editSettings.showConfirmDialog](#) requires the [editSettings.mode](#) to be **Batch**

\* If [editSettings.showConfirmDialog](#) set to **false**, then confirmation dialog does not display in batch editing.

#### *How to make editing in single click and arrow keys*

When using batch mode, the TAB key allows you to edit or move to the next cell or row from the current record by default. Using the grid's load event, the same functionality can also be achieved by pressing the arrow keys. Additionally, the `editCell` method of the grid allows for cells to be made editable with a single click.

In the following sample, the [load](#) event of the Grid will be used to bind the keydown event handler. When any arrow key is pressed, the `editCell` method of the Grid will be used to identify the next or previous cell (td) and set it to be editable. Additionally, it is possible to enable editing of a cell with a single click by utilizing the `editCell` method within the [created](#) event of the Grid.

#### **APP.VUE**

```
<template>
  <div id="app">
    <ejs-grid ref="grid" id="grid" :dataSource='data'
:allowPaging='true' :enableHover='false' :created='created' :load='load'
:editSettings='editSettings' :toolbar='toolbar' height='273px'>
      <e-columns>
        <e-column field='OrderID' headerText='Order ID'
textAlign='Right' :isPrimaryKey='true' width=100></e-column>
        <e-column field='CustomerID' headerText='Customer ID'
width=120></e-column>
        <e-column field='Freight' headerText='Freight' textAlign=
'Right' editType= 'numericedit' width=120 format= 'C2'></e-column>
        <e-column field='OrderDate' headerText='Order Date'
textAlign= 'Right' editType= 'datepickeredit' width=120 format= 'yMd'></e-
column>
        <e-column field='ShipCountry' headerText='Ship Country'
width=150></e-column>
      </e-columns>
    </ejs-grid>
  </div>
</template>
<script>
import Vue from "vue";
import { GridPlugin, Page, Toolbar, Edit } from "@syncfusion/ej2-vue-grids";
import { data } from './datasource.js';
import { isNullOrUndefined } from '@syncfusion/ej2-base';
Vue.use(GridPlugin);
export default {
  data() {
    return {
      data: data,
      editSettings: { allowEditing: true, allowAdding: true, allowDeleting:
true, mode: 'Batch' },
      toolbar: ['Add', 'Edit', 'Delete', 'Update', 'Cancel']
    };
  },
}
```

```

methods: {
  created: function() {
    let gridInstance =
document.getElementById("grid").ej2_instances[0];
    gridInstance.getContentTable().addEventListener('click', (args)
=> {
      if (args.target.classList.contains('e-rowcell')) {

gridInstance.editModule.editCell(parseInt(args.target.getAttribute('index'))
,

gridInstance.getColumnByIndex(parseInt(args.target.getAttribute('data-
colindex'))).field);
      }
    });
  },
  editACell: function(args) {
    let gridInstance =
document.getElementById("grid").ej2_instances[0];
    gridInstance.editModule.editCell(
      parseInt(args.getAttribute('index')),
      gridInstance.getColumnByIndex(parseInt(args.getAttribute('data-
colindex'))).field);
  },
  load: function() {
    let gridInstance =
document.getElementById("grid").ej2_instances[0];
    gridInstance.element.addEventListener('keydown', (e) => {
      var closesttd = e.target.closest('td');
      if (e.keyCode === 39 && !isNullOrUndefined(closesttd.nextSibling))
{
        this.editACell(closesttd.nextSibling);
      }
      if (e.keyCode === 37 &&
!isNullOrUndefined(closesttd.previousSibling) &&
!gridInstance.getColumnByIndex(
      parseInt(closesttd.previousSibling.getAttribute('data-
colindex'))).isPrimaryKey)
      {
        this.editACell(closesttd.previousSibling);
      }
      if (e.keyCode === 40 &&
!isNullOrUndefined(closesttd.closest('tr').nextSibling)) {
        this.editACell(
          closesttd.closest('tr').nextSibling.querySelectorAll('td')[
            parseInt(closesttd.getAttribute('data-colindex'))]);
      }
      if (e.keyCode === 38 &&
!isNullOrUndefined(closesttd.closest('tr').previousSibling)) {
        this.editACell(
          closesttd.closest('tr').previousSibling.querySelectorAll('td')[
            parseInt(closesttd.getAttribute('data-colindex'))]);
      }
    });
  },
}

```

```

    provide: {
      grid: [Page, Edit, Toolbar]
    }
  }
</script>
<style>
  @import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/grid/edit/default-cs5" %}

### Command column editing in Vue Grid component

The command column provides an option to add CRUD action buttons in a column. This can be defined by the [column.commands](#) property.

To use command column, inject the **CommandColumn** module in the **provide** section.

The available built-in command buttons are:

Command Button	Actions
Edit	Edit the current row.
Delete	Delete the current row.
Save	Update the edited row.
Cancel	Cancel the edited state.

### APP.VUE

```

<template>
  <div id="app">
    <ejs-grid :dataSource='data' :editSettings='editSettings'
height='310px'>
      <e-columns>
        <e-column field='OrderID' headerText='Order ID'
textAlign='Right' :isPrimaryKey='true' width=100></e-column>
        <e-column field='CustomerID' headerText='Customer ID'
width=120></e-column>
        <e-column field='Freight' headerText='Freight' textAlign=
'Right' editType= 'numericedit' width=120 format= 'C2'></e-column>
        <e-column field='ShipCountry' headerText='Ship Country'
editType= 'dropdownedit' width=150></e-column>
        <e-column headerText='Commands' width=120
:commands='commands'></e-column>
      </e-columns>
    </ejs-grid>
  </div>
</template>
<script>
import Vue from "vue";
import { GridPlugin, Page, Toolbar, Edit, CommandColumn } from
"@syncfusion/ej2-vue-grids";
import { data } from './datasource.js';
Vue.use(GridPlugin);

```



```

export default {
  data() {
    return {
      data: data,
      editSettings: { allowEditing: true, allowDeleting: true },
      commands: [{ type: 'Edit', buttonOption: { cssClass: 'e-flat',
        iconCss: 'e-edit e-icons' } },
        { type: 'Delete', buttonOption: { cssClass: 'e-flat', iconCss: 'e-
        delete e-icons' } },
        { type: 'Save', buttonOption: { cssClass: 'e-flat', iconCss: 'e-
        update e-icons' } },
        { type: 'Cancel', buttonOption: { cssClass: 'e-flat', iconCss: 'e-
        cancel-icon e-icons' } }]
    };
  },
  provide: {
    grid: [Page, Edit, Toolbar, CommandColumn]
  }
}
</script>
<style>
  @import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/grid/edit/default-cs6" %}

#### Custom command column

The custom command buttons can be added in a column by using the [column.commands](#) property and the action for the custom buttons can be defined using [commandClick](#) event.

#### APP.VUE

```

<template>
  <div id="app">
    <ejs-grid ref='grid' :dataSource='data' :editSettings='editSettings'
    :commandClick='commandClick' height='310px'>
      <e-columns>
        <e-column field='OrderID' headerText='Order ID'
        textAlign='Right' :isPrimaryKey='true' width=100></e-column>
        <e-column field='CustomerID' headerText='Customer ID'
        width=120></e-column>
        <e-column field='ShipCountry' headerText='Ship Country'
        editType= 'dropdownedit' width=150></e-column>
        <e-column headerText='Commands' width=150
        :commands='commands'></e-column>
      </e-columns>
    </ejs-grid>
  </div>
</template>
<script>
import Vue from "vue";
import { GridPlugin, Page, Toolbar, Edit, CommandColumn } from
"@syncfusion/ej2-vue-grids";
import { closest } from "@syncfusion/ej2-base";
import { data } from '../datasource.js';
Vue.use(GridPlugin);

```

```

export default {
  data() {
    return {
      data: data,
      editSettings: { allowEditing: true, allowDeleting: true },
      commands: [{ buttonOption: { content: 'Details', cssClass: 'e-flat' } }
    ]
  },
  provide: {
    grid: [Page, Edit, Toolbar, CommandColumn]
  },
  methods: {
    commandClick: function(args) {
      alert(JSON.stringify(args.rowData));
    }
  }
}
</script>
<style>
  @import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/grid/edit/default-cs7" %}

## Validation in Vue Grid component

### Column validation

Column validation allows you to validate the edited or added row data and it display errors for invalid fields before saving data. Grid uses **Form Validator** component for column validation. You can set validation rules by defining the [columns.validationRules](#).

### APP.VUE

```

<template>
  <div id="app">
    <ejs-grid :dataSource='data' :editSettings='editSettings'
    :toolbar='toolbar' height='273'>
      <e-columns>
        <e-column field='OrderID' headerText='Order ID'
        textAlign='Right' :isPrimaryKey='true' :validationRules='orderIDRules'
        width=100></e-column>
        <e-column field='CustomerID' headerText='Customer ID'
        :validationRules='customerIDRules' width=120></e-column>
        <e-column field='Freight' headerText='Freight' textAlign=
        'Right' editType= 'numericedit' width=120 format= 'C2'></e-column>
        <e-column field='ShipCountry' headerText='Ship Country'
        editType= 'dropdownedit' width=150></e-column>
      </e-columns>
    </ejs-grid>
  </div>
</template>
<script>
import Vue from "vue";
import { GridPlugin, Page, Toolbar, Edit } from "@syncfusion/ej2-vue-grids";
import { data } from './datasource.js';
Vue.use(GridPlugin);

```

```

export default {
  data() {
    return {
      data: data,
      editSettings: { allowEditing: true, allowAdding: true, allowDeleting:
true, mode: 'Normal' },
      toolbar: ['Add', 'Edit', 'Delete', 'Update', 'Cancel'],
      orderIDRules: { required: true },
      customerIDRules: { required: true, minLength: 3 }
    };
  },
  provide: {
    grid: [Page, Edit, Toolbar]
  }
}
</script>
<style>
  @import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/grid/edit/default-cs34" %}

#### Custom validation

You can define your own custom validation rules for the specific columns by using **Form Validator custom rules**.

In the below demo, custom validation applied for **CustomerID** column.

#### APP.VUE

```

<template>
  <div id="app">
    <ejs-grid :dataSource='data' :editSettings='editSettings'
:toolbar='toolbar' height='273'>
      <e-columns>
        <e-column field='OrderID' headerText='Order ID'
textAlign='Right' :isPrimaryKey='true' :validationRules='orderIDRules'
width=100></e-column>
        <e-column field='CustomerID' headerText='Customer ID'
:validationRules='customerIDRules' width=120></e-column>
        <e-column field='Freight' headerText='Freight' textAlign=
'Right' editType= 'numericedit' width=120 format= 'C2'></e-column>
        <e-column field='ShipCountry' headerText='Ship Country'
editType= 'dropdownedit' width=150></e-column>
      </e-columns>
    </ejs-grid>
  </div>
</template>
<script>
import Vue from "vue";
import { GridPlugin, Page, Toolbar, Edit } from "@syncfusion/ej2-vue-grids";
import { data } from './datasource.js';
Vue.use(GridPlugin);
export default {
  data() {
    return {
      data: data,

```

```

        editSettings: { allowEditing: true, allowAdding: true, allowDeleting:
true, mode: 'Normal' },
        toolbar: ['Add', 'Edit', 'Delete', 'Update', 'Cancel'],
        orderIDRules: { required: true },
        customerIDRules: { required: true, minLength: [(args) => {return
args['value'].length >= 5;}, 'Need atleast 5 letters'] }
    };
},
    provide: {
        grid: [Page, Edit, Toolbar]
    }
}
</script>
<style>
    @import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/grid/edit/default-cs35" %}

#### Custom validation based on dropdown change

You can apply validation rules and messages to a column based on another column value in edit mode. You can achieve this requirement by using the custom validation feature of Grid.

In the following sample, dropdownlist edit type is used for the **Role** and **Salary** columns. Here, you can apply the custom validation in the **Salary** column based on the value selected in the **Role** column.

#### APP.VUE

```

<template>
    <div id="app">
        <ejs-grid ref="grid" :dataSource="data" :editSettings="editSettings"
:toolbar="toolbar" :load="load">
            <e-columns>
                <e-column field="EmployeeID" headerText="Employee ID"
textAlign="Right" :isPrimaryKey="true" width="120"></e-column>
                <e-column field="Role" headerText="Role"
editType="dropdownedit" :edit="roleParams"
width="160"></e-column>
                <e-column field="Salary" headerText="Salary"
textAlign="Right" editType="dropdownedit"
width="160" :edit="salaryParams"></e-column>
                <e-column field="Address" headerText="Address"
width="150"></e-column>
            </e-columns>
        </ejs-grid>
    </div>
</template>
<script>
import Vue from "vue";
import { GridPlugin, Toolbar, Edit } from "@syncfusion/ej2-vue-grids";
import { employeeDetails } from '../datasource.js';
import { Query } from '@syncfusion/ej2-data';
Vue.use(GridPlugin);
window.role = "";
let jobRole = [
    { role: "TeamLead", destId: "0" },

```

```

    { role: "Manager", destId: "1" },
    { role: "Engineer", destId: "2" },
    { role: "Sales", destId: "3" },
    { role: "Support", destId: "4" },
];
let salaryDetails = [
    { salary: "11000", destId: "1" },
    { salary: "13500", destId: "2" },
    { salary: "16500", destId: "2" },
    { salary: "18500", destId: "1" },
    { salary: "21500", destId: "2" },
    { salary: "23000", destId: "2" },
];
export default {
    data() {
        return {
            data: employeeDetails,
            editSettings: {
                allowEditing: true,
                allowAdding: true,
                allowDeleting: true
            },
            toolbar: ["Add", "Edit", "Delete", "Update", "Cancel"],
            roleParams: {
                params: {
                    allowFiltering: true,
                    dataSource: jobRole,
                    fields: { value: "role", text: "role" },
                    query: new Query(),
                    change: this.valChange
                }
            },
            salaryParams: {
                params: {
                    allowFiltering: true,
                    dataSource: salaryDetails,
                    fields: { value: "salary", text: "salary" },
                    query: new Query()
                }
            },
            customFn: function (args) {
                switch (window["role"]) {
                    case "Engineer":
                        if (args.value > 10000 && args.value < 15000) {
                            return true;
                        } else {
                            this.rules["Salary"]["required"][1] = "Please
enter valid Engineer Salary";
                        }
                        break;
                    case "TeamLead":
                        if (args.value > 15000 && args.value < 20000) {
                            return true;
                        } else {
                            this.rules["Salary"]["required"][1] = "Please
enter valid TeamLead Salary";
                        }
                }
            }
        }
    }
}

```

```

        break;
        case "Manager":
            if (args.value > 20000 && args.value < 25000) {
                return true;
            } else {
                this.rules["Salary"]["required"][1] = "Please
enter valid Manager Salary";
            }
            break;
        case "Sales":
            if (args.value > 5000 && args.value < 25000) {
                return true;
            } else {
                this.rules["Salary"]["required"][1] = "Please
enter valid Manager Salary";
            }
            break;
        case "Support":
            if (args.value > 10000 && args.value < 19000) {
                return true;
            } else {
                this.rules["Salary"]["required"][1] = "Please
enter valid Manager Salary";
            }
            break;
    }
    return false;
}

};
},
methods: {
    load: function () {
        var column = this.$refs.grid.getColumnByField("Salary");
        column.validationRules = {
            required: [this.customFn, window["Please enter valid
salary"]],
        };
    },
    valChange: function (args) {
        window["role"] = args.value;
    }
},
provide: {
    grid: [Edit, Toolbar]
}
};
</script>
<style>
    @import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/grid/edit/default-cs36" %}

Persisting data in server in Vue Grid component

Edited data can be persisted in the database using the RESTful web services.

All the CRUD operations in the grid are done through [DataManager](#). The [DataManager](#) has an option to bind all the CRUD related data in server-side.

For your information, the ODataAdaptor persists data in the server as per OData protocol.

In the below section, we have explained how to get the edited data details on the server-side using the [UrlAdaptor](#).

#### *Using URL adaptor*

You can use the [UrlAdaptor](#) of [DataManager](#) when binding data source from remote data. In the initial load of grid, data are fetched from remote data and bound to the grid using **url** property of [DataManager](#). You can map The CRUD operation in grid can be mapped to server-side Controller actions using the properties **insertUrl**, **removeUrl**, **updateUrl**, **crudUrl** and [Link to the Video](#).

To learn about Server side CRUD operation with UrlAdaptor, you can check on this video:

The following code example describes the above behavior.

```
<template>
<div id="app">
<ejs-grid :dataSource='data' :editSettings='editSettings' :toolbar='toolbar' height='273px'>
<e-columns>
<e-column field='OrderID' headerText='Order ID' textAlign='Right' :isPrimaryKey='true'
:validationRules='orderIDRules' width=100></e-column>
<e-column field='CustomerID' headerText='Customer ID' :validationRules='customerIDRules'
width=120></e-column>
<e-column field='Freight' headerText='Freight' textAlign= 'Right' editType= 'numericedit' width=120
format= 'C2'></e-column>
<e-column field='ShipCountry' headerText='Ship Country' editType= 'dropdownedit' width=150></e-
column>
</e-columns>
</ejs-grid>
</div>
</template>
<script>
import Vue from "vue";
import { GridPlugin, Page, Toolbar, Edit } from "@syncfusion/ej2-vue-grids";
import { DataManager, UrlAdaptor } from '@syncfusion/ej2-data';
Vue.use(GridPlugin);
export default Vue.extend({
data() {
```

```

return {
  data: new DataManager({
    url: "Home/DataSource",
    updateUrl: "Home/Update",
    insertUrl: "Home/Insert",
    removeUrl: "Home/Delete",
    adaptor: new UrlAdaptor
  }),
  editSettings: { allowEditing: true, allowAdding: true, allowDeleting: true, mode: 'Normal' },
  toolbar: ['Add', 'Edit', 'Delete', 'Update', 'Cancel'],
  orderIDRules: { required: true },
  customerIDRules: { required: true, minLength: 3 }
};
},
provide: {
  grid: [Page, Edit, Toolbar]
}
});
</script>
<style>
@import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
</style>
`

```

Also, when using the [UrlAdaptor](#), you need to return the data as JSON from the controller action and the JSON object must contain a property as **result** with dataSource as its value and one more property **count** with the dataSource total records count as its value.

The following code example describes the above behavior.

```

`ts
public ActionResult DataSource(DataManager dm)
{
  var DataSource = OrderRepository.GetAllRecords();
  DataResult result = new DataResult();
  result.result = DataSource.Skip(dm.Skip).Take(dm.Take).ToList();
  result.count = result.result.Count;
}

```



```

return Json(result, JsonRequestBehavior.AllowGet);
}
public class DataResult
{
    public List<EditableOrder> result { get; set; }
    public int count { get; set; }
}

```

#### Insert record

Using the **insertUrl** property, you can specify the controller action mapping URL to perform insert operation on the server-side.

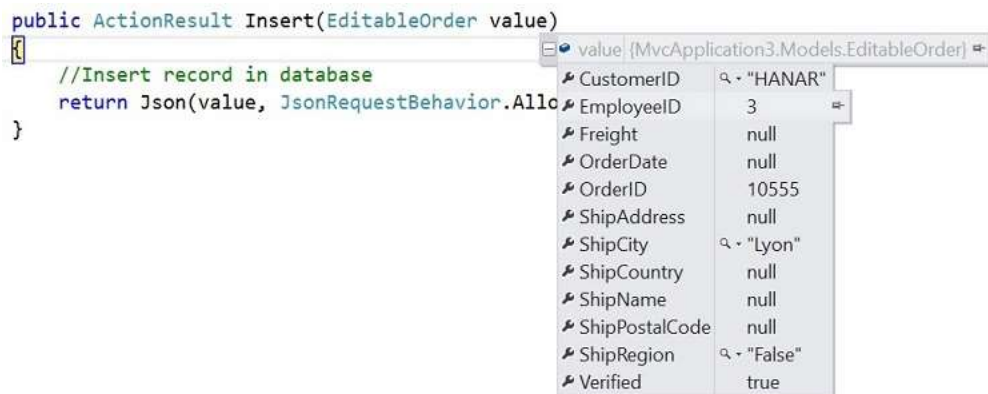
The following code example describes the above behavior.

```

`ts
public ActionResult Insert(EditableOrder value)
{
    //Insert record in database
}

```

The newly added record details are bound to the **value** parameter. Please refer to the following screenshot.



#### Update record

Using the **updateUrl** property, the controller action mapping URL can be specified to perform save/update operation on the server-side.

The following code example describes the previous behavior.

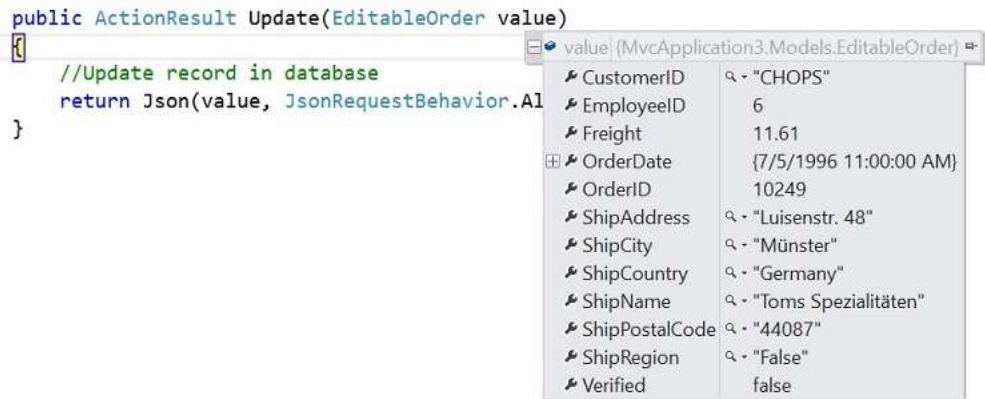
```

`ts
public ActionResult Update(EditableOrder value)
{

```

```
//Update record in database
}
,
```

The updated record details are bound to the **value** parameter. Please refer to the following screenshot.



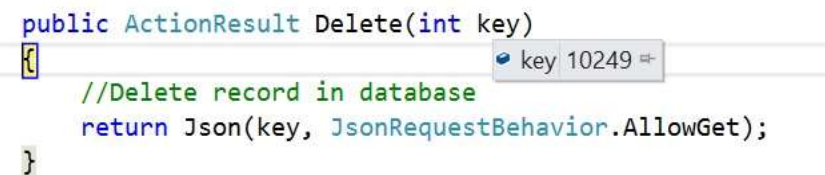
#### Delete record

Using the **removeUrl** property, the controller action mapping URL can be specified to perform delete operation on the server-side.

The following code example describes the previous behavior.

```
`ts
public ActionResult Delete(int key)
{
    //Delete record in database
}
,
```

The deleted record primary key value is bound to the **key** parameter. Please refer to the following screenshot.



#### CRUD URL

Using the **crudUrl** property, the controller action mapping URL can be specified to perform all the CRUD operation at server-side using a single method instead of specifying separate controller action method for CRUD (insert, update and delete) operations.

The action parameter of **crudUrl** is used to get the corresponding CRUD action.

The following code example describes the above behavior.

```

<template>
<div id="app">
<ejs-grid :dataSource='data' :editSettings='editSettings' :toolbar='toolbar' height='273px'>
<e-columns>
<e-column field='OrderID' headerText='Order ID' textAlign='Right' :isPrimaryKey='true'
:validationRules='orderIDRules' width=100></e-column>
<e-column field='CustomerID' headerText='Customer ID' :validationRules='customerIDRules'
width=120></e-column>
<e-column field='Freight' headerText='Freight' textAlign= 'Right' editType= 'numericedit' width=120
format= 'C2'></e-column>
<e-column field='ShipCountry' headerText='Ship Country' editType= 'dropdownedit' width=150></e-
column>
</e-columns>
</ejs-grid>
</div>
</template>
<script>
import Vue from "vue";
import { GridPlugin, Page, Toolbar, Edit } from "@syncfusion/ej2-vue-grids";
import { DataManager, UrlAdaptor } from '@syncfusion/ej2-data';
Vue.use(GridPlugin);
export default Vue.extend({
data() {
return {
data: new DataManager({
url: "Home/DataSource",
crudUrl: "Home/CrudUpdate",
adaptor: new UrlAdaptor
}),
editSettings: { allowEditing: true, allowAdding: true, allowDeleting: true, mode: 'Normal' },
toolbar: ['Add', 'Edit', 'Delete', 'Update', 'Cancel'],
orderIDRules: { required: true },
customerIDRules: { required: true, minLength: 3 }
}
}
}

```

```
};  
},  
provide: {  
  grid: [Page, Edit, Toolbar]  
}  
});  
</script>  
<style>  
@import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";  
</style>  
,  
  
`ts  
public ActionResult CrudUpdate(EditableOrder value, string action)  
{  
  if(action == "update"){  
    //Update record in database  
  }  
  else if (action == "insert")  
  {  
    //Insert record in database  
  }  
  else  
  {  
    //Delete record in database  
  }  
}  
,
```

Please refer to the following screenshot to know about the action parameter.

```

public ActionResult CrudUpdate(EditableOrder value, string action)
{
    if(action == "update"){
        //Update record in database
    }
    else if (action == "insert")
    {
        //Insert record in database
    }
    else
    {
        //Delete record in database
    }
    return Json(value, JsonRequestBehavior.AllowGet);
}

```

If you specify **insertUrl** along with **crudUrl**, then while adding **insertUrl** only will be invoked.

#### Batch URL

The **batchUrl** property supports only for batch editing mode. You can specify the controller action mapping URL to perform batch operation on the server-side.

The following code example describes the above behavior.

```

<template>
<div id="app">
<ejs-grid :dataSource='data' :editSettings='editSettings' :toolbar='toolbar' height='273px'>
<e-columns>
<e-column field='OrderID' headerText='Order ID' textAlign='Right' :isPrimaryKey='true'
:validationRules='orderIDRules' width=100></e-column>
<e-column field='CustomerID' headerText='Customer ID' :validationRules='customerIDRules'
width=120></e-column>
<e-column field='Freight' headerText='Freight' textAlign= 'Right' editType= 'numericedit' width=120
format= 'C2'></e-column>
<e-column field='ShipCountry' headerText='Ship Country' editType= 'dropdownedit' width=150></e-
column>
</e-columns>
</ejs-grid>
</div>
</template>
<script>
import Vue from "vue";
import { GridPlugin, Page, Toolbar, Edit } from "@syncfusion/ej2-vue-grids";
import { DataManager, UrlAdaptor } from '@syncfusion/ej2-data';

```

```

Vue.use(GridPlugin);
export default Vue.extend({
  data() {
    return {
      data: new DataManager({
        url: "Home/DataSource",
        batchUrl: "Home/BatchUpdate",
        adaptor: new UrlAdaptor
      });
      editSettings: { allowEditing: true, allowAdding: true, allowDeleting: true, mode: 'Batch' },
      toolbar: ['Add', 'Edit', 'Delete', 'Update', 'Cancel'],
      orderIDRules: { required: true },
      customerIDRules: { required: true, minLength: 3 }
    };
  },
  provide: {
    grid: [Page, Edit, Toolbar]
  }
});
</script>
<style>
@import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
</style>
`ts
public ActionResult BatchUpdate(string action, List<EditableOrder> added, List<EditableOrder> changed,
List<EditableOrder> deleted, int? key)
{
  //Save the batch changes in database
}
`

```

```
public ActionResult BatchUpdate(string action, List<EditableOrder> added, List<EditableOrder> changed, List<EditableOrder> deleted, int? key)
{
    //Save the batch changes in database
    return Json(changed, JsonRequestBehavior.AllowGet);
}
```

changed Count = 2

(MvcApplication3.Models.EditableOrder)	
CustomerID	4 - "HANAR-Updated"
EmployeeID	4
Freight	65.83
OrderDate	[7/8/1996 5:30:00 AM]
OrderID	10250
ShipAddress	"Rua do Paço, 67"
ShipCity	"Rio de Janeiro"
ShipCountry	"Brazil"
ShipName	"Hanari Carnes"
ShipPostalCode	"05454-876"
ShipRegion	"RJ"
Verified	true

## Sorting in Vue Grid component

The Grid component has support to sort data bound columns in **ascending** or **descending** order. This can be achieved by setting [allowSorting](#) property as true.

To dynamically sort a particular column, click on its column header. The order switch between **Ascending** and **Descending** each time you click a column header for sorting.

To use Sorting, you need to inject [Sort](#) module in the **provide** section.

### APP.VUE

```
<template>
  <div id="app">
    <ejs-grid :dataSource='data' :allowSorting='true' height='315px'>
      <e-columns>
        <e-column field='OrderID' headerText='Order ID'
textAlign='Right' width=120></e-column>
        <e-column field='CustomerID' headerText='Customer ID'
width=150></e-column>
        <e-column field='ShipCity' headerText='Ship City'
width=150></e-column>
        <e-column field='ShipName' headerText='Ship Name'
width=150></e-column>
      </e-columns>
    </ejs-grid>
  </div>
</template>
<script>
import Vue from "vue";
import { GridPlugin, Sort } from "@syncfusion/ej2-vue-grids";
import { data } from './datasource.js';
Vue.use(GridPlugin);
export default {
  data() {
    return {
      data: data
    };
  },
  provide: {
    grid: [Sort]
  }
}
</script>
<style>
@import
"https://ej2.syncfusion.com/vue/documentation/node_modules/@syncfusion/ej2-
vue-grids/styles/material.css";
</style>
```

```
{% previewsample "page.domainurl/code-snippet/grid/sort/default-cs1" %}
```

\* Grid column sorted in **Ascending** order. If you click on an already sorted column, then toggles the sort direction.

\* You can apply and clear sorting by invoking [sortColumn](#) and [clearSorting](#) methods.

\* To disable Sorting for a particular column, by specifying [columns.allowSorting](#) to false.

### Initial Sort

By default, the Grid records are not sorted at initial rendering. To apply sort at initial rendering, set the [field](#) and [direction](#) in **sortSettings.columns**.

### APP.VUE

```
<template>
  <div id="app">
    <ejs-grid :dataSource='data' :allowSorting='true'
    :sortSettings='sortOptions' height='315px'>
      <e-columns>
        <e-column field='OrderID' headerText='Order ID'
        textAlign='Right' width=120></e-column>
        <e-column field='CustomerID' headerText='Customer ID'
        width=150></e-column>
        <e-column field='ShipCity' headerText='Ship City'
        width=150></e-column>
        <e-column field='ShipName' headerText='Ship Name'
        width=150></e-column>
      </e-columns>
    </ejs-grid>
  </div>
</template>
<script>
import Vue from "vue";
import { GridPlugin, Sort } from "@syncfusion/ej2-vue-grids";
import { data } from './datasource.js';
Vue.use(GridPlugin);
export default {
  data() {
    return {
      data: data,
      sortOptions: { columns: [{ field: 'OrderID', direction: 'Ascending' },
      { field: 'ShipCity', direction: 'Descending' }] }
    };
  },
  provide: {
    grid: [Sort]
  }
}
</script>
<style>
@import
"https://ej2.syncfusion.com/vue/documentation/node_modules/@syncfusion/ej2-
vue-grids/styles/material.css";
</style>
```



```
{% previewsample "page.domainurl/code-snippet/grid/sort/default-cs2" %}
```

### Multi-column sorting

You can sort more than one column in a Grid. To sort multiple columns, press and hold the **CTRL** key and click the column header. The sorting order will be displayed in the header while performing multi-column sorting.

To clear sorting for a particular column, press the "Shift + mouse left click".

The [allowSorting](#) must be true while enabling multi-column sort.

Set [allowMultiSorting](#) property as **false** to disable multi-column sorting.

### APP.VUE

```
<template>
  <div id="app">
    <ejs-grid :dataSource='data' :allowSorting='true'
:allowMultiSorting='true' height='315px'>
      <e-columns>
        <e-column field='OrderID' headerText='Order ID'
textAlign='Right' width=120></e-column>
        <e-column field='CustomerID' headerText='Customer ID'
width=150></e-column>
        <e-column field='ShipCity' headerText='Ship City'
width=150></e-column>
        <e-column field='ShipName' headerText='Ship Name'
width=150></e-column>
      </e-columns>
    </ejs-grid>
  </div>
</template>
<script>
import Vue from "vue";
import { GridPlugin, Sort } from "@syncfusion/ej2-vue-grids";
import { data } from './datasource.js';
Vue.use(GridPlugin);
export default {
  data() {
    return {
      data: data
    };
  },
  provide: {
    grid: [Sort]
  }
}
</script>
<style>
@import
"https://ej2.syncfusion.com/vue/documentation/node_modules/@syncfusion/ej2-
vue-grids/styles/material.css";
</style>
```

```
{% previewsample "page.domainurl/code-snippet/grid/sort/default-cs3" %}
```

### Sort order

By default, the sorting order will be as **ascending** -> **descending** -> **none**.

When first click a column header it sorts the column in ascending. Again click the same column header, it will sort the column in descending. A repetitive third click on the same column header will clear the sorting.

### Sort foreign key column based on Text

For local data in Grid, sorting will be performed based on the [foreignKeyValue](#).

For remote data in Grid, sorting will be performed based on the [foreignKeyField](#), we need to handle the sorting operation at the server side.

The following code example describes the handling of sorting operation at the server side.

```
<template>
<div id="app">
<ejs-grid :dataSource='data' height='315' :allowSorting='true'>
<e-columns>
<e-column field='OrderID' headerText='Order ID' textAlign='Right' width=100></e-column>
<e-column field='EmployeeID' headerText='Employee Name' width=120 foreignKeyValue='FirstName'
:dataSource='employeeData'></e-column>
<e-column field='Freight' headerText='Freight' textAlign='Right' width=80></e-column>
<e-column field='ShipCity' headerText='Ship City' width=130 ></e-column>
</e-columns>
</ejs-grid>
</div>
</template>
<script>
import Vue from "vue";
import { GridPlugin, ForeignKey, Sort } from "@syncfusion/ej2-vue-grids";
import { DataManager, ODataV4Adaptor } from "@syncfusion/ej2-data";
import { data, employeeData } from './datasource.js';
Vue.use(GridPlugin);
export default {
data() {
return {
data: new DataManager({
url:'/OData/Items',
```

```

adaptor: new ODataV4Adaptor
}},
employeeData: new DataManager({
url:'/OData/Brands',
adaptor: new ODataV4Adaptor
})
};
},
provide: {
grid: [ForeignKey, Sort]
}
}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
</style>
`

```

The following code example describes the handling of sorting operation at server side.

```

public class ItemsController : ODataController
{
[EnableQuery]
public IQueryable<Item> Get()
{
List<Item> GridData =
JsonConvert.DeserializeObject<Item[]>(Properties.Resources.ItemsJson).AsQueryable().ToList();

List<Brand> empData =
JsonConvert.DeserializeObject<Brand[]>(Properties.Resources.BrandsJson).AsQueryable().ToList();

var queryString = HttpContext.Current.Request.QueryString;
var allUrlKeyValues = ControllerContext.Request.GetQueryNameValuePairs();
string key = allUrlKeyValues.LastOrDefault(x => x.Key == "$orderby").Value;
if (key != null)
{
if (key == "EmployeeID") {

```

```

GridData = SortFor(key); //Only for foreignKey Column ascending
}
else if(key == "EmployeeID desc") {
GridData = SortFor(key); //Only for foreignKey Column descending
}
}
var count = GridData.Count();
var data = GridData.AsQueryable();
return data;
}
public List<Item> SortFor(String Sorted)
{
List<Item> GridData =
JsonConvert.DeserializeObject<Item[]>(Properties.Resources.ItemsJson).AsQueryable().ToList();
List<Brand> empData =
JsonConvert.DeserializeObject<Brand[]>(Properties.Resources.BrandsJson).AsQueryable().ToList();
if (Sorted == "EmployeeID") //check whether ascending or descending
empData = empData.OrderBy(e => e.FirstName).ToList();
else if(Sorted == "EmployeeID desc")
empData = empData.OrderByDescending(e => e.FirstName).ToList();
List<Item> or = new List<Item>();
for (int i = 0; i < empData.Count(); i++) {
//Select the Field matching records
IEnumerable<Item> list = GridData.Where(pred => pred.EmployeeID ==
empData[i].EmployeeID).ToList();
or.AddRange(list);
}
return or;
}
}
`

```

### Sorting Events

During the sort action, the Grid component triggers two events. [actionBegin](#) event triggers before the sort action start and [actionComplete](#) event triggers after the sort action complete. Using these events you can perform any actions.

**APP.VUE**

```

<template>
  <div id="app">
    <ejs-grid :dataSource='data' :actionComplete='actionHandler'
    :actionBegin='actionHandler' :allowSorting='true' height='315px'>
      <e-columns>
        <e-column field='OrderID' headerText='Order ID'
        textAlign='Right' width=120></e-column>
        <e-column field='CustomerID' headerText='Customer ID'
        width=150></e-column>
        <e-column field='ShipCity' headerText='Ship City'
        width=150></e-column>
        <e-column field='ShipName' headerText='Ship Name'
        width=150></e-column>
      </e-columns>
    </ejs-grid>
  </div>
</template>
<script>
import Vue from "vue";
import { GridPlugin, Sort } from "@syncfusion/ej2-vue-grids";
import { data } from './datasource.js';
Vue.use(GridPlugin);
export default {
  data() {
    return {
      data: data
    };
  },
  methods: {
    actionHandler: function(args) {
      alert(args.requestType + ' ' + args.type); // custom Action
    }
  },
  provide: {
    grid: [Sort]
  }
}
</script>
<style>
@import
"https://ej2.syncfusion.com/vue/documentation/node_modules/@syncfusion/ej2-
vue-grids/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/grid/sort/default-cs4" %}

[args.requestType](#) is current action name. For example in sorting, the [args.requestType](#) value is **sorting**.

**Sort Comparer**

You can customize the default sort action for a column by defining [column.sortComparer](#) property. The sort comparer function has the same functionality like [Array.sort](#) sort comparer.

In the following example, custom sort comparer function was defined in **Customer ID** column.

**APP.VUE**

```

<template>
  <div id="app">
    <ejs-grid :dataSource='data' :allowSorting='true' height='315px'>
      <e-columns>
        <e-column field='OrderID' headerText='Order ID'
textAlign='Right' width=120></e-column>
        <e-column field='CustomerID' :sortComparer='sortComparer'
headerText='Customer ID' width=150></e-column>
        <e-column field='Freight' headerText='Freight' width=100
format='C2'></e-column>
        <e-column field='ShipName' headerText='Ship Name'
width=150></e-column>
      </e-columns>
    </ejs-grid>
  </div>
</template>
<script>
import Vue from "vue";
import { GridPlugin, Sort } from "@syncfusion/ej2-vue-grids";
import { data } from './datasource.js';
Vue.use(GridPlugin);
export default {
  data() {
    return {
      data: data
    };
  },
  methods: {
    sortComparer: function(reference, comparer) {
      if (reference < comparer) {
        return -1;
      }
      if (reference > comparer) {
        return 1;
      }
      return 0;
    }
  },
  provide: {
    grid: [Sort]
  }
}
</script>
<style>
  @import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/grid/sort/default-cs5" %}

### Touch Interaction

When you tap Grid header on touchscreen devices, then the selected column header is sorted. Also, it



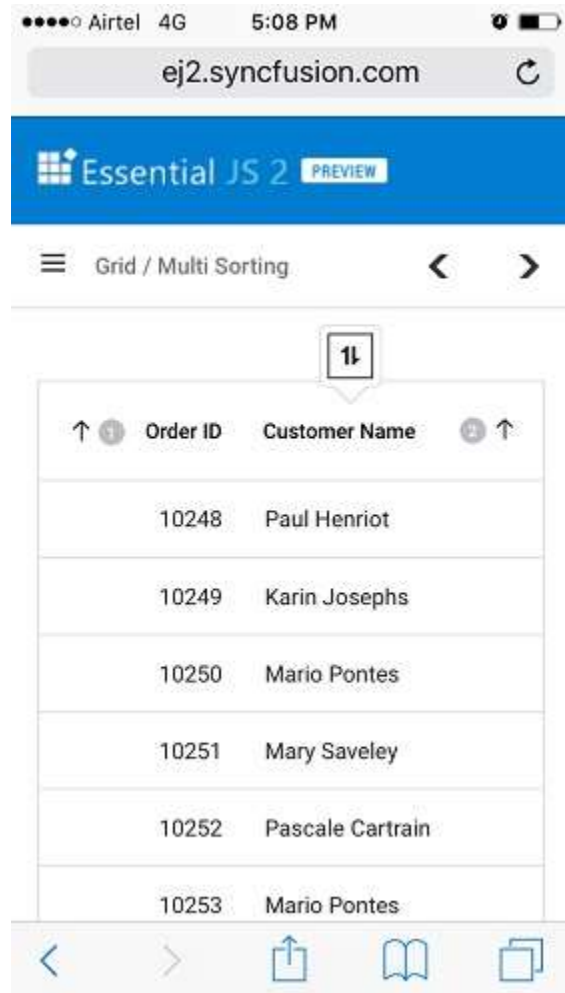
will show a popup for multi-column sorting.



To sort multiple columns, tap the popup and then tap the desired Grid headers.

The [allowMultiSorting](#) and [allowSorting](#) should be **true** then only the popup will be shown.

The following screenshot represents a Grid touch sorting in the device.



See Also

- [Switching the column sort order between ascending and descending order in Vue Grid](#)

## Grouping

### Grouping in Vue Grid component

The Grid has options to group records by dragging and dropping the column header to the group drop area. When grouping is applied, grid records are organized into a hierarchical structure to facilitate easier expansion and collapse of records.

To enable Grouping in the grid, set the [allowGrouping](#) to true. Grouping options can be configured in [groupSettings](#).

To use Grouping, you need to inject [Group](#) module in the [Link to the Video](#) section.

To get start quickly with Grouping Options, you can check on this video:

### APP.VUE

```
<template>
  <div id="app">
    <ejs-grid :dataSource='data' :allowGrouping='true' height='267px'>
      <e-columns>
        <e-column field='OrderID' headerText='Order ID'
textAlign='Right' width=120></e-column>
        <e-column field='CustomerID' headerText='Customer ID'
width=150></e-column>
        <e-column field='ShipCity' headerText='Ship City'
width=150></e-column>
        <e-column field='ShipName' headerText='Ship Name'
width=150></e-column>
      </e-columns>
    </ejs-grid>
  </div>
</template>
<script>
import Vue from "vue";
import { GridPlugin, Group } from "@syncfusion/ej2-vue-grids";
import { data } from './datasource.js';
Vue.use(GridPlugin);
export default {
  data() {
    return {
      data: data
    };
  },
  provide: {
    grid: [Group]
  }
}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
</style>
```

{% previewsample "page.domainurl/code-snippet/grid/group/default-cs2" %}

\* You can group and ungroup columns by using the [groupColumn](#) and [ungroupColumn](#) methods.

\* To disable grouping for a particular column, set the [columns.allowGrouping](#) to false.

#### Initial group

To apply group at initial rendering, set the column field name in the [groupSettings.columns](#).

### APP.VUE

```
<template>
  <div id="app">
    <ejs-grid :dataSource='data' :allowGrouping='true'
:groupSettings='groupOptions' height='267px'>
      <e-columns>
```



```

        <e-column field='OrderID' headerText='Order ID'
textAlign='Right' width=120></e-column>
        <e-column field='CustomerID' headerText='Customer ID'
width=150></e-column>
        <e-column field='ShipCity' headerText='Ship City'
width=150></e-column>
        <e-column field='ShipName' headerText='Ship Name'
width=150></e-column>
    </e-columns>
</ejs-grid>
</div>
</template>
<script>
import Vue from "vue";
import { GridPlugin, Group } from "@syncfusion/ej2-vue-grids";
import { data } from './datasource.js';
Vue.use(GridPlugin);
export default {
  data() {
    return {
      data: data,
      groupOptions: { columns: ['CustomerID', 'ShipCity'] }
    };
  },
  provide: {
    grid: [Group]
  }
}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/grid/group/default-cs3" %}

#### Hide drop area

To avoid ungrouping or further grouping of a column after initial column grouping, define the [groupSettings.showDropArea](#) as false.

#### APP.VUE

```

<template>
  <div id="app">
    <ejs-grid :dataSource='data' :allowGrouping='true'
:groupSettings='groupOptions' height='267px'>
      <e-columns>
        <e-column field='OrderID' headerText='Order ID'
textAlign='Right' width=120></e-column>
        <e-column field='CustomerID' headerText='Customer ID'
width=150></e-column>
        <e-column field='ShipCity' headerText='Ship City'
width=150></e-column>
        <e-column field='ShipName' headerText='Ship Name'
width=150></e-column>
      </e-columns>
    </ejs-grid>
  </div>
</template>

```

```

    </div>
</template>
<script>
import Vue from "vue";
import { GridPlugin, Group } from "@syncfusion/ej2-vue-grids";
import { data } from '../datasource.js';
Vue.use(GridPlugin);
export default {
  data() {
    return {
      data: data,
      groupOptions: { showDropArea: false, columns: ['CustomerID',
'ShipCity'] }
    };
  },
  provide: {
    grid: [Group]
  }
}
</script>
<style>
  @import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/grid/group/default-cs4" %}

#### Group with paging

On grouping columns with paging feature, the aggregated information and total items are displayed based on the current page. The grid does not consider aggregated information and total items from other pages. To get additional details (aggregated information and total items) from other pages, set the [groupSettings.disablePageWiseAggregates](#) to true.

If remote data is bound to grid dataSource, two requests will be sent when performing grouping action; one for getting the grouped data and another for getting aggregate details and total items count.

#### Group by format

By default, columns will be grouped by the data or value present for the particular row. To group numeric or datetime column based on the mentioned format, you have to enable the [enableGroupByFormat](#) property of the corresponding grid columns.

### APP.VUE

```

<template>
  <div id="app">
    <ejs-grid :dataSource='data' :allowGrouping='true'
:groupSettings='groupOptions' height='315px'>
      <e-columns>
        <e-column field='OrderID' headerText='Order ID'
textAlign='Right' width=120></e-column>
        <e-column field='CustomerID' headerText='Customer ID'
width=150></e-column>
        <e-column field='OrderDate' headerText='Order Date'
format='yMMM' :enableGroupByFormat='true' width=150 type='date'></e-column>
        <e-column field='Freight' headerText='Freight' format='C2'
:enableGroupByFormat='true' width=150></e-column>
      </e-columns>
    </div>
  </template>

```

```

        </ejs-grid>
    </div>
</template>
<script>
import Vue from "vue";
import { GridPlugin, Group } from "@syncfusion/ej2-vue-grids";
import { data } from './datasource.js';
Vue.use(GridPlugin);
export default {
  data() {
    return {
      data: data,
      groupOptions: { columns: ['OrderDate', 'Freight'] }
    };
  },
  provide: {
    grid: [Group]
  }
}
</script>
<style>
  @import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/grid/group/default-cs5" %}

### Grouping events

During the group action, the grid component triggers two events. The [actionBegin](#) event triggers before the group action starts and the [actionComplete](#) event triggers after the group action is completed. Using these events you can perform any action.

### APP.VUE

```

<template>
  <div id="app">
    <ejs-grid :dataSource='data' :allowGrouping='true'
      :actionComplete='actionHandler' :actionBegin='actionHandler' height='260px'>
      <e-columns>
        <e-column field='OrderID' headerText='Order ID'
          textAlign='Right' width=120></e-column>
        <e-column field='CustomerID' headerText='Customer ID'
          width=150></e-column>
        <e-column field='ShipCity' headerText='Ship City'
          width=150></e-column>
        <e-column field='ShipName' headerText='Ship Name'
          width=150></e-column>
      </e-columns>
    </ejs-grid>
  </div>
</template>
<script>
import Vue from "vue";
import { GridPlugin, Group } from "@syncfusion/ej2-vue-grids";
import { data } from './datasource.js';
Vue.use(GridPlugin);
export default {

```

```

data() {
  return {
    data: data
  };
},
methods: {
  actionHandler: function(args) {
    alert(args.requestType + ' ' + args.type); // custom Action
  }
},
provide: {
  grid: [Group]
}
}
</script>
<style>
  @import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/grid/group/default-cs6" %}

[args.requestType](#) is current action name. For example in grouping, the [args.requestType](#) value is **grouping**.

#### *Collapse by external button*

To collapse the selected grouped row from an external button by using the [expandCollapse](#) method.

### **APP.VUE**

```

<template>
  <div id="app">
    <ejs-button id='collapse' cssClass='e-flat'
    @click.native='collapse'>Collapse</ejs-button>
    <ejs-grid ref='grid' :dataSource='data' :allowGrouping='true'
    :groupSettings='groupSettings' height='240px'>
      <e-columns>
        <e-column field='OrderID' headerText='Order ID'
        textAlign='Right' width=120></e-column>
        <e-column field='CustomerID' headerText='Customer ID'
        width=150></e-column>
        <e-column field='ShipCity' headerText='Ship City'
        width=150></e-column>
        <e-column field='ShipName' headerText='Ship Name'
        width=150></e-column>
      </e-columns>
    </ejs-grid>
  </div>
</template>
<script>
import Vue from "vue";
import { GridPlugin, Group } from "@syncfusion/ej2-vue-grids";
import { ButtonPlugin } from "@syncfusion/ej2-vue-buttons";
import { data } from './datasource.js';
Vue.use(GridPlugin);
Vue.use(ButtonPlugin);
export default {

```

```

data() {
  return {
    data: data,
    groupSettings: { columns: ['CustomerID'] }
  };
},
methods: {
  collapse: function() {
    if (this.$refs.grid.getSelectedRowIndex().length) {
      let currentTr =
this.$refs.grid.getRows()[this.$refs.grid.getSelectedRowIndex()[0]];
      while (currentTr.classList && currentTr.classList.length) {
        currentTr = currentTr.previousSibling;
      }
      let collapseElement = currentTr.querySelector('.e-
recordplusexpand');

this.$refs.grid.ej2Instances.groupModule.expandCollapseRows(collapseElement)
; // pass the collapse row element.
    }
  },
  provide: {
    grid: [Group]
  }
}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/grid/group/default-cs7" %}

#### Sort grouped columns in descending order during initial grouping

By default, grouped columns are sorted in ascending order. To sort grouped columns in descending order during initial grouping, you can set the [field](#) and [direction](#) in the `sortSettings.columns` property.

The `CustomerID` column will be sorted in descending order when the grid is initially grouped, as shown in the following example.

#### APP.VUE

```

<template>
  <div id="app">
    <ejs-grid :dataSource='data' :allowGrouping='true'
:allowSorting='true' :sortSettings='sortOptions'
:groupSettings='groupSettings' height='267px'>
      <e-columns>
        <e-column field='OrderID' headerText='Order ID'
textAlign='Right' width=120></e-column>
        <e-column field='CustomerID' headerText='Customer ID'
width=150></e-column>
        <e-column field='ShipCity' headerText='Ship City'
width=150></e-column>
        <e-column field='ShipName' headerText='Ship Name'
width=150></e-column>
      </e-columns>
    </div>
  </template>

```

```

        </e-columns>
      </ejs-grid>
    </ejs-grid>
  </div>
</template>
<script>
import Vue from "vue";
import { GridPlugin, Group, Sort } from "@syncfusion/ej2-vue-grids";
import { data } from './datasource.js';
Vue.use(GridPlugin);
export default {
  data() {
    return {
      data: data,
      sortOptions: { columns: [{ field: 'CustomerID', direction:
'Descending' }] },
      groupSettings: { columns: ['CustomerID'] }
    };
  },
  provide: {
    grid: [Group, Sort]
  }
}
</script>
<style>
  @import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/grid/group/default-cs8" %}

[See Also](#)

- [Exporting grouped records](#)
- [How to customize the group caption row text in Vue Grid](#)

### Caption template in Vue Grid component

You can customize the group caption by using the [groupSettings.captionTemplate](#) property.

#### APP.VUE

```

<template>
  <div id="app">
    <ejs-grid :dataSource='data' :allowGrouping='true'
:groupSettings='groupOptions' height='267px'>
      <e-columns>
        <e-column field='OrderID' headerText='Order ID'
textAlign='Right' width=120></e-column>
        <e-column field='CustomerID' headerText='Customer ID'
width=150></e-column>
        <e-column field='ShipCity' headerText='Ship City'
width=150></e-column>
        <e-column field='ShipName' headerText='Ship Name'
width=150></e-column>
      </e-columns>
    </ejs-grid>
  </div>
</template>

```

```

    </div>
</template>
<script>
import Vue from "vue";
import { GridPlugin, Group } from "@syncfusion/ej2-vue-grids";
import { data } from './datasource.js';
Vue.use(GridPlugin);
export default {
  data() {
    return {
      data: data,
      groupOptions: { showDropArea: false, columns: ['CustomerID',
'ShipCity'],
      captionTemplate: '<span class="groupItems" style="color:blue">
${count} Items</span>' }
    };
  },
  provide: {
    grid: [Group]
  }
}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/grid/group/default-cs1" %}

### Reordering on grouped columns in Vue Grid component

Grid provides an option to interchange the position of the Grouped Columns by dragging and dropping the grouped headercells in the droparea. So, any new column entering into the droparea can also be dropped in any position.

To enable this feature, you have to set the [groupSettings.allowReordering](#) property as **true**.

### APP.VUE

```

<template>
  <div id="app">
    <ejs-grid ref='grid' :dataSource='data' :allowGrouping='true'
:groupSettings='groupSettings' height='240px'>
      <e-columns>
        <e-column field='OrderID' headerText='Order ID'
textAlign='Right' width=120></e-column>
        <e-column field='CustomerID' headerText='Customer ID'
width=150></e-column>
        <e-column field='ShipCity' headerText='Ship City'
width=150></e-column>
        <e-column field='ShipName' headerText='Ship Name'
width=150></e-column>
      </e-columns>
    </ejs-grid>
  </div>
</template>
<script>
import Vue from "vue";

```

```

import { GridPlugin, Group } from "@syncfusion/ej2-vue-grids";
import { data } from './datasource.js';
Vue.use(GridPlugin);
export default {
  data() {
    return {
      data: data,
      groupSettings: { columns: ['CustomerID'], allowReordering: true }
    };
  },
  methods: { },
  provide: {
    grid: [Group]
  }
}
</script>
<style>
  @import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/grid/group/default-cs9" %}

### Lazy load grouping in Vue Grid component

The lazy load grouping allows you to load grouped records to the Grid through the on-demand concept. So, you can use this feature to load a huge amount of grouped data to the Grid without any performance degradation.

When you enable this feature, the Grid will render only the initial level caption rows in the collapsed state at grouping. The child rows of each caption will be fetched from the server and render in the Grid when you expand the caption row. The fetching child records count will be implicitly determined by the content area occupying rows count. So, if the child records exceed the count, then the Grid will request the server again to fetch the next block of child records on scrolling.

The caption row expand/collapse state will be persisted on paging and Grid pages count will be determined based on the caption rows count instead of the child rows.

To enable this feature, you have to set the [groupSettings.enableLazyLoading](#) property as **true**.

### APP.VUE

```

<template>
  <div id="app">
    <ejs-grid ref='grid' :dataSource="data" :allowPaging='true'
    :allowGrouping='true' :groupSettings='groupSettings'>
      <e-columns>
        <e-column field='OrderID' headerText='Order ID' textAlign=
        'Right' width='120'></e-column>
        <e-column field='ProductName' headerText='Product Name'
        width='160'></e-column>
        <e-column field='ProductID' headerText='Product ID'
        textAlign= 'Right' width='120'></e-column>
        <e-column field='CustomerID' headerText='Customer ID'
        width='120'></e-column>
        <e-column field='CustomerName' headerText='Customer Name'
        width='160'></e-column>
      </e-columns>
    </div>
  </template>

```



```

        </ejs-grid>
    </div>
</template>
<script>
import Vue from "vue";
import { GridPlugin, Page, Group, LazyLoadGroup } from "@syncfusion/ej2-vue-grids";
import { data } from './datasource.js';
Vue.use(GridPlugin);
export default {
    data() {
        return {
            data: data,
            groupSettings: { enableLazyLoading: true, columns: ['ProductName', 'CustomerName'] }
        };
    },
    methods: { },
    provide: {
        grid: [Page, Group, LazyLoadGroup]
    }
}
</script>
<style>
    @import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/grid/group/lazy-load-group-cs1" %}

#### Handling the lazy load grouping at server-side

You can use the [UrlAdaptor](#) of **DataManager** when binding the remote data. Along with the default server request, this feature will additionally send the below details to handle the lazy load grouping. In the server end, these details are bound with the **ISLazyLoad** and **OnDemandGroupInfo** parameters in the **DataManagerRequest** model. Please refer to the below table and screenshots.

Property Name | Description

**isLazyLoad** | To differentiate the default grouping and lazy load grouping

**onDemandGroupInfo** | Have the details of expanded caption row grouping **level**, **skip**, **take** and **filter** query of the child records

```

.....} if (dm.IsLazyLoad)
.....{
.....    dm.IsLazyLoad = true;
.....    groupedData = operation.PerformGrouping<Customers>(DataSource, dm); // Lazy-load grouping
.....    if (dm.OnDemandGroupInfo != null && dm.Group.Count() == dm.OnDemandGroupInfo.Level)
.....    {
.....        count = groupedData.Cast<Customers>().Count();
.....    }
.....    else
.....    {
.....        count = groupedData.Cast<Group>().Count();
.....    }
.....    groupedData = operation.PerformSkip(groupedData, dm.OnDemandGroupInfo == null ? dm.Skip : dm.OnDemandGroupInfo.Skip);
.....    groupedData = operation.PerformTake(groupedData, dm.OnDemandGroupInfo == null ? dm.Take : dm.OnDemandGroupInfo.Take);
.....}
.....return dm.RequiresCounts ? Json(new { result = groupedData == null ? DataSource : groupedData, count = count }) : Json(DataSource);

```

```

if (dm.IsLazyLoad)
{
    groupedData = operation.PerformGrouping<Customers>(DataSource, dm); // Lazy load grouping
    if (dm.OnDemandGroupInfo != null && dm.Group.Count() == dm.OnDemandGroupInfo.Level)
    {
        dm.OnDemandGroupInfo = {Syncfusion.EJ2.Base.OnDemandGroupInfo}
        count = groupedData.Count();
    }
    else
    {
        count = groupedData.Count();
    }
    groupedData = operation.PerformSkip(groupedData, dm.OnDemandGroupInfo == null ? dm.Skip : dm.OnDemandGroupInfo.Skip);
    groupedData = operation.PerformTake(groupedData, dm.OnDemandGroupInfo == null ? dm.Take : dm.OnDemandGroupInfo.Take);
}
return dm.RequiresCounts ? Json(new { result = groupedData == null ? DataSource : groupedData, count = count }) : Json(DataSource);
}

```

The following code example describes the lazy load grouping handled at the server-side with other the grid actions.

```
`ts
```

```

public IActionResult UrlDatasource([FromBody] DataManagerRequest dm)
{
    IEnumerable groupedData = null;
    IEnumerable<Customers> DataSource = customers;
    DataOperations operation = new DataOperations();
    if (dm.Search != null && dm.Search.Count > 0)
    {
        DataSource = operation.PerformSearching(DataSource, dm.Search); //Search
    }
    if (dm.Sorted != null && dm.Sorted.Count > 0) //Sorting
    {
        DataSource = operation.PerformSorting(DataSource, dm.Sorted);
    }
    if (dm.Where != null && dm.Where.Count > 0) //Filtering
    {
        DataSource = operation.PerformFiltering(DataSource, dm.Where, dm.Where[0].Operator);
    }
    int count = DataSource.Cast<Customers>().Count();
    if (dm.IsLazyLoad == false && dm.Skip != 0)
    {
        DataSource = operation.PerformSkip(DataSource, dm.Skip); // Paging
    }
    if (dm.IsLazyLoad == false && dm.Take != 0)
    {
        DataSource = operation.PerformTake(DataSource, dm.Take);
    }
}

```

```

}
if (dm.IsLazyLoad)
{
groupedData = operation.PerformGrouping<Customers>(DataSource, dm); // Lazy load grouping
if (dm.OnDemandGroupInfo != null && dm.Group.Count() == dm.OnDemandGroupInfo.Level)
{
count = groupedData.Cast<Customers>().Count();
}
else
{
count = groupedData.Cast<Group>().Count();
}

groupedData = operation.PerformSkip(groupedData, dm.OnDemandGroupInfo == null ? dm.Skip :
dm.OnDemandGroupInfo.Skip);

groupedData = operation.PerformTake(groupedData, dm.OnDemandGroupInfo == null ? dm.Take :
dm.OnDemandGroupInfo.Take);
}

return dm.RequiresCounts ? Json(new { result = groupedData == null ? DataSource : groupedData, count
= count }) : Json(DataSource);
}
,

```

#### *Lazy load grouping with infinite scrolling*

Infinite scrolling loads a huge amount of data without degrading the Grid's performance. By default, infinite scrolling is enabled only for the expanded grouped rows when lazy loading is enabled. Now, the Grid has an option to allow infinite scrolling for group caption rows. This is achieved by setting the [enableInfiniteScrolling](#) property as true when lazy loading is enabled in the grouped records.

This is demonstrated in the following sample:

#### **APP.VUE**

```

<template>
  <div id="app">
    <ejs-grid ref='grid' :dataSource="data" :allowGrouping='true'
:enableInfiniteScrolling='true' :groupSettings='groupSettings'>
      <e-columns>
        <e-column field='OrderID' headerText='Order ID' textAlign=
'Right' width='120'></e-column>
        <e-column field='ProductName' headerText='Product Name'
width='160'></e-column>
        <e-column field='ProductID' headerText='Product ID'
textAlign= 'Right' width='120'></e-column>

```

```

        <e-column field='CustomerID' headerText='Customer ID'
width='120'></e-column>
        <e-column field='CustomerName' headerText='Customer Name'
width='160'></e-column>
    </e-columns>
</ejs-grid>
</div>
</template>
<script>
import Vue from "vue";
import { GridPlugin, Group, LazyLoadGroup, InfiniteScroll } from
"@syncfusion/ej2-vue-grids";
import { data } from './datasource.js';
Vue.use(GridPlugin);
export default {
  data() {
    return {
      data: data,
      groupSettings: { enableLazyLoading: true, columns: ['ProductName',
'CustomerName'] }
    };
  },
  methods: { },
  provide: {
    grid: [Group, LazyLoadGroup, InfiniteScroll]
  }
}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/grid/group/lazy-load-group-cs2" %}

The Grid also supports the lazy load grouping with [virtualization](#)(virtual scrolling).

#### *Limitations for lazy load grouping*

- Due to the element height limitation in browsers, the maximum number of records loaded by the grid is limited due to the browser capability.
- DataManager's [UrlAdaptor](#) and [JsonAdaptor](#) only have the built-in lazy load grouping support.
- Lazy load grouping is not compatible batch editing, row template etc.
- Programmatic selection is not supported.

## Filtering

### Filtering in Vue Grid component

Filtering allows you to view particular records based on filter criteria. To enable filtering in the Grid, set the [allowFiltering](#) to true. Filtering options can be configured through [filterSettings](#). To use filter, inject [Filter](#) module in the [Link to the Video](#) section.

To get start quickly with Filtering Options, you can check on this video:

<!-- The Grid supports three types of filter, they are

- Filter bar
- Excel
- Checkbox -->

**APP.VUE**

```

<template>
  <div id="app">
    <ejs-grid :dataSource='data' :allowFiltering='true' height='273px'>
      <e-columns>
        <e-column field='OrderID' headerText='Order ID'
textAlign='Right' width=100></e-column>
        <e-column field='CustomerID' headerText='Customer ID'
width=120></e-column>
        <e-column field='ShipCity' headerText='Ship City'
width=100></e-column>
        <e-column field='ShipName' headerText='Ship Name'
width=100></e-column>
      </e-columns>
    </ejs-grid>
  </div>
</template>
<script>
import Vue from "vue";
import { GridPlugin, Filter } from "@syncfusion/ej2-vue-grids";
import { data } from './datasource.js'
Vue.use(GridPlugin);
export default {
  data() {
    return {
      data: data
    };
  },
  provide: {
    grid: [Filter]
  }
}
</script>
<style>
  @import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/grid/filter/default-cs8" %}

\* You can apply and clear filtering, by using [filterByColumn](#) and [clearFiltering](#) methods.

\* To disable Filtering for a particular column, by specifying [columns.allowFiltering](#) to false.

*Initial filter*

To apply the filter at initial rendering, set the filter [predicate](#) object in [filterSettings.columns](#).

**APP.VUE**

```

<template>
  <div id="app">

```

```

    <ejs-grid :dataSource='data' :allowFiltering='true'
:filterSettings='filterOptions' height='273px'>
      <e-columns>
        <e-column field='OrderID' headerText='Order ID'
textAlign='Right' width=100></e-column>
        <e-column field='CustomerID' headerText='Customer ID'
width=120></e-column>
        <e-column field='ShipCity' headerText='Ship City'
width=100></e-column>
        <e-column field='ShipName' headerText='Ship Name'
width=100></e-column>
      </e-columns>
    </ejs-grid>
  </div>
</template>
<script>
import Vue from "vue";
import { GridPlugin, Filter } from "@syncfusion/ej2-vue-grids";
import { data } from './datasource.js'
Vue.use(GridPlugin);
export default {
  data() {
    return {
      data: data,
      filterOptions: {
        columns: [{ field: 'ShipCity', matchCase: false, operator:
'startswith', predicate: 'and', value: 'reims' },
        { field: 'ShipName', matchCase: false, operator: 'startswith',
predicate: 'and', value: 'Vins et alcools Chevalier' }]
      }
    };
  },
  provide: {
    grid: [Filter]
  }
}
</script>
<style>
  @import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/grid/filter/default-cs9" %}

### Filter operators

The filter operator for a column can be defined in [filterSettings.columns.operator](#) property.

The available operators and its supported data types are,

Operator | Description | Supported Types

a\*b | Everything that starts with "a" and ends with "b".

a\* | Everything that starts with "a".

\*b | Everything that ends with "b".

a | Everything that has an "a" in it.

*ab\** | Everything that has an "a" in it, followed by anything, followed by a "b", followed by anything.

Search			Columns	
Order ID	Shipped Date	Ship Country		
10250	7/12/1996	Brazil		
10251	7/15/1996	France		
10252	7/11/1996	Belgium		
10253	7/16/1996	Brazil		
10254	7/23/1996	Switzerland		
10255	7/15/1996	Switzerland		
10256	7/17/1996	Brazil		
10257	7/22/1996	Venezuela		
10258	7/23/1996	Austria		

#### LIKE filtering

The **LIKE** filter can process single search patterns using the "%" symbol, retrieving values matching the specified patterns. The following Grid features support LIKE filtering on string-type columns:

- Filter Menu
- Filter Bar with the [filterSettings.showFilterBarOperator](#) property enabled on the Grid [filterSettings](#).
- Custom Filter of Excel filter type.

#### For example:

Operator | Description

%ab% | Returns all the value that are contains "ab" character.

ab% | Returns all the value that are ends with "ab" character.

%ab | Returns all the value that are starts with "ab" character.

Order ID	Shipped Date	Ship Country
10250	7/12/1996	Brazil
10251	7/15/1996	France
10252	7/11/1996	Belgium
10253	7/16/1996	Brazil
10254	7/23/1996	Switzerland
10255	7/15/1996	Switzerland
10256	7/17/1996	Brazil
10257	7/22/1996	Venezuela
10258	7/23/1996	Austria

By default, the [filterSettings.columns.operator](#) value is **equal**.

#### *Diacritics filter*

By default, grid ignores diacritic characters while filtering. To include diacritic characters, set the [filterSettings.ignoreAccent](#) as **true**.

In the following sample, type **mun** in **Ship City** column to filter diacritic characters.

#### **APP.VUE**

```
<template>
  <div id="app">
    <ejs-grid :dataSource='data' :allowFiltering='true'
:filterSettings='filterOptions' height='273px'>
      <e-columns>
        <e-column field='OrderID' headerText='Order ID'
textAlign='Right' width=100></e-column>
        <e-column field='CustomerID' headerText='Customer ID'
width=120></e-column>
        <e-column field='ShipCity' headerText='Ship City'
width=100></e-column>
        <e-column field='ShipName' headerText='Ship Name'
width=100></e-column>
      </e-columns>
    </ejs-grid>
  </div>
</template>
```



```

<script>
import Vue from "vue";
import { GridPlugin, Filter } from "@syncfusion/ej2-vue-grids";
import { data } from './datasource.js'
Vue.use(GridPlugin);
export default {
  data() {
    return {
      data: data,
      filterOptions: {
        ignoreAccent: true
      }
    };
  },
  provide: {
    grid: [Filter]
  }
}
</script>
<style>
  @import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/grid/filter/default-cs10" %}

[See Also](#)

- [Hide sorting options on Excel filter Dialog](#)

### Filter bar in Vue Grid component

By defining the [allowFiltering](#) to true, then filter bar row will be rendered next to header which allows you to filter data. You can filter the records with different expressions depending upon the column type.

#### Filter bar Expressions:

You can enter the following filter expressions(operators) manually in the filter bar.

Expression	Example	Description	Column Type
=	=value	equal	Number
!=	!=value	notequal	Number
>	>value	greaterthan	Number
<	<value	lessthan	Number
= >=	>=value	greaterthanorequal	Number
<= <=	<=value	lessthanorequal	Number
/	/value	startswith	String
%	%value	endswith	String
N/A	N/A	Always <b>equal</b> operator will be used for Date filter	Date
N/A	N/A	Always <b>equal</b> operator will be used for Boolean filter	Boolean

**APP.VUE**

```

<template>
  <div id="app">
    <ejs-grid :dataSource='data' :allowFiltering='true' height='273px'>
      <e-columns>
        <e-column field='OrderID' headerText='Order ID'
textAlign='Right' width=100></e-column>
        <e-column field='CustomerID' headerText='Customer ID'
width=120></e-column>
        <e-column field='ShipCity' headerText='Ship City'
width=100></e-column>
        <e-column field='ShipName' headerText='Ship Name'
width=100></e-column>
      </e-columns>
    </ejs-grid>
  </div>
</template>
<script>
import Vue from "vue";
import { GridPlugin, Filter } from "@syncfusion/ej2-vue-grids";
import { data } from './datasource.js'
Vue.use(GridPlugin);
export default {
  data() {
    return {
      data: data
    };
  },
  provide: {
    grid: [Filter]
  }
}
</script>
<style>
  @import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/grid/filter/default-cs2" %}

*Filter bar template with custom component*

The [filterBarTemplate](#) is used to add a custom component for a particular column and this contains the following functions.

- **create** – It is used for creating custom components.
- **write** - It is used to wire events for custom components.

In the following sample dropdown is used as custom component in EmployeeID column.

**APP.VUE**

```

<template>
  <div id="app">
    <ejs-grid ref='grid' :dataSource='data' :allowFiltering='true'
height='273px'>
      <e-columns>

```

```

        <e-column field='OrderID' headerText='Order ID'
textAlign='Right' width=100></e-column>
        <e-column field='EmployeeID' headerText='Employee ID'
width=120 :filterBarTemplate='templateOptions'></e-column>
        <e-column field='ShipCity' headerText='Ship City'
width=100></e-column>
        <e-column field='ShipName' headerText='Ship Name'
width=100></e-column>
    </e-columns>
</ejs-grid>
</div>
</template>
<script>
import Vue from "vue";
import { GridPlugin, Filter } from "@syncfusion/ej2-vue-grids";
import { DropDownList } from '@syncfusion/ej2-dropdowns';
import { data } from './datasource.js'
Vue.use(GridPlugin);
export default {
    data() {
        return {
            data: data,
            templateOptions: {
                create: function (args) {
                    const dd = document.createElement('input');
                    dd.id = 'EmployeeID';
                    return dd;
                },
                write: function (args) {
                    let DropDownListObj: DropDownList = new DropDownList({
                        dataSource: ['All', '1', '3', '4', '5', '6', '8', '9'],
                        fields: { text: 'EmployeeID', value: 'EmployeeID' },
                        placeholder: 'Select a value',
                        popupHeight: '200px',
                        change: function (e) {
                            var gridObj = (document.getElementsByClassName('e-
grid')[0] as any).ej2_instances[0];
                            if (e.value == 'All') {
                                gridObj.removeFilteredColsByField('EmployeeID');
                            } else {
                                gridObj.filterByColumn('EmployeeID', 'equal', parseInt(e.value as any));
                            }
                        }
                    });
                    DropDownListObj.appendTo('#EmployeeID');
                }
            }
        };
    },
    provide: {
        grid: [Filter]
    }
}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";

```

```
</style>
```

```
{% previewsample "page.domainurl/code-snippet/grid/filter/default-cs3" %}
```

*See also*

- [How to perform filter by using Wildcard and LIKE operator filter](#)

### Filter menu in Vue Grid component

You can enable filter menu by setting the [filterSettings.type](#) as **Menu**. The filter menu UI will be rendered based on its column type, which allows you to filter data. You can filter the records with different operators.

### APP.VUE

```
<template>
  <div id="app">
    <ejs-grid :dataSource='data' :allowFiltering='true'
    :filterSettings='filterOptions' height='273px'>
      <e-columns>
        <e-column field='OrderID' headerText='Order ID'
        textAlign='Right' width=100></e-column>
        <e-column field='CustomerID' headerText='Customer ID'
        width=120></e-column>
        <e-column field='ShipCity' headerText='Ship City'
        width=100></e-column>
        <e-column field='ShipName' headerText='Ship Name'
        width=100></e-column>
      </e-columns>
    </ejs-grid>
  </div>
</template>
<script>
import Vue from "vue";
import { GridPlugin, Filter } from "@syncfusion/ej2-vue-grids";
import { data } from './datasource.js'
Vue.use(GridPlugin);
export default {
  data() {
    return {
      data: data,
      filterOptions: {
        type: 'Menu'
      }
    };
  },
  provide: {
    grid: [Filter]
  }
}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
</style>
```

```
{% previewsample "page.domainurl/code-snippet/grid/filter/default-cs4" %}
```

\* [allowFiltering](#) must be set as true to enable filter menu.

\* Setting [columns.allowFiltering](#) as false will prevent filter menu rendering for a particular column.

#### *Custom component in filter menu*

The [column.filter.ui](#) is used to add custom filter components to a particular column. To implement custom filter ui, define the following functions:

- **create:** Creates custom component.
- **write:** Wire events for custom component.
- **read:** Read the filter value from custom component.

In the following sample, dropdown is used as custom component in the OrderID column.

#### **APP.VUE**

```
<template>
  <div id="app">
    <ejs-grid :dataSource='data' :allowFiltering='true'
:filterSettings='filterOptions' height='273px'>
      <e-columns>
        <e-column field='OrderID' :filter= 'filter'
headerText='Order ID' textAlign='Right' width=100></e-column>
        <e-column field='CustomerID' headerText='Customer ID'
width=120></e-column>
        <e-column field='ShipCity' headerText='Ship City'
width=100></e-column>
        <e-column field='ShipName' headerText='Ship Name'
width=100></e-column>
      </e-columns>
    </ejs-grid>
  </div>
</template>
<script>
import Vue from "vue";
import { GridPlugin, Filter } from "@syncfusion/ej2-vue-grids";
import { data } from './datasource.js';
import { DropDownList } from "@syncfusion/ej2-dropdowns";
import { DataManager } from "@syncfusion/ej2-data";
import { createElement } from "@syncfusion/ej2-base";
Vue.use(GridPlugin);
export default {
  data() {
    let dropInstance = null;
    return {
      data: data,
      filterOptions: {
        type: 'Menu'
      },
      filter: {
        ui: {
          create: function (args) {
            let db = new DataManager(data);
```

```

        let flValInput = createElement('input', { className:
'flm-input' });
        args.target.appendChild(flValInput);
        dropInstance = new DropDownList({
        dataSource: new DataManager(data),
        fields: { text: 'OrderID', value: 'OrderID' },
        placeholder: 'Select a value',
        popupHeight: '200px'
        });
        dropInstance.appendTo(flValInput);
    },
    write: function (args) {
        dropInstance.value = args.filteredValue;
    },
    read: function (args) {
        args.fltrObj.filterByColumn(args.column.field,
args.operator, dropInstance.value);
    }
    }
    };
},
provide: {
    grid: [Filter]
}
}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/grid/filter/default-cs5" %}

### [Customizing filter menu operators list](#)

You can customize the default filter operator list by defining the [filterSettings.operators](#) property. The available options are:

- **stringOperator** - defines customized string operator list.
- **numberOperator** - defines customized number operator list.
- **dateOperator** - defines customized date operator list.
- **booleanOperator** - defines customized boolean operator list.

In the following sample, we have customized string filter operators.

### **APP.VUE**

```

<template>
  <div id="app">
    <ejs-grid ref='grid' :dataSource='data' allowFiltering='true'
:filterSettings='filterOptions' height='273px' >
      <e-columns>
        <e-column field='OrderID' headerText='Order ID'
textAlign='Right' width=100></e-column>

```

```

        <e-column field='CustomerID' headerText='Customer ID'
width=120></e-column>
        <e-column field='ShipCity' headerText='Ship City'
width=100></e-column>
        <e-column field='ShipName' headerText='Ship Name'
width=100></e-column>
    </e-columns>
</ejs-grid>
</div>
</template>
<script>
import Vue from "vue";
import { GridPlugin, Filter } from "@syncfusion/ej2-vue-grids";
import { data } from './datasource.js';
Vue.use(GridPlugin);
export default {
  data: () => {
    return {
      data: data,
      filterOptions: {
        type: 'Menu',
        operators: {
          stringOperator: [
            { value: 'startsWith', text: 'starts with' },
            { value: 'endsWith', text: 'ends with' },
            { value: 'contains', text: 'contains' }
          ]
        }
      }
    };
  },
  provide: {
    grid: [Filter],
  },
}
</script>
<style>
  @import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/grid/how-to/default-cs10" %}

#### Enable different filter for a column

You can use both **Menu** and **Checkbox** filter in a same Grid. To do so, set the [column.filter.type](#) as **Menu** or **Checkbox**.

In the following sample menu filter is enabled by default and checkbox filter is enabled for the CustomerID column using the [column.filter.type](#).

#### APP.VUE

```

<template>
  <div id="app">
    <ejs-grid :dataSource='data' :allowFiltering='true'
:filterSettings='filterOptions' height='273px'>
      <e-columns>

```

```

        <e-column field='OrderID' headerText='Order ID'
textAlign='Right' width=100></e-column>
        <e-column field='CustomerID' :filter='filter'
headerText='Customer ID' width=120></e-column>
        <e-column field='ShipCity' headerText='Ship City'
width=100></e-column>
        <e-column field='ShipName' headerText='Ship Name'
width=100></e-column>
    </e-columns>
</ejs-grid>
</div>
</template>
<script>
import Vue from "vue";
import { GridPlugin, Filter } from "@syncfusion/ej2-vue-grids";
import { data } from './datasource.js'
Vue.use(GridPlugin);
export default {
  data() {
    return {
      data: data,
      filterOptions: {
        type: 'Menu'
      },
      filter: {
        type : 'CheckBox'
      }
    };
  },
  provide: {
    grid: [Filter]
  }
}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/grid/filter/default-cs6" %}

#### *Filter by multiple keywords using filter menu*

By default, the filtering action is performed based on the single keyword filter value from the built-in component of the filter menu dialog. Now data grid has an option to perform filtering actions based on multiple keywords instead of a single keyword alone. For this, set [filterSettings.type](#) as `Menu`.

In the following sample, filtering action with multiple keywords can be done by rendering the `MultiSelect` component as custom component in the OrderID column filter menu dialog.

#### **APP.VUE**

```

<template>
  <div id="app">
    <ejs-grid ref='grid' :dataSource='data' :allowFiltering='true'
:filterSettings='filterSettings' height='273px'>
      <e-columns>

```



```

        <e-column field='OrderID' :filter= 'filter'
headerText='Order ID' textAlign='Right' width=100></e-column>
        <e-column field='CustomerID' headerText='Customer ID'
width=120></e-column>
        <e-column field='ShipCity' headerText='Ship City'
width=100></e-column>
        <e-column field='ShipName' headerText='Ship Name'
width=100></e-column>
    </e-columns>
</ejs-grid>
</div>
</template>
<script>
import Vue from "vue";
import { GridPlugin, Filter } from "@syncfusion/ej2-vue-grids";
import { data } from './datasource.js';
import { MultiSelect, CheckBoxSelection } from "@syncfusion/ej2-dropdowns";
import { DataManager } from "@syncfusion/ej2-data";
import { createElement } from "@syncfusion/ej2-base";
import { MultiSelectPlugin } from "@syncfusion/ej2-vue-dropdowns";
MultiSelect.Inject(CheckBoxSelection);
Vue.use(MultiSelectPlugin);
Vue.use(GridPlugin);
export default {
    data() {
        let dropInstance = null;
        return {
            data: data,
            filterSettings: {
                type: 'Menu'
            },
            filter: {
                ui: {
                    create: function (args) {
                        let db = new DataManager(data);
                        let flValInput = createElement('input', { className:
'flm-input' });
                        args.target.appendChild(flValInput);
                        dropInstance = new MultiSelect({
                            dataSource: new DataManager(data),
                            fields: { text: 'OrderID', value: 'OrderID' },
                            placeholder: 'Select a value',
                            popupHeight: '200px',
                            allowFiltering: true,
                            mode: "CheckBox",
                        });
                        dropInstance.appendTo(flValInput);
                    },
                    write: function (args) {
                        var grid =
document.getElementById("Grid").ej2_instances[0];
                        var filteredValue = [];
                        grid.filterSettings.columns.map((item) => {
                            if (item.field === "OrderID" && item.value) {
                                filteredValue.push(item.value);
                            }
                        });
                    }
                }
            }
        };
    }
};

```

```

        if (filteredValue.length > 0) {
            dropInstance.value = filteredValue;
        }
    },
    read: function (args) {
        var grid =
document.getElementById("Grid").ej2_instances[0];
        grid.removeFilteredColsByField(args.column.field);
        args.fltrObj.filterByColumn(
            args.column.field,
            "contains",
            dropInstance.value
        );
    }
}
};
},
provide: {
    grid: [Filter, CheckBoxSelection]
}
}
</script>
<style>
    @import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
</style>

```

```
{% previewsample "page.domainurl/code-snippet/grid/filter/default-cs7" %}
```

#### [Add current selection to filter checkbox](#)

By default, the CheckBox filter can only filter the selected items. If filtering is done multiple times on the same column, the previously filtered values in the column will be cleared. Now, it is possible to retain those previous values by using the **Add current selection to filter** checkbox. This checkbox is displayed when data is searched in the search bar of the CheckBox filter.

The following image describes the above mentioned behavior.

Order ID	Customer Name	Order Date	Freight
10248		4/1/1996 10:10 AM	\$32.38
10249		5/1/1996 12:20 PM	\$11.61
10250		3/1/1996 08:40 AM	\$65.83
10251		3/1/1996 07:50 AM	\$41.34
10252		9/1/1996 12:05 PM	\$51.30
10253		10/1/1996 11:20 AM	\$58.17
10254		1/1/1996 10:00 AM	\$22.98
10255		2/1/1996 10:40 AM	\$148.33
10256		5/1/1996 08:50 PM	\$13.97
10257	Carlos Hernández	7/16/1996 03:20 AM	\$81.91
10258	Roland Mendel	7/17/1996 06:30 PM	\$140.51
10259	Francisco Chang	7/18/1996 01:20 AM	\$3.25

See also

- [How to perform filter by using Wildcard and LIKE operator filter](#)

### Excel like filter in Vue Grid component

You can enable Excel like filter by defining. [filterSettings.type](#) as Excel. The excel menu contains an option such as Sorting, Clear filter, Sub menu for advanced filtering.

### APP.VUE

```
<template>
  <div id="app">
    <ejs-grid ref="Grid" :dataSource="data" :allowFiltering="true"
    :filterSettings="filterOptions" height="273px">
      <e-columns>
        <e-column field="OrderID" headerText="ID" width="80"
isPrimaryKey="true"></e-column>
        <e-column field="CustomerID" headerText="CustomerID" width="90"></e-
column>
        <e-column field="OrderDate" headerText="OrderDate" width="120"
format="yMd"></e-column>
        <e-column field="ShipName" headerText="ShipName" width="120"></e-
column>
        <e-column field="ShipCity" headerText="ShipCity" width="120"></e-
column>
      </e-columns>
    </ejs-grid>
  </div>
</template>
<script>
import Vue from "vue";
```

```

import { GridPlugin, Filter } from "@syncfusion/ej2-vue-grids";
import { data } from './datasource.js'
Vue.use(GridPlugin);
export default {
  data() {
    return {
      data: data,
      filterOptions: { type: "Excel" },
    };
  },
  methods: {
  },
  provide: {
    grid: [Filter],
  },
};
</script>
<style>
@import "https://cdn.syncfusion.com/ej2/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/grid/filter/default-cs1" %}

\* By default, while opening the excel/checkbox filter in the Grid, the filter dialog will get and display the distinct data from the first 1000 records bound to the Grid to optimize performance. The remaining records will be returned as a result of the search option of the filter dialog.

\* However, we can increase the excel/checkbox filter count by modifying the `filterChoiceCount` argument value(as per our need) in the `actionBegin` event when the `requestType` is `filterchoicerequest` or `filtersearchbegin`. This is demonstrated in the below code snippet,

```

`ts
methods: {
  actionBegin(args) {
    if (args.requestType === "filterchoicerequest" || args.requestType === "filtersearchbegin" ) {
      args.filterChoiceCount = 3000;
    }
  },
},
,
`

```

#### *Render checkbox list data in on-demand for excel/checkbox filtering*

The Excel/Checkbox filter type of Grid has a restriction where only the first 1000 unique sorted items are accessible to view in the filter dialog checkbox list content by scrolling. This limitation is in place to avoid any rendering delays when opening the filter dialog. However, the searching and filtering processes consider all unique items in that particular column.

The Excel/Checkbox filter in the Grid provides an option to load large data sets on-demand during scrolling to improve scrolling limitation functionality. This is achieved by setting the

[filterSettings.enableInfiniteScrolling](#) property to **true**. This feature proves especially beneficial for managing extensive datasets, enhancing data loading performance in the checkbox list, and allowing interactive checkbox selection with persistence for the selection based on filtering criteria.

The Excel/Checkbox filter retrieves distinct data in ascending order, governed by its [filterSettings.itemsCount](#) property, with a default value of **50**. As the checkbox list data scroller reaches its end, the next dataset is fetched and displayed, with the notable advantage that this process only requests new checkbox list data without redundantly fetching the existing loaded dataset.

#### Customize the items count for initial rendering

Based on the items count value, the Excel/Checkbox filter gets unique data and displayed in Excel/Checkbox filter content dialog. You can customize the count of on-demand data rendering for Excel/Checkbox filter by adjusting the [filterSettings.itemsCount](#) property. The default value is **50**.

`ts

```
grid.filterSettings = { enableInfiniteScrolling = true, itemsCount = 40 };
```

,

It is recommended to keep the itemsCount below **300**. Higher values will result in unwanted whitespace because of DOM maintenance performance degradation.

#### Customize the loading animation effect

A loading effect is presented to signify that loading is in progress when the checkbox list data scroller reaches the end, and there is a delay in receiving the data response from the server. The loading effect during on-demand data retrieval for Excel/Checkbox filter can be customized using the [filterSettings.loadingIndicator](#) property. The default value is **Shimmer**.

`ts

```
grid.filterSettings = { enableInfiniteScrolling = true, loadingIndicator = 'Spinner' };
```

,

In the provided example, On-Demand Excel filter has been enabled for the Vue Grid

#### APP.VUE

```
<template>
  <div id="app">
    <ejs-grid ref='grid' :dataSource="getTradeData" :query="query"
:allowPaging='true' :allowFiltering='true'
    :allowSorting='true' :pageSettings='pageSettings'
:filterSettings='filterSettings'>
      <e-columns>
        <e-column field='EmployeeID' headerText='Employee ID' width='120'
textAlign='Right'></e-column>
        <e-column field='Employees' headerText='Employee Name'
width='150'></e-column>
        <e-column field='Designation' headerText='Designation' width='130'
textAlign='Right'></e-column>
        <e-column field='CurrentSalary' headerText='CurrentSalary'
width='120' format='C2' textAlign='Right'></e-column>
      </e-columns>
    </ejs-grid>
  </div>
</template>
```

```

<script>
import Vue from "vue";
import { GridPlugin, Filter, Page, Sort } from "@syncfusion/ej2-vue-grids";
import { DataManager, Query, UrlAdaptor } from "@syncfusion/ej2-data";
Vue.use(GridPlugin);
export default {
  data() {
    return {
      pageSettings: { pageCount: 5 },
      filterSettings: { type: "Excel", enableInfiniteScrolling: true },
    };
  },
  methods: {
  },
  computed: {
    getTradeData: function () {
      let SERVICE_URI = "https://services.syncfusion.com/vue/production/";
      let getTradeData = new DataManager({
        url: SERVICE_URI + 'api/UrlDataSource',
        adaptor: new UrlAdaptor()
      });
      return getTradeData;
    },
    query: function () {
      let query = new Query().addParams('dataCount', '10000');
      return query;
    }
  },
  provide: {
    grid: [Filter, Page, Sort],
  },
};
</script>

```

{% previewsample "page.domainurl/code-snippet/grid/checkbox-excel-filter" %}

[See also](#)

- [How to perform filter by using Wildcard and LIKE operator filter](#)

## Searching in Vue Grid component

You can search records in a Grid, by using the [search](#) method with search key as a parameter. This also provides an option to integrate search text box in grid's toolbar by adding **Search** item to the [toolbar](#).

To use searching, you need to inject [Search](#) module in the **provide** section.

The clear icon is shown in the Data Grid search text box when it is focused on search text or after typing the single character in the search text box. A single click of the clear icon clears the text in the search box as well as the search results in the Data Grid.

## APP.VUE

```

<template>
  <div id="app">

```

```

        <ejs-grid :dataSource='data' :toolbar='toolbarOptions'
height='272px'>
            <e-columns>
                <e-column field='OrderID' headerText='Order ID'
textAlign='Right' width=100></e-column>
                <e-column field='CustomerID' headerText='Customer ID'
width=120></e-column>
                <e-column field='ShipCity' headerText='Ship City'
width=100></e-column>
                <e-column field='ShipName' headerText='Ship Name'
width=100></e-column>
            </e-columns>
        </ejs-grid>
    </div>
</template>
<script>
import Vue from "vue";
import { GridPlugin, Toolbar, Search } from "@syncfusion/ej2-vue-grids";
import { data } from './datasource.js'
Vue.use(GridPlugin);
export default {
    data() {
        return {
            data: data,
            toolbarOptions: ['Search']
        };
    },
    provide: {
        grid: [Toolbar, Search]
    }
}
</script>
<style>
@import
"https://ej2.syncfusion.com/vue/documentation/node_modules/@syncfusion/ej2-
vue-grids/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/grid/search/default-cs3" %}

### Initial search

To apply search at initial rendering, set the fields, operator, key, and ignoreCase in the [searchSettings](#).

### APP.VUE

```

<template>
    <div id="app">
        <ejs-grid :dataSource='data' :searchSettings='searchOptions'
:toolbar='toolbarOptions' height='272px'>
            <e-columns>
                <e-column field='OrderID' headerText='Order ID'
textAlign='Right' width=100></e-column>
                <e-column field='CustomerID' headerText='Customer ID'
width=120></e-column>
                <e-column field='ShipCity' headerText='Ship City'
width=100></e-column>
            </e-columns>
        </ejs-grid>
    </div>
</template>

```

```

        <e-column field='ShipName' headerText='Ship Name'
width=100></e-column>
    </e-columns>
</ejs-grid>
</div>
</template>
<script>
import Vue from "vue";
import { GridPlugin, Toolbar, Search } from "@syncfusion/ej2-vue-grids";
import { data } from './datasource.js'
Vue.use(GridPlugin);
export default {
  data() {
    return {
      data: data,
      toolbarOptions: ['Search'],
      searchOptions: { fields: ['CustomerID'], operator: 'contains', key:
'Ha', ignoreCase: true }
    };
  },
  provide: {
    grid: [Toolbar, Search]
  }
}
</script>
<style>
@import
"https://ej2.syncfusion.com/vue/documentation/node_modules/@syncfusion/ej2-
vue-grids/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/grid/search/default-cs4" %}

By default, grid searches all the bound column values. To customize this behavior define the [searchSettings.fields](#) property.

### Search operators

The search operator can be defined in [searchSettings.operator](#) property to configure specified searching.

The following operators are supported in searching:

#### Operator | Description

startsWith | Checks whether a value begins with the specified value.

endsWith | Checks whether a value ends with specified value.

contains | Checks whether a value contains with specified value.

equal | Checks whether a value equal to specified value.

notEqual | Checks whether a value not equal to specified value.

### Search by external button

To search grid records from an external button, invoke the [search](#) method.

### **APP.VUE**



```

<template>
  <div id="app">
    <div class="e-float-input" style="width: 200px; display: inline-block;">
      <input type="text" class="searchtext"/>
      <span class="e-float-line"></span>
      <label class="e-float-text">Search text</label>
    </div>
    <ejs-button id='search' @click.native='search'>Search</ejs-button>
    <ejs-grid ref='grid' :dataSource='data' height='262px'>
      <e-columns>
        <e-column field='OrderID' headerText='Order ID'
textAlign='Right' width=100></e-column>
        <e-column field='CustomerID' headerText='Customer ID'
width=120></e-column>
        <e-column field='ShipCity' headerText='Ship City'
width=100></e-column>
        <e-column field='ShipName' headerText='Ship Name'
width=100></e-column>
      </e-columns>
    </ejs-grid>
  </div>
</template>
<script>
import Vue from "vue";
import { GridPlugin, Toolbar, Search } from "@syncfusion/ej2-vue-grids";
import { ButtonPlugin } from '@syncfusion/ej2-vue-buttons';
import { data } from './datasource.js'
Vue.use(GridPlugin);
Vue.use(ButtonPlugin);
export default {
  data() {
    return {
      data: data
    };
  },
  methods: {
    search: function() {
      let searchText =
document.getElementsByClassName('searchtext')[0].value;
      this.$refs.grid.search(searchText);
    }
  },
  provide: {
    grid: [Search]
  }
}
</script>
<style>
@import
"https://ej2.syncfusion.com/vue/documentation/node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/grid/search/default-cs5" %}

### Search Specific Columns

By default, grid searches all visible columns. You can search specific columns by defining the specific column's field names in the [searchSettings.fields](#) property.

#### APP.VUE

```
<template>
  <div id="app">
    <ejs-grid :dataSource='data' :searchSettings='searchOptions'
:toolbar='toolbarOptions' height='272px'>
      <e-columns>
        <e-column field='OrderID' headerText='Order ID'
textAlign='Right' width=100></e-column>
        <e-column field='CustomerID' headerText='Customer ID'
width=120></e-column>
        <e-column field='ShipCity' headerText='Ship City'
width=100></e-column>
        <e-column field='ShipName' headerText='Ship Name'
width=100></e-column>
      </e-columns>
    </ejs-grid>
  </div>
</template>
<script>
import Vue from "vue";
import { GridPlugin, Toolbar, Search } from "@syncfusion/ej2-vue-grids";
import { data } from './datasource.js'
Vue.use(GridPlugin);
export default {
  data() {
    return {
      data: data,
      toolbarOptions: ['Search'],
      searchOptions: {fields: ['CustomerID', 'ShipCity']}
    };
  },
  provide: {
    grid: [Toolbar, Search]
  }
}
</script>
<style>
@import
"https://ej2.syncfusion.com/vue/documentation/node_modules/@syncfusion/ej2-
vue-grids/styles/material.css";
</style>
```

{% previewsample "page.domainurl/code-snippet/grid/search/default-cs6" %}

### Clear search by external button

To clear the searched grid records from the external button, set [searchSettings.key](#) property as **empty** string.

#### APP.VUE

```
<template>
```

```

    <div id="app">
      <ejs-button id='clear' @click.native='clear'>Clear Search</ejs-
button>
      <ejs-grid ref='grid' :dataSource='data'
:searchSettings='searchOptions' :toolbar='toolbarOptions' height='262px'>
        <e-columns>
          <e-column field='OrderID' headerText='Order ID'
textAlign='Right' width=100></e-column>
          <e-column field='CustomerID' headerText='Customer ID'
width=120></e-column>
          <e-column field='ShipCity' headerText='Ship City'
width=100></e-column>
          <e-column field='ShipName' headerText='Ship Name'
width=100></e-column>
        </e-columns>
      </ejs-grid>
    </div>
  </template>
  <script>
import Vue from "vue";
import { GridPlugin, Toolbar, Search } from "@syncfusion/ej2-vue-grids";
import { ButtonPlugin } from '@syncfusion/ej2-vue-buttons';
import { data } from './datasource.js'
Vue.use(GridPlugin);
Vue.use(ButtonPlugin);
export default {
  data() {
    return {
      data: data,
      toolbarOptions: ['Search'],
      searchOptions: { fields: ['CustomerID'], operator: 'contains', key:
'Ha', ignoreCase: true }
    };
  },
  methods: {
    clear: function() {
      this.$refs.grid.ej2Instances.searchSettings.key = "";
    }
  },
  provide: {
    grid: [Search,Toolbar]
  }
}
  </script>
  <style>
    @import
    "https://ej2.syncfusion.com/vue/documentation/node_modules/@syncfusion/ej2-
vue-grids/styles/material.css";
  </style>

```

{% previewsample "page.domainurl/code-snippet/grid/search/default-cs7" %}

### Search on each key stroke

You can search the Grid data on each key stroke by binding the `keyup` event for the search input element inside the `created` event. Inside the `keyup` handler you can search the Grid by invoking the `search` method of the Grid component.

#### APP.VUE

```
<template>
  <div id="app">
    <ejs-grid ref='grid' id='grid' :dataSource='data'
:toolbar='toolbarOptions' height='262px' :created='created'>
      <e-columns>
        <e-column field='OrderID' headerText='Order ID'
textAlign='Right' width=100></e-column>
        <e-column field='CustomerID' headerText='Customer ID'
width=120></e-column>
        <e-column field='ShipCity' headerText='Ship City'
width=100></e-column>
        <e-column field='ShipName' headerText='Ship Name'
width=100></e-column>
      </e-columns>
    </ejs-grid>
  </div>
</template>
<script>
import Vue from "vue";
import { GridPlugin, Toolbar, Search } from "@syncfusion/ej2-vue-grids";
import { data } from './datasource.js'
Vue.use(GridPlugin);
export default {
  data() {
    return {
      data: data,
      toolbarOptions: ['Search'],
    };
  },
  methods: {
    created: function() {
      document.getElementById(this.$refs.grid.$el.id +
"_searchbar").addEventListener('keyup', () => {
        this.$refs.grid.search((event.target as HTMLInputElement).value)
      });
    }
  },
  provide: {
    grid: [Search, Toolbar]
  }
}
</script>
<style>
@import
"https://ej2.syncfusion.com/vue/documentation/node_modules/@syncfusion/ej2-
vue-grids/styles/material.css";
</style>
```

```
{% previewsample "page.domainurl/code-snippet/grid/search/default-cs8" %}
```

### Perform search operation in Grid using multiple keywords

You can perform a searching operation in the Grid using multiple keywords. This can be achieved by the [actionBegin](#) event of the Grid.

In the following sample, we have performed the searching with multiple keywords by using the query property of grid when the requestType is searching in the [actionBegin](#) event.

#### APP.VUE

```
<template>
  <div id="app">
    <ejs-grid ref='grid' :dataSource='data'
      :searchSettings="searchOptions" :toolbar="toolbarOptions" height='273px'
      :actionBegin = 'actionBegin' :actionComplete = 'actionComplete'>
      <e-columns>
        <e-column field='OrderID' headerText='Order ID'
          textAlign='Right' :isPrimaryKey='true' width=100></e-column>
        <e-column field='CustomerID' headerText='Customer ID'
          width=120></e-column>
        <e-column field='Freight' headerText='Freight' textAlign=
          'Right' editType= 'numericedit' width=120 format= 'C2'></e-column>
        <e-column field='ShipCountry' headerText='Ship Country'
          editType= 'dropdownedit' width=150></e-column>
      </e-columns>
    </ejs-grid>
  </div>
</template>
<script>
import Vue from "vue";
import { GridPlugin, Toolbar, Search } from "@syncfusion/ej2-vue-grids";
import { data } from './datasource.js';
import { Predicate, Query } from "@syncfusion/ej2-data";
Vue.use(GridPlugin);
export default {
  data() {
    return {
      values: "",
      key: "",
      refresh: false,
      removeQuery: false,
      valueAssign: false,
      data: data,
      toolbarOptions: ["Search"],
      searchOptions: {
        fields: ["OrderID", "CustomerID", "ShipCity", "ShipName"],
        key: "",
      },
    };
  },
  methods: {
    actionBegin(args) {
      if (args.requestType === "searching") {
        const keys = args.searchString.split(",");
        var flag = true;
        var predicate;
```

```

        if (keys.length > 1) {
            if (this.$refs.grid.ej2Instances.searchSettings.key !== "") {
                this.values = args.searchString;
                keys.forEach((key) => {
                    this.$refs.grid.ej2Instances.getColumns().forEach((col) => {
                        if (flag) {
                            predicate = new Predicate(col.field, "contains", key,
true);
                            flag = false;
                        } else {
                            var predic = new Predicate(col.field, "contains", key,
true);
                            predicate = predicate.or(predic);
                        }
                    });
                });
                this.$refs.grid.ej2Instances.query = new
Query().where(predicate);
                this.$refs.grid.ej2Instances.searchSettings.key = "";
                this.refresh = true;
                this.valueAssign = true;
                this.removeQuery = true;
                this.$refs.grid.ej2Instances.refresh();
            }
        }
    },
    actionComplete(args) {
        if (args.requestType === "refresh" && this.valueAssign) {
            document.getElementById(
                this.$refs.grid.ej2Instances.element.id + "_searchbar"
            ).value = this.values;
            this.valueAssign = false;
        } else if (
            document.getElementById(
                this.$refs.grid.ej2Instances.element.id + "_searchbar"
            ).value === "" &&
            args.requestType === "refresh" &&
            this.removeQuery
        ) {
            this.$refs.grid.ej2Instances.query = new Query();
            this.removeQuery = false;
            this.$refs.grid.ej2Instances.refresh();
        }
    },
    provide: {
        grid: [Toolbar, Search]
    }
}
</script>
<style>
    @import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/grid/search/default-cs9" %}

See also

- [How to perform search by using Wildcard and LIKE operator filter](#)

## Paging in Vue Grid component

Paging provides an option to display Grid data in page segments. To enable paging, set the [allowPaging](#) to true. When paging is enabled, pager component renders at the bottom of the grid. Paging options can be configured through the [pageSettings](#).

In the below sample, [pageSize](#) is calculated based on the grid height by using the [load](#) event.

To use Paging, you need to inject [Page](#) module in the **provide** section.

### APP.VUE

```
<template>
  <div id="app">
    <ejs-grid ref='grid' :dataSource="data" :allowPaging="true"
    :pageSettings='pageSettings' height=323 :load='load'>
      <e-columns>
        <e-column field='OrderID' headerText='Order ID'
        textAlign='Right' width=90></e-column>
        <e-column field='CustomerID' headerText='Customer ID'
        width=120></e-column>
        <e-column field='Freight' headerText='Freight' textAlign='Right'
        format='C2' width=90></e-column>
        <e-column field='ShipCity' headerText='Ship City' width=150></e-
        column>
      </e-columns>
    </ejs-grid>
  </div>
</template>
<script>
import Vue from "vue";
import { GridPlugin, Page } from "@syncfusion/ej2-vue-grids";
import { data } from "../datasource.js";
Vue.use(GridPlugin);
export default {
  data() {
    return {
      data: data,
      pageSettings: { pageSize: 5 }
    };
  },
  methods: {
    load: function() {
      let rowHeight = this.$refs.grid.ej2Instances.getRowHeight(); //height
of the each row
      let gridHeight = this.$refs.grid.height; //grid height
      let pageSize = this.$refs.grid.pageSettings.pageSize; //initial page
size
      let pageResize = (gridHeight - (pageSize * rowHeight)) / rowHeight;
//new page size is obtained here
      this.$refs.grid.pageSettings = {pageSize: pageSize +
      Math.round(pageResize)};
    }
  }
}
```

```

    },
    provide: {
      grid: [Page]
    }
  }
</script>
<style>
  @import
  "https://ej2.syncfusion.com/vue/documentation/node_modules/@syncfusion/ej2-
  vue-grids/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/grid/page/default-cs1" %}

You can achieve better performance by using grid paging to fetch only a pre-defined number of records from the data source.

### Template

You can use custom elements inside the pager instead of default elements. The custom elements can be defined by using [pagerTemplate](#). Inside this template, you can access the [currentPage](#), [pageSize](#), [pageCount](#), [totalPage](#) and [totalRecordCount](#) values.

### APP.VUE

```

<template>
  <div id="app">
    <ejs-grid ref="grid" :dataSource="data" :allowPaging="true"
    :pageSettings='pageSettings' :pagerTemplate='pagerTemp'>
      <e-columns>
        <e-column field='OrderID' headerText='Order ID'
        textAlign='Right' width=90></e-column>
        <e-column field='CustomerID' headerText='Customer ID'
        width=120></e-column>
        <e-column field='Freight' headerText='Freight' textAlign='Right'
        format='C2' width=90></e-column>
        <e-column field='ShipCity' headerText='Ship City' width=150></e-
        column>
      </e-columns>
    </ejs-grid>
  </div>
</template>
<script>
import Vue from "vue";
import { GridPlugin, Page } from "@syncfusion/ej2-vue-grids";
import { NumericTextBoxPlugin } from "@syncfusion/ej2-vue-inputs";
import { data } from "../datasource.js";
Vue.use(GridPlugin);
Vue.use(NumericTextBoxPlugin);
export default {
  data() {
    return {
      data: data,
      pageSettings: { pageSize: 5 },
      pagerTemp: function() {
        return {
          template: Vue.component("pagerTemplate", {

```



```

template: `<div class="e-pagertemplate">
  <div class="col-lg-12 control-section">
    <div class="content-wrapper">
      <ejs-numericinput width="100"
: value="value" : change="change"></ejs-numericinput>
    </div>
  </div>
  <div id="totalPages" class="e-pagertemplatemessage"
    style="margin-top: 5px; margin-left: 30px; border:
none; display: inline-block ">
    <span class="e-
pagenomsg">{{this.$data.data.currentPage}} of {{this.$data.data.totalPages}}
pages
    <span class="e-recordscount">{{this.$data.data.totalRecordsCount}}
items</span>
  </div>
</div>`,
computed: {
  value: function() {
    return this.$data.data.currentPage;
  },
  methods: {
    change: function(args) {
      let grid: any = this.$el.closest(".e-
grid").ej2_instances[0];
      grid.goToPage(args.value);
    }
  }
})
};
}
};
},
provide: {
  grid: [Page]
}
}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
.e-pagertemplate {
  display: inline-block;
  overflow: hidden;
}
.control-section {
  margin-left: 20px;
  width: 25%;
}
.content-wrapper {
  width: 25%;
  margin: 0 auto;
  min-width: 185px;
}
.e-float-input.e-numeric.e-input-group {
  margin-top: 12px;
  display: inline-flex;

```

```
width: 180px;
}
@media (max-width: 1120px) {
  .e-pager .content-wrapper {
    display: inline-block
  }
}
</style>
```

{% previewsample "page.domainurl/code-snippet/grid/page/template-cs1" %}

### Pager with Page Size Dropdown

The pager Dropdown allows you to change the number of records in the Grid dynamically. It can be enabled by defining the [pageSettings.pageSizes](#) property as true.

### APP.VUE

```
<template>
  <div id="app">
    <ejs-grid :dataSource="data" :allowPaging="true"
:pageSettings='pageSettings'>
      <e-columns>
        <e-column field='OrderID' headerText='Order ID'
textAlign='Right' width=90></e-column>
        <e-column field='CustomerID' headerText='Customer ID'
width=120></e-column>
        <e-column field='Freight' headerText='Freight' textAlign='Right'
format='C2' width=90></e-column>
        <e-column field='ShipCity' headerText='Ship City' width=150></e-
column>
      </e-columns>
    </ejs-grid>
  </div>
</template>
<script>
import Vue from "vue";
import { GridPlugin, Page } from "@syncfusion/ej2-vue-grids";
import { data } from "../datasource.js";
Vue.use(GridPlugin);
export default {
  data() {
    return {
      data: data,
      pageSettings: { pageSizes: true, pageSize: 9 }
    };
  },
  provide: {
    grid: [Page]
  }
}
</script>
<style>
@import
"https://ej2.syncfusion.com/vue/documentation/node_modules/@syncfusion/ej2-
vue-grids/styles/material.css";
</style>
```

```
{% previewsample "page.domainurl/code-snippet/grid/page/default-cs2" %}
```

### How to render Pager at the Top of the Grid

By default, Pager will be rendered at the bottom of the Grid. You can also render the Pager at the top of the Grid by using the [dataBound](#) event.

#### APP.VUE

```
<template>
  <div id="app">
    <ejs-grid ref='grid' :dataSource="data" :allowPaging="true"
:pageSettings='pageSettings' :toolbar='toolbar' :dataBound='dataBound'>
      <e-columns>
        <e-column field='OrderID' headerText='Order ID'
textAlign='Right' width=90></e-column>
        <e-column field='CustomerID' headerText='Customer ID'
width=120></e-column>
        <e-column field='Freight' headerText='Freight' textAlign='Right'
format='C2' width=90></e-column>
        <e-column field='ShipCity' headerText='Ship City' width=150></e-
column>
      </e-columns>
    </ejs-grid>
  </div>
</template>
<script>
import Vue from "vue";
import { GridPlugin, Page, Toolbar } from "@syncfusion/ej2-vue-grids";
import { data } from "../datasource.js";
Vue.use(GridPlugin);
export default {
  data() {
    return {
      data: data,
      initialGridLoad: true,
      pageSettings: { pageSizes: true, pageSize: 9 },
      toolbar: ['Add', 'Edit', 'Delete', 'Update', 'Cancel']
    };
  },
  methods: {
    dataBound: function() {
      if (this.initialGridLoad) {
        this.initialGridLoad = false;
        let pager = document.getElementsByClassName('e-gridpager');
        let topElement;
        console.log('allowGroping n toolbar', this.$refs.grid);
        if (this.$refs.grid.allowGrouping || this.$refs.grid.toolbar) {
          topElement = this.$refs.grid.allowGrouping ?
document.getElementsByClassName('e-groupdroparea') :
document.getElementsByClassName('e-toolbar');
        } else {
          topElement = document.getElementsByClassName('e-gridheader');
        }
        console.log('insetBefore', this.$refs.grid.$el);
        this.$refs.grid.$el.insertBefore(pager[0], topElement[0]);
      }
    }
  }
}
```

```

    },
    provide: {
      grid: [Page, Toolbar]
    }
  }
</script>
<style>
  @import
  "https://ej2.syncfusion.com/vue/documentation/node_modules/@syncfusion/ej2-
  vue-grids/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/grid/page/default-cs3" %}

During the paging action, the pager component triggers the below three events.

- \* The [created](#) event triggers when Pager is created.
- \* The [click](#) event triggers when the numeric items in the pager is clicked.
- \* The [dropDownChanged](#) event triggers when pageSize DropDownList value is selected.

See Also

- [Group with Paging](#)

### Scrolling in Vue Grid component

The scrollbar will be displayed in the grid when content exceeds the element [width](#) or [height](#). The vertical and horizontal scrollbars will be displayed based on the following criteria:

- The vertical scrollbar appears when the total height of rows present in the grid exceeds its element height.
- The horizontal scrollbar appears when the sum of columns width exceeds the grid element width.
- The [height](#) and [width](#) are used to set the grid height and width, respectively.

The default value for [height](#) and [width](#) is **auto**.

Set width and height

To specify the [width](#) and [height](#) of scroller in pixel, set the pixel value as number.

#### APP.VUE

```

<template>
  <div id="app">
    <ejs-grid :dataSource="data">
      <e-columns>
        <e-column field='OrderID' headerText='Order ID'
        textAlign='Right' width=120></e-column>
        <e-column field='CustomerID' headerText='Customer ID'
        width=150></e-column>
        <e-column field='ShipCity' headerText='Ship City'
        width=150></e-column>
      </e-columns>
    </ejs-grid>
  </div>
</template>

```

```

        <e-column field='ShipCountry' headerText='Ship Country'
width=150></e-column>
        <e-column field='ShipName' headerText='Ship Name'
width=150></e-column>
    </e-columns>
</ejs-grid>
</div>
</template>
<script>
import Vue from "vue";
import { GridPlugin } from "@syncfusion/ej2-vue-grids";
import { data } from "../datasource.js";
Vue.use(GridPlugin);
export default {
  data() {
    return {
      data: data
    };
  }
}
</script>
<style>
  @import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/grid/scroll/default-cs4" %}

### Responsive with parent container

Specify the [width](#) and [height](#) as **100%** to make the grid element fill its parent container. Setting the [height](#) to **100%** requires the grid parent element to have explicit height.

### APP.VUE

```

<template>
  <div id="app">
    <p class="e-text"> The parent container can be resizable by dragging
the bottom-right corner.</p>
    <div id='container' class='e-resizable'>
      <ejs-grid :dataSource='data' height='100%' width='100%'>
        <e-columns>
          <e-column field='OrderID' headerText='Order ID'
textAlign='Right' width=120></e-column>
          <e-column field='CustomerID' headerText='Customer ID'
width=150></e-column>
          <e-column field='ShipCity' headerText='Ship City'
width=150></e-column>
          <e-column field='ShipCountry' headerText='Ship Country'
width=150></e-column>
          <e-column field='ShipName' headerText='Ship Name'
width=150></e-column>
        </e-columns>
      </ejs-grid>
    </div>
  </div>
</template>
<script>

```

```

import Vue from "vue";
import { GridPlugin } from "@syncfusion/ej2-vue-grids";
import { data } from "../datasource.js";
Vue.use(GridPlugin);
export default {
  data() {
    return {
      data: data
    };
  }
}
</script>
<style>
  @import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
  .e-resizable {
    resize: both;
    overflow: auto;
    border: 1px solid red;
    padding: 10px;
    height: 300px;
    min-height: 250px;
    min-width: 250px;
  }
</style>

```

{% previewsample "page.domainurl/code-snippet/grid/scroll/default-cs5" %}

### Sticky Header

You can make the Grid column headers remain fixed while scrolling by using the [enableStickyHeader](#) property.

In the below demo, the Grid headers will be sticky while scrolling the Grid's parent div element.

### APP.VUE

```

<template>
  <div id="app">
    <div style='height:350px;'>
      <ejs-grid ref='grid' :dataSource='data' enableStickyHeader=true>
        <e-columns>
          <e-column field='OrderID' headerText='Order ID'
            textAlign='Right' width=100></e-column>
          <e-column field='CustomerID' headerText='Customer ID'
            width=120></e-column>
          <e-column field='ShipCity' headerText='Ship City'
            width=100></e-column>
          <e-column field='ShipName' headerText='Ship Name'
            width=100></e-column>
        </e-columns>
      </ejs-grid>
    </div>
  </div>
</template>
<script>
import Vue from "vue";
import { GridPlugin } from "@syncfusion/ej2-vue-grids";

```

```
import { data } from './datasource.js'
Vue.use(GridPlugin);
export default {
  data() {
    return {
      data: data,
    };
  }
}
</script>
<style>
  @import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
</style>
```

{% previewsample "page.domainurl/code-snippet/grid/search/default-cs1" %}

### Scroll To Selected Row

You can scroll the grid content to the selected row position by using the [rowSelected](#) event.

### APP.VUE

```
<template>
  <div id="app">
    <ejs-numerictextbox ref='numeric' format='N' min='0' placeholder='Enter
index to select a row' width=200 :showSpinButton='false'
:change='onchange'></ejs-numerictextbox>
    <ejs-grid ref='grid' :dataSource='data' height='280' width='100%'
:rowSelected='rowSelected'>
      <e-columns>
        <e-column field='OrderID' headerText='Order ID'
textAlign='Right' width=120></e-column>
        <e-column field='CustomerID' headerText='Customer ID'
width=150></e-column>
        <e-column field='ShipCity' headerText='Ship City'
width=150></e-column>
        <e-column field='ShipCountry' headerText='Ship Country'
width=150></e-column>
        <e-column field='ShipName' headerText='Ship Name'
width=150></e-column>
      </e-columns>
    </ejs-grid>
  </div>
</template>
<script>
import Vue from "vue";
import { GridPlugin } from "@syncfusion/ej2-vue-grids";
import { NumericTextBoxPlugin } from "@syncfusion/ej2-vue-inputs";
import { data } from './datasource.js';
Vue.use(GridPlugin);
Vue.use(NumericTextBoxPlugin);
export default {
  data() {
    return {
      data: data
    };
  },
}
```

```

methods: {
  onchange: function() {
    this.$refs.grid.selectRow(parseInt(this.$refs.numeric.getText(), 10));
  }
  rowSelected: function (args) {
    let rowHeight =
this.$refs.grid.getRows()[this.$refs.grid.getSelectedRowIndex()[0]].scroll
Height;
    this.$refs.grid.getContent().children[0].scrollTop = rowHeight *
this.$refs.grid.getSelectedRowIndex()[0];
  }
}
}
</script>
<style>
  @import
  "https://ej2.syncfusion.com/vue/documentation/node_modules/@syncfusion/ej2-
vue-grids/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/grid/scroll/selectrow-cs1" %}

Hide the scrollbar when the content is not overflown

You can hide the scrollbar of Grid content by using the [hideScroll](#) method when the content doesn't overflow its parent element.

In the following sample, we have invoked the [hideScroll](#) method inside the [dataBound](#) event.

#### APP.VUE

```

<template>
  <div id="app">
    <ejs-grid ref='grid' :dataSource='data' height='312px'
: dataBound='dataBound'>
      <e-columns>
        <e-column field='OrderID' headerText='Order ID'
textAlign='Right' width=100></e-column>
        <e-column field='CustomerID' headerText='Customer ID'
width=120></e-column>
        <e-column field='ShipCity' headerText='Ship City'
width=100></e-column>
        <e-column field='ShipName' headerText='Ship Name'
width=100></e-column>
      </e-columns>
    </ejs-grid>
  </div>
</template>
<script>
import Vue from "vue";
import { GridPlugin } from "@syncfusion/ej2-vue-grids";
import { data } from './datasource.js'
Vue.use(GridPlugin);
export default {
  data() {
    return {
      data: data.slice(0, 2),

```



```

    };
  },
  methods: {
    dataBound: function() {
      this.$refs.grid.hideScroll();
    }
  }
}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/grid/search/default-cs2" %}

### Frozen in Vue Grid component

Frozen rows and columns provides an option to make rows and columns always visible in the top and left side of the grid while scrolling.

In this demo, the [frozenColumns](#) is set as '2' and the [frozenRows](#) is set as '3'. Hence, the left two columns and top three rows are frozen.

#### APP.VUE

```

<template>
  <div id="app">
    <ejs-grid :dataSource='data' height=315 :frozenColumns='2'
    :frozenRows='3' :allowSelection='false' :enableHover='false' width='600'>
      <e-columns>
        <e-column field='OrderID' headerText='Order ID'
        textAlign='Right' width=120></e-column>
        <e-column field='CustomerID' headerText='Customer ID'
        width=150></e-column>
        <e-column field='OrderDate' headerText='Order Date'
        width=130 format='yMd' textAlign='Right' type='date'></e-column>
        <e-column field='ShipName' headerText='Ship Name'
        width=150></e-column>
        <e-column field='ShipAddress' headerText='Ship Address'
        width=170></e-column>
        <e-column field='ShipCity' headerText='Ship City'
        width=150></e-column>
        <e-column field='ShipCountry' headerText='Ship Country'
        width=150></e-column>
        <e-column field='ShipRegion' headerText='Ship Region'
        width=150></e-column>
        <e-column field='ShipPostalCode' headerText='Ship Postal
        Code' width=150></e-column>
        <e-column field='Freight' headerText='Freight'
        width=120></e-column>
      </e-columns>
    </ejs-grid>
  </div>
</template>
<script>
import Vue from "vue";
import { GridPlugin, Freeze } from "@syncfusion/ej2-vue-grids";

```

```
import { data } from "../datasource.js";
Vue.use(GridPlugin);
export default {
  data() {
    return {
      data: data
    };
  },
  provide: {
    grid: [Freeze]
  }
}
</script>
<style>
  @import
  "https://ej2.syncfusion.com/vue/documentation/node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
</style>
```

{% previewsample "page.domainurl/code-snippet/grid/scroll/default-cs1" %}

- \* The **height** and **width** must be set while giving frozen rows and columns.
- \* Frozen rows and columns should not be set outside the grid view port.
- \* Frozen Grid will support row and column virtualization feature, which helps to improve the Grid performance while loading a large dataset.

### Frozen Grid Limitations

The following features are not supported in frozen rows and columns:

- Detail Template
- Hierarchy Grid

### Freeze Direction

You can freeze the Grid columns on the left or right side by using the [column.freeze](#) property and the remaining columns will be movable. The grid will automatically move the columns to the left or right position based on the [column.freeze](#) value.

Types of the [column.freeze](#) directions:

- **Left:** Allows you to freeze the columns at the left.
- **Right:** Allows you to freeze the columns at the right.
- **Fixed:** Allows you to lock the column at a fixed position by ensuring its visibility during horizontal scroll.

In this demo, the **ShipCountry** column is frozen at the left and the **CustomerID** column is frozen at the right side of the content table.

### APP.VUE

```
<template>
  <div id="app">
```

```

        <ejs-grid :dataSource='data' height=315 :frozenRows='2'
:enableHover='false'>
            <e-columns>
                <e-column field='OrderID' headerText='Order ID'
textAlign='Right' width=120></e-column>
                <e-column field='Freight' headerText='Freight'
width=120></e-column>
                <e-column field='CustomerID' headerText='Customer ID'
width=150 freeze='Right'></e-column>
                <e-column field='OrderDate' headerText='Order Date'
width=130 format='yMd' textAlign='Right' type='date'></e-column>
                <e-column field='ShipName' headerText='Ship Name'
width=150></e-column>
                <e-column field='ShipAddress' headerText='Ship Address'
width=170></e-column>
                <e-column field='ShipCity' headerText='Ship City'
width=150></e-column>
                <e-column field='ShipCountry' headerText='Ship Country'
width=150 freeze='Left'></e-column>
            </e-columns>
        </ejs-grid>
    </div>
</template>
<script>
import Vue from "vue";
import { GridPlugin, Freeze } from "@syncfusion/ej2-vue-grids";
import { data } from "../datasource.js";
Vue.use(GridPlugin);
export default {
    data() {
        return {
            data: data
        };
    },
    provide: {
        grid: [Freeze]
    }
}
</script>
<style>
    @import
"https://ej2.syncfusion.com/vue/documentation/node_modules/@syncfusion/ej2-
vue-grids/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/grid/scroll/default-cs2" %}

Freeze Direction is not compatible with the [isFrozen](#) and [frozenColumns](#) properties.

### Deprecated Methods

Deprecated Methods | [Previous](#) | [Current](#) | [Suggested Alternative Methods](#) | [Example for achieving the same results](#)

**getMovableRows()** | In the previous architecture of frozen grid, three separate tables were created for the left, right, and movable contents. When calling this method, it would return only the movable table rows (tr's). | In the current architecture, the frozen left, right, and movable sections are applied within a

single table. When calling this method, it will return all table rows (tr's) of the entire table. However, in this approach, we have introduced the `e-unfreeze` class for movable cells. This allows us to selectively retrieve the movable rows using the `e-unfreeze` class selector. | `getRows()` | `gridInstance.getMovableRows()[0].querySelectorAll('.e-unfreeze')` // Deprecated <br><br> (or) <br><br> `gridInstance.getRows()[0].querySelectorAll('.e-unfreeze')` // Alternative method

`getFrozenRightRows()` | In the previous architecture, this method would return only the table rows (tr's) from the freeze right table. | In the current architecture, the frozen left, right, and movable sections are applied within a single table. When calling this method, it will return all the rows (tr's) of the entire table. In this new approach, we have introduced the `e-rightfreeze` class for right freeze cells. As a result, you can now selectively retrieve the right freeze rows using the `e-rightfreeze` class selector. | `getRows()` | `gridInstance.getFrozenRightRows()[0].querySelectorAll('.e-rightfreeze')` // Deprecated <br><br> (or) <br><br> `gridInstance.getRows()[0].querySelectorAll('.e-rightfreeze')` // Alternative method

`getMovableRowByIndex()` <br> `getFrozenRowByIndex()` <br> `getFrozenRightRowByIndex()` | In the previous architecture, you could select rows by using separate methods for each table section. Like, <br> `getMovableRowByIndex` - select a movable row <br> `getFrozenRowByIndex` - select a freeze row <br> `getFrozenRightRowByIndex` - select a right freeze row. | In the current architecture, the `getMovableRowByIndex`, `getFrozenRightRowByIndex` and `getFrozenRowByIndex` methods all return the same table row (tr) based on the given index. Additionally, class names for table cells (td's) have been separated as follows: <br> Left-Freeze : `e-leftfreeze` <br> Movable : `e-unfreeze` <br> Right-Freeze : `e-rightfreeze`.<br>This separation of class names makes it easier to target and customize the cells within the particular row. | `getRowByIndex()` | **To get the left freeze cells:** <br> `gridInstance.getRowByIndex(1).querySelectorAll('.e-leftfreeze')` <br><br> **To get the movable cells:** <br> `gridInstance.getRowByIndex(1).querySelectorAll('.e-unfreeze')` <br><br> **To get the right freeze cells:** <br> `gridInstance.getRowByIndex(1).querySelectorAll('.e-rightfreeze')`

`getMovableCellFromIndex()` <br> `getFrozenRightCellFromIndex()` | `getMovableCellFromIndex()` - select a particular cell in the movable table. <br> `getFrozenRightCellFromIndex()` - select a particular cell in the right freeze table. | In the new approach, you can select a particular cell by using both the `getFrozenRightCellFromIndex` and `getMovableCellFromIndex` methods. | `getCellFromIndex()` | `gridInstance.getCellFromIndex(1,1)`

`getMovableDataRows()` <br> `getFrozenRightDataRows()` <br> `getFrozenDataRows()` | These methods returns the viewport data rows for the freeze, movable, and right tables separately. | In the new approach, when calling the `getMovableDataRows`, `getFrozenRightDataRows`, and `getFrozenDataRows` methods, returns the entire viewport data rows. You can then select specific cells within these rows using the following selectors <br> Left-Freeze : `e-leftfreeze` <br> Movable : `e-unfreeze` <br> \* Right-Freeze : `e-rightfreeze`. | `getDataRows()` | **To get the movable data cells:** <br> `gridInstance.getDataRows()[0].querySelectorAll('.e-unfreeze')` <br><br> **To get the right freeze data cells:** <br> `gridInstance.getDataRows()[0].querySelectorAll('.e-rightfreeze')` <br><br> **To get the left freeze data cells:** <br> `gridInstance.getDataRows()[0].querySelectorAll('.e-leftfreeze')`

`getMovableColumnHeaderByIndex()` <br> `getFrozenRightColumnHeaderByIndex()` <br> `getFrozenLeftColumnHeaderByIndex()` | In the previous architecture, these methods selects the movable, right freeze, and left freeze headers from the table separately. | In the new approach, when calling the `getMovableColumnHeaderByIndex`, `getFrozenRightColumnHeaderByIndex`, and

`getFrozenLeftColumnHeaderByIndex` methods, you will still receive the same results as before. | `getColumnHeaderByIndex()` | `gridInstance.getColumnHeaderByIndex(1)`

When a validation message is displayed in the frozen part (Left, Right, Fixed) of the table, scrolling is prevented until the validation message is cleared.

### Virtual scroll in Vue Grid component

Grid allows you to load large amount of data without performance degradation.

To use virtualization, you need to inject `VirtualScroll` module in the `provide` section.

#### Row Virtualization

Row virtualization allows you to load and render rows only in content viewport. It is an alternative way of paging in which the data will load while scrolling vertically.

To setup the row virtualization, you need to define [enableVirtualization](#) as true and content height by [height](#) property.

The number of records displayed in the Grid is determined implicitly by height of content area. Also you have an option to define visible number of records by [pageSettings.pageSize](#) property. The loaded data will be cached and reused when it is needed for next time.

### APP.VUE

```
<template>
  <div id="app">
    <ejs-grid :dataSource='data' height=300 :enableVirtualization=true
:pageSettings='options' :editSettings='editSettings' :toolbar='toolbar'>
      <e-columns>
        <e-column field='TaskID' headerText='Task ID'
textAlign='Right' width=100 :isPrimaryKey='true'
:validationRules='rules'></e-column>
        <e-column field='Engineer' width=100></e-column>
        <e-column field='Designation' width=140
editType='dropdownedit' :validationRules='rules'></e-column>
        <e-column field='Estimation' textAlign='Right' width=110
editType='numericedit' :validationRules='rules'></e-column>
        <e-column field='Status' width=140
editType='dropdownedit'></e-column>
      </e-columns>
    </ejs-grid>
  </div>
</template>
<script>
import Vue from "vue";
import { GridPlugin, VirtualScroll, Toolbar, Edit } from "@syncfusion/ej2-
vue-grids";
Vue.use(GridPlugin);
let names = ['TOM', 'Hawk', 'Jon', 'Chandler', 'Monica', 'Rachel', 'Phoebe',
'Gunter', 'Ross', 'Geller', 'Joey', 'Bing', 'Tribbiani', 'Janice', 'Bong',
'Perk', 'Green', 'Ken', 'Adams'];
let hours = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10];
let designation = ['Manager', 'Engineer 1', 'Engineer 2', 'Developer',
'Tester'];
let status = ['Completed', 'Open', 'In Progress', 'Review', 'Testing']
let data = (count) => {
  let result = [];
```

```

    for (let i = 0; i < count; i++) {
      result.push({
        TaskID: i + 1,
        Engineer: names[Math.round(Math.random() * names.length)] ||
names[0],
        Designation: designation[Math.round(Math.random() *
designation.length)] || designation[0],
        Estimation: hours[Math.round(Math.random() * hours.length)] ||
hours[0],
        Status: status[Math.round(Math.random() * status.length)] ||
status[0]
      });
    }
    return result;
  };
export default {
  data() {
    return {
      data: data(1000),
      options: { pageSize: 50 },
      editSettings: { allowEditing: true, allowAdding: true, allowDeleting:
true },
      toolbar: ['Add', 'Edit', 'Delete', 'Update', 'Cancel'],
      rules: { required: true },
    };
  },
  provide: {
    grid: [VirtualScroll, Toolbar, Edit]
  }
}
</script>
<style>
  @import
  "https://ej2.syncfusion.com/vue/documentation/node_modules/@syncfusion/ej2-
vue-grids/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/grid/virtualscroll/default-cs4" %}

### Column Virtualization

Column virtualization allows you to virtualize columns. It will render columns which are in viewport. You can scroll horizontally to view more columns.

To setup the column virtualization, set the [enableVirtualization](#) and [enableColumnVirtualization](#) properties as **true**.

### APP.VUE

```

<template>
  <div id="app">
    <ejs-grid :dataSource='data' height=300 :enableVirtualization=true
:enableColumnVirtualization=true :pageSettings='options'
:editSettings='editSettings' :toolbar='toolbar'>
      <e-columns>
        <e-column field='SNo' headerText='S.No' width=140
:isPrimaryKey='true' :validationRules='rules'></e-column>

```

```

        <e-column field='FIELD1' headerText='Player Name' width=140
editType='dropdownedit' :validationRules='rules'></e-column>
        <e-column field='FIELD2' headerText='Year' width=120
textAlign='Right'></e-column>
        <e-column field='FIELD3' headerText='Stint' width=120
textAlign='Right'></e-column>
        <e-column field='FIELD4' headerText='TMID' width=120
textAlign='Right'></e-column>
        <e-column field='FIELD5' headerText='LGID' width=120
textAlign='Right'></e-column>
        <e-column field='FIELD6' headerText='GP' width=120
textAlign='Right'></e-column>
        <e-column field='FIELD7' headerText='GS' width=120
textAlign='Right'></e-column>
        <e-column field='FIELD8' headerText='Minutes' width=120
textAlign='Right'></e-column>
        <e-column field='FIELD9' headerText='Points' width=120
textAlign='Right'></e-column>
        <e-column field='FIELD10' headerText='oRebounds' width=130
textAlign='Right'></e-column>
        <e-column field='FIELD11' headerText='dRebounds' width=130
textAlign='Right'></e-column>
        <e-column field='FIELD12' headerText='Rebounds' width=120
textAlign='Right'></e-column>
        <e-column field='FIELD13' headerText='Assists' width=120
textAlign='Right'></e-column>
        <e-column field='FIELD14' headerText='Steals' width=120
textAlign='Right'></e-column>
        <e-column field='FIELD15' headerText='Blocks' width=120
textAlign='Right'></e-column>
        <e-column field='FIELD16' headerText='Turnovers' width=130
textAlign='Right'></e-column>
        <e-column field='FIELD17' headerText='PF' width=130
textAlign='Right'></e-column>
        <e-column field='FIELD18' headerText='fgAttempted' width=150
textAlign='Right'></e-column>
        <e-column field='FIELD19' headerText='fgMade' width=120
textAlign='Right'></e-column>
        <e-column field='FIELD20' headerText='ftAttempted' width=150
textAlign='Right'></e-column>
        <e-column field='FIELD21' headerText='ftMade' width=120
textAlign='Right'></e-column>
        <e-column field='FIELD22' headerText='ThreeAttempted'
width=150 textAlign='Right'></e-column>
        <e-column field='FIELD23' headerText='ThreeMade' width=130
textAlign='Right'></e-column>
        <e-column field='FIELD24' headerText='PostGP' width=120
textAlign='Right'></e-column>
        <e-column field='FIELD25' headerText='PostGS' width=120
textAlign='Right'></e-column>
        <e-column field='FIELD26' headerText='PostMinutes' width=120
textAlign='Right'></e-column>
        <e-column field='FIELD27' headerText='PostPoints' width=130
textAlign='Right'></e-column>
        <e-column field='FIELD28' headerText='PostoRebounds'
width=130 textAlign='Right'></e-column>

```

```

        <e-column field='FIELD29' headerText='PostdRebounds'
width=130 textAlign='Right'></e-column>
        <e-column field='FIELD30' headerText='PostRebounds'
width=130 textAlign='Right' editType='numericedit'
:validationRules='rules'></e-column>
    </e-columns>
</ejs-grid>
</div>
</template>
<script>
import Vue from "vue";
import { GridPlugin, VirtualScroll, Toolbar, Edit } from "@syncfusion/ej2-
vue-grids";
Vue.use(GridPlugin);
let virtualData = [];
let names: string[] = ['hardire', 'abramjo01', 'aubucch01', 'Hook',
'Rumpelstiltskin', 'Belle', 'Emma', 'Regina', 'Aurora', 'Elsa', 'Anna',
'Snow White', 'Prince Charming', 'Cora', 'Zelena', 'August', 'Mulan',
'Graham', 'Discord', 'Will', 'Robin Hood', 'Jiminy Cricket', 'Henry',
'Neal', 'Red', 'Aaran', 'Aaren', 'Aarez', 'Aarman', 'Aaron', 'Aaron-James',
'Aarron', 'Aaryan', 'Aaryn', 'Aayan', 'Aazaan', 'Abaan', 'Abbas',
'Abdallah', 'Abdalroof', 'Abdihakim', 'Abdirahman', 'Abdisalam', 'Abdul',
'Abdul-Aziz', 'Abdulbasir', 'Abdulkadir', 'Abdulkarem', 'Abdulkhader',
'Abdullah', 'Abdul-Majeed', 'Abdulmalik', 'Abdul-Rehman', 'Abdur',
'Abdurraheem', 'Abdur-Rahman', 'Abdur-Rehmaan', 'Abel', 'Abhinav',
'Abhisumant', 'Abid', 'Abir', 'Abraham', 'Abu', 'Abubakar', 'Ace', 'Adain',
'Adam', 'Adam-James', 'Addison', 'Addisson', 'Adegbola', 'Adegbolahan',
'Aden', 'Adenn', 'Adie', 'Adil', 'Aditya', 'Adnan', 'Adrian', 'Adrien',
'Aedan', 'Aedin', 'Aedyn', 'Aeron', 'Afonso', 'Ahmad', 'Ahmed', 'Ahmed-
Aziz', 'Ahoua', 'Ahtasham', 'Aiadan', 'Aidan', 'Aiden', 'Aiden-Jack',
'Aiden-Vee'];
function dataSource() {
    for (let i = 0; i < 1000; i++) {
        virtualData.push({ 'SNo': i + 1, 'FIELD1':
names[Math.floor(Math.random() * names.length)],
        'FIELD2': 1967 + (i % 10), 'FIELD3': Math.floor(Math.random() *
200),
        'FIELD4': Math.floor(Math.random() * 100), 'FIELD5':
Math.floor(Math.random() * 2000), 'FIELD6': Math.floor(Math.random() *
1000), 'FIELD7': Math.floor(Math.random() * 100), 'FIELD8':
Math.floor(Math.random() * 10), 'FIELD9': Math.floor(Math.random() * 10),
        'FIELD10': Math.floor(Math.random() * 100), 'FIELD11':
Math.floor(Math.random() * 100), 'FIELD12': Math.floor(Math.random() *
1000), 'FIELD13': Math.floor(Math.random() * 10), 'FIELD14':
Math.floor(Math.random() * 10), 'FIELD15': Math.floor(Math.random() * 1000),
        'FIELD16': Math.floor(Math.random() * 200), 'FIELD17':
Math.floor(Math.random() * 300), 'FIELD18': Math.floor(Math.random() * 400),
        'FIELD19': Math.floor(Math.random() * 500), 'FIELD20':
Math.floor(Math.random() * 700), 'FIELD21': Math.floor(Math.random() * 800),
        'FIELD22': Math.floor(Math.random() * 1000), 'FIELD23':
Math.floor(Math.random() * 2000), 'FIELD24': Math.floor(Math.random() *
150), 'FIELD25': Math.floor(Math.random() * 1000), 'FIELD26':
Math.floor(Math.random() * 100), 'FIELD27': Math.floor(Math.random() * 400),
        'FIELD28': Math.floor(Math.random() * 600), 'FIELD29':
Math.floor(Math.random() * 500), 'FIELD30': Math.floor(Math.random() * 300),
    });
    }
}

```



```

}
dataSource();
export default {
  data() {
    return {
      data: virtualData,
      options: { pageSize: 50 },
      editSettings: { allowEditing: true, allowAdding: true, allowDeleting:
true },
      toolbar: ['Add', 'Edit', 'Delete', 'Update', 'Cancel'],
      rules: { required: true },
    };
  },
  provide: {
    grid: [VirtualScroll, Toolbar, Edit]
  }
}
</script>
<style>
  @import
  "https://ej2.syncfusion.com/vue/documentation/node_modules/@syncfusion/ej2-
vue-grids/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/grid/virtualscroll/default-cs5" %}

Column's [width](#) is required for column virtualization. If column's width is not defined then Grid will consider its value as 200px.

### Virtualization with Grouping

Both the row and column virtualization can be used along with grouping. At initial rendering, the virtual height of scrollbar will be set based on the total number of records and after grouping, it will be refreshed based on the grouped state(expand/collapse). While collapse the group caption row in current viewport then the next view page grouped records are shown.

The collapsed/expanded state will persist only for local dataSource while scrolling.

### Limitations for virtual scrolling

- While using column virtual scrolling, column width should be in pixel. Percentage values are not accepted.
- Due to the element height limitation in browsers, the maximum number of records loaded by the Grid is limited by the browser capability.
- The cell selection is not supported for both row and column virtual scrolling.
- Virtual scrolling is not compatible with batch editing, detail template, rowspan, colspan and hierarchy features.
- Group expand and collapse state will not be persisted.
- Since data is virtualized in grid, the aggregated information and total group items are displayed based on the current view items.

To get these information regardless of the view items, refer to the

[Group with Page](#) topic.

- The page size provided must be two times larger than the number of visible rows in the grid.

If the page size is failed to meet this condition then the size will be determined by grid.

- The height of the grid content is calculated using the row height and total number of records in the data source and hence features which changes row height such as text wrapping are not supported. If you want to increase the row height to accommodate the content then you can specify the [rowHeight](#) as below to ensure all the table rows are in same height.

```
<ejs-grid :dataSource='data' rowHeight=50>
```

```
</ejs-grid>
```

- Programmatic selection using [selectRows](#) method is not supported in virtual scrolling.

#### Browser height limitation in virtual scrolling and solution

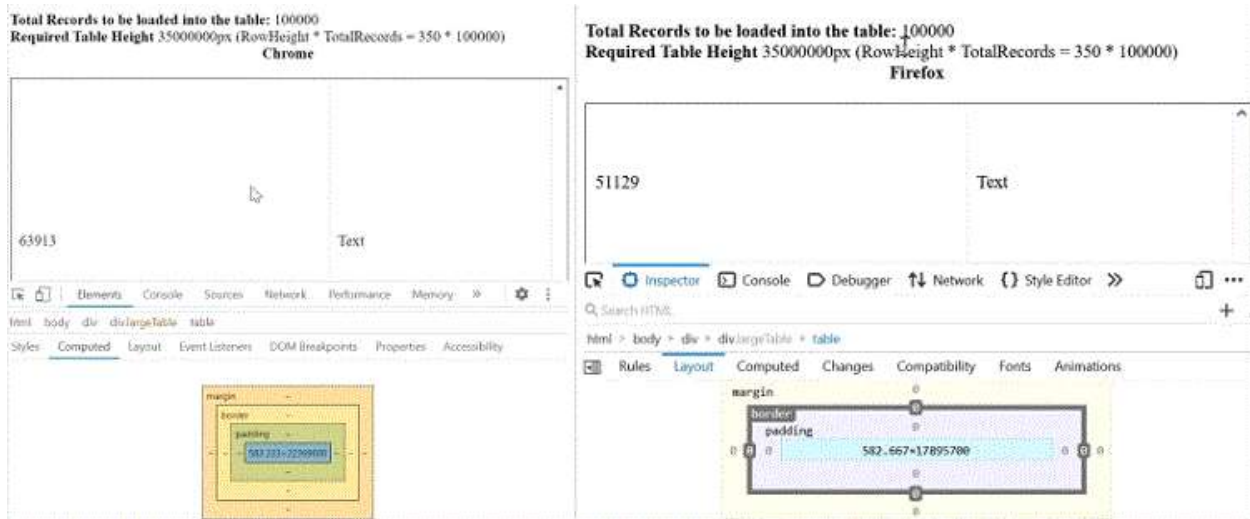
You can load millions of records in the Grid by using virtual scrolling, where the grid loads and renders rows on-demand while scrolling vertically. As a result, Grid lightens the browser's load by minimizing the DOM elements and rendering elements visible in the viewport. The height of the grid is calculated using the Total Records Count \* [Row Height](#) property.

The browser has some maximum pixel height limitations for the scroll bar element. The content placed above the maximum height can't be scrolled if the element height is greater than the browser's maximum height limit. The browser height limit affects the virtual scrolling of the grid. When a large number of records are bound to the Grid, it can only display the records until the maximum height limit of the browser. Once the browser's height limit is reached while scrolling, the user won't be able to scroll further to view the remaining records.

For example, if the row height is set as 30px and the total record count is 1000000(1 million), then the height of the grid element will be 30,000,000 pixels. In this case, the browser's maximum height limit for a div is about 22,369,600 (The maximum pixel height limitation differs for different browsers). The records above the maximum height limit of the browser can't be scrolled.

This height limitation is not related to the Grid component. It fully depends on the default behavior of the browser. The same issue is reproduced in the normal HTML table too.

The following image illustrates the height limitation issue of a normal HTML table in different browsers (Chrome and Firefox).



Grid component also faced the same issue as mentioned in the below image.



The Grid has an option to overcome this limitation of the browser in the following ways.

#### [Solution 1: Using external buttons](#)

You can prevent the height limitation problem in the browser when scrolling through millions of records by loading the segment of data through different strategy.

In the following sample, Grid is rendered with a large number of records(nearly 2 million). Here, you can scroll 0.5 million records at a time in Grid. Once you reach the last page of 0.5 million records, the **Load Next Set** button will be shown at the bottom of the Grid. By clicking that button, you can view the next set of 0.5 million records in Grid. Also, the **Load Previous Set** button will be shown at the top of the Grid to load the previous set of 0.5 million records.

Let's see the step by step procedure for how we can overcome the limitation in the Syncfusion Grid component.

1. Create a custom adaptor by extending `UrlAdaptor` and binding it to the grid `dataSource` property. In the `processQuery` method of the custom adaptor, we handled the `Skip` query based on the current page set to perform the data operation with whole records on the server.

```

`ts
class CustomUrlAdaptor extends UrlAdaptor {
  processQuery(args) {
    if (arguments[1].queries) {
      for (var i = 0; i < arguments[1].queries.length; i++) {
        if (arguments[1].queries[i].fn === 'onPage') {
          // pageSet - defines the number of segments that we are going to split the 2million records. In this
          // example we have considered 0.5 million records for each set so the pageSet is 1, 2, 3 and 4.
          // maxRecordsPerPageSet – In this example we define the value as 0.5 million.
          // gridPageSize – the pageSize that we have defined in the Grid pageSettings->pageSize property
          // customize the pageIndex based on the current pageSet (It send the skip query including the previous
          // pageSet ) so that the other operations performed for total 2millions records instead of 0.5 million alone.
          arguments[1].queries[i].e.pageIndex = (((pageSet - 1) * maxRecordsPerPageSet) / gridPageSize) +
            arguments[1].queries[i].e.pageIndex;
        }
      }
    }
    let original = super.processQuery.apply(this, arguments);
    return original;
  }
}

data: new DataManager({
  adaptor: new CustomUrlAdaptor(),
  url: "Home/UrlDatasource"
})
`

```

## 2. Render the grid by define the following features.

```

`ts
<ejs-grid :dataSource="data" :enableVirtualization='true' height='360' :pageSettings='pageSettings'
:beforeDataBound='beforeDataBound'>
  <e-columns>
    <e-column field='OrderID' headerText='Order ID' textAlign='Right' width=90></e-column>
    .....
    .....
  </e-columns>
</ejs-grid>

```

```
</e-columns>
```

```
</ejs-grid>
```

```
,
```

3. In the `beforeDataBound` event, we set the `args.count` as 0.5 million to perform scrolling with 0.5 million records and all the data operations are performed with whole records which is handled using the custom adaptor. And also particular segment records count is less than 0.5 million means it will directly assigned the original segmented count instead of 0.5 million.

```
`ts
```

```
beforeDataBound(args) {
```

```
// storing the total records count which means 2 million records count
```

```
totalRecords = args.count;
```

```
// change the count with respect to maxRecordsPerPageSet (maxRecordsPerPageSet = 500000)
```

```
args.count = args.count - ((pageSet - 1) * maxRecordsPerPageSet) > maxRecordsPerPageSet
```

```
?maxRecordsPerPageSet : args.count - ((pageSet - 1) * maxRecordsPerPageSet);
```

```
}
```

```
,
```

4. Render “Load Next Set” button and “Load Previous Set” button at bottom and top of the grid component.

```
`ts
```

```
<ejs-button cssClass='e-info prevbtn' :onClick='prevBtnClick'>Load Previous Set...</ejs-button>
```

```
<ejs-grid :dataSource='data' :enableVirtualization='true' height='360' :pageSettings='pageSettings'
:beforeDataBound='beforeDataBound'>
```

```
<e-columns>
```

```
<e-column field='OrderID' headerText='Order ID' textAlign='Right' width=90></e-column>
```

```
.....
```

```
.....
```

```
</e-columns>
```

```
</ejs-grid>
```

```
<ejs-button cssClass='e-info nxtbtn' :onClick='nxtBtnClick'>Load Next Set...</ejs-button>
```

```
,
```

5. While click on the **Load Next Set** / **Load Previous Set** button corresponding page data set is loaded to view remaining records of total 2 millions records after doing some simple calculation.

```
`ts
```

```
// Triggered when clicking the Previous/ Next button.
prevNxtBtnClick(args) {
  if (this.grid.element.querySelector('.e-content') && this.grid.element.querySelector('.e-content').getAttribute('aria-busy') === 'false') {
    // Increase/decrease the pageSet based on the target element.
    pageSet = args.target.classList.contains('prevbtn') ? --pageSet : ++pageSet;
    this.rerenderGrid(); // Re-render the Grid component.
  }
}
,
```

You can view the hosted link for this sample [here](#).

LOAD PREVIOUS SET...

Order ID	Customer ID	Freight	Country	Status
1	Alfki	\$12.30	Denmark	Active
2	Anatr	\$3.30	Brazil	Inactive
3	Anton	\$4.30	Germany	Active
4	Blomp	\$5.30	Austria	Inactive
5	Bolid	\$6.30	Switzerland	Active
6	Hanar	\$12.30	France	Active
7	Thomas	\$3.30	Itali	Inactive
8	Jack	\$4.30	Austria	Active
9	Alfki	\$5.30	USA	Inactive
10	Hanar	\$6.30	Belgium	Active

LOAD NEXT SET...

If you perform grid actions such as filtering, sorting, etc., after scrolling through the 0.5 million data, the Grid performs those data actions with the whole records, not just the current loaded 0.5 million data.

#### *Solution 2: Using RowHeight property*

You can reduce the [row height](#) using the [rowHeight](#) property of the Grid. It will reduce the overall height to accommodate more rows. But this approach optimizes the limitation, but if the height limit is reached after reducing row height also, you have to opt for the previous solution or use paging.

In the following image, you can see how many records will be scrollable when setting rowHeight to "36px" and "30px".

**RowHeight = 36px**

Order ID	Customer ID	Freight	Country	Status
1	Alfki	\$12.30	Denmark	Active
2	Anatr	\$3.30	Brazil	Inactive
3	Anton	\$4.30	Germany	Active
4	Blomp	\$5.30	Austria	Inactive
5	Bolid	\$6.30	Switzerland	Active
6	Hanar	\$12.30	France	Active
7	Thomas	\$3.30	Itali	Inactive
8	Jack	\$4.30	Austria	Active

**RowHeight = 30px**

Order ID	Customer ID	Freight	Country	Status
1	Alfki	\$12.30	Denmark	Active
2	Anatr	\$3.30	Brazil	Inactive
3	Anton	\$4.30	Germany	Active
4	Blomp	\$5.30	Austria	Inactive
5	Bolid	\$6.30	Switzerland	Active
6	Hanar	\$12.30	France	Active
7	Thomas	\$3.30	Itali	Inactive
8	Jack	\$4.30	Austria	Active
9	Alfki	\$5.30	USA	Inactive
10	Hanar	\$6.30	Belgium	Active

### *Solution 3: Using paging instead of virtual scrolling*

Similar to virtual scrolling, the [paging](#) feature also loads the data in an on-demand concept. Pagination is also compatible with all the other features (Grouping, Editing, etc.) in Grid. So, use the paging feature instead of virtual scrolling to view a large number of records in the Grid without any kind of performance degradation or browser height limitation.

### Infinite scroll in Vue Grid component

Infinite scrolling is used to load a huge amount of data without degrading the Grid performance. This feature works like the lazy loading concept, which means the buffer data is loaded only when the scrollbar reaches the end of the scroller.

To enable Infinite scrolling, set `enableInfiniteScrolling` property as true.

In this feature, Grid will not make a new data request when you visit the same page again.

### **APP.VUE**

```
<template>
  <div id="app">
```

```

    <ejs-grid :dataSource='data' height=300
:enableInfiniteScrolling='true' :pageSettings='options'>
      <e-columns>
        <e-column field='TaskID' headerText='Task ID'
textAlign='Right' width=70></e-column>
        <e-column field='Engineer' width=100></e-column>
        <e-column field='Designation' width=100></e-column>
        <e-column field='Estimation' textAlign='Right'
width=100></e-column>
        <e-column field='Status' width=100></e-column>
      </e-columns>
    </ejs-grid>
  </div>
</template>
<script>
import Vue from "vue";
import { GridPlugin, InfiniteScroll } from "@syncfusion/ej2-vue-grids";
Vue.use(GridPlugin);
let names = ['TOM', 'Hawk', 'Jon', 'Chandler', 'Monica', 'Rachel', 'Phoebe',
'Gunter', 'Ross', 'Geller', 'Joey', 'Bing', 'Tribbiani', 'Janice', 'Bong',
'Perk', 'Green', 'Ken', 'Adams'];
let hours = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10];
let designation = ['Manager', 'Engineer 1', 'Engineer 2', 'Developer',
'Tester'];
let status = ['Completed', 'Open', 'In Progress', 'Review', 'Testing']
let data = (count) => {
  let result = [];
  for (let i = 0; i < count; i++) {
    result.push({
      TaskID: i + 1,
      Engineer: names[Math.round(Math.random() * names.length)] ||
names[0],
      Designation: designation[Math.round(Math.random() *
designation.length)] || designation[0],
      Estimation: hours[Math.round(Math.random() * hours.length)] ||
hours[0],
      Status: status[Math.round(Math.random() * status.length)] ||
status[0]
    });
  }
  return result;
};
export default {
  data() {
    return {
      data: data(1000),
      options: { pageSize: 50 }
    };
  },
  provide: {
    grid: [InfiniteScroll]
  }
}
</script>
<style>
  @import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
</style>

```



```
{% previewsample "page.domainurl/code-snippet/grid/virtualscroll/default-cs1" %}
```

### InitialBlocks

You can define the initial loading pages count by using `infiniteScrollSettings.initialBlocks` property. By default, this feature loads three pages in initial rendering.

In the below demo, we have changed this property value to load five page records instead of three.

### APP.VUE

```
<template>
  <div id="app">
    <ejs-grid :dataSource='data' height=300
:enableInfiniteScrolling='true' :pageSettings='options'
:infiniteScrollSettings='infiniteOptions'>
      <e-columns>
        <e-column field='TaskID' headerText='Task ID'
textAlign='Right' width=70></e-column>
        <e-column field='Engineer' width=100></e-column>
        <e-column field='Designation' width=100></e-column>
        <e-column field='Estimation' textAlign='Right'
width=100></e-column>
        <e-column field='Status' width=100></e-column>
      </e-columns>
    </ejs-grid>
  </div>
</template>
<script>
import Vue from "vue";
import { GridPlugin, InfiniteScroll } from "@syncfusion/ej2-vue-grids";
Vue.use(GridPlugin);
let names = ['TOM', 'Hawk', 'Jon', 'Chandler', 'Monica', 'Rachel', 'Phoebe',
'Gunter', 'Ross', 'Geller', 'Joey', 'Bing', 'Tribbiani', 'Janice', 'Bong',
'Perk', 'Green', 'Ken', 'Adams'];
let hours = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10];
let designation = ['Manager', 'Engineer 1', 'Engineer 2', 'Developer',
'Tester'];
let status = ['Completed', 'Open', 'In Progress', 'Review', 'Testing']
let data = (count) => {
  let result = [];
  for (let i = 0; i < count; i++) {
    result.push({
      TaskID: i + 1,
      Engineer: names[Math.round(Math.random() * names.length)] ||
names[0],
      Designation: designation[Math.round(Math.random() *
designation.length)] || designation[0],
      Estimation: hours[Math.round(Math.random() * hours.length)] ||
hours[0],
      Status: status[Math.round(Math.random() * status.length)] ||
status[0]
    });
  }
  return result;
};
```

```

export default {
  data() {
    return {
      data: data(1000),
      options: { pageSize: 50 },
      infiniteOptions: { initialBlocks: 5 }
    };
  },
  provide: {
    grid: [InfiniteScroll]
  }
}
</script>
<style>
  @import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/grid/virtualscroll/default-cs2" %}

### Cache Mode

Cache is used to store the loaded rows object in the Grid instance which can be reused for creating the row elements whenever you scroll to already visited page. Also, this mode maintains row elements based on the `infiniteScrollSettings.maxBlocks` count value, once this limit exceeds then it will remove row elements from DOM for new rows.

To enable the cache mode in Infinite scrolling, set `infiniteScrollSettings.enableCache` property as true.

### APP.VUE

```

<template>
  <div id="app">
    <ejs-grid :dataSource='data' height=300
:enableInfiniteScrolling='true' :pageSettings='options'
:infiniteScrollSettings='infiniteOptions'>
      <e-columns>
        <e-column field='TaskID' headerText='Task ID'
textAlign='Right' width=70></e-column>
        <e-column field='Engineer' width=100></e-column>
        <e-column field='Designation' width=100></e-column>
        <e-column field='Estimation' textAlign='Right'
width=100></e-column>
        <e-column field='Status' width=100></e-column>
      </e-columns>
    </ejs-grid>
  </div>
</template>
<script>
import Vue from "vue";
import { GridPlugin, InfiniteScroll } from "@syncfusion/ej2-vue-grids";
Vue.use(GridPlugin);
let names = ['TOM', 'Hawk', 'Jon', 'Chandler', 'Monica', 'Rachel', 'Phoebe',
'Gunter', 'Ross', 'Geller', 'Joey', 'Bing', 'Tribbiani', 'Janice', 'Bong',
'Perk', 'Green', 'Ken', 'Adams'];
let hours = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10];
let designation = ['Manager', 'Engineer 1', 'Engineer 2', 'Developer',
'Tester'];

```

```

let status = ['Completed', 'Open', 'In Progress', 'Review', 'Testing']
let data = (count) => {
  let result = [];
  for (let i = 0; i < count; i++) {
    result.push({
      TaskID: i + 1,
      Engineer: names[Math.round(Math.random() * names.length)] ||
names[0],
      Designation: designation[Math.round(Math.random() *
designation.length)] || designation[0],
      Estimation: hours[Math.round(Math.random() * hours.length)] ||
hours[0],
      Status: status[Math.round(Math.random() * status.length)] ||
status[0]
    });
  }
  return result;
};
export default {
  data() {
    return {
      data: data(1000),
      options: { pageSize: 50 },
      infiniteOptions: { enableCache: true }
    };
  },
  provide: {
    grid: [InfiniteScroll]
  }
}
</script>
<style>
  @import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/grid/virtualscroll/default-cs3" %}

### Limitations for Infinite Scrolling

- Due to the element height limitation in browsers, the maximum number of records loaded by the grid is limited due to the browser capability.
- Initial loading rows total height must be greater than the viewport height.
- Cell selection will not be persisted in cache mode.
- Infinite scrolling is not compatible with batch editing, detail template and hierarchy features.
- The group records cannot be collapsed in cache mode.
- The aggregated information and total group items are displayed based on the current view items. To get these information regardless of the view items, refer to the [Group with Page](#) topic.
- Programmatic selection using the [selectRows](#) and [selectRow](#) method is not supported in infinite scrolling.

## Selection

### Selection in Vue Grid component

Selection provides an option to highlight a row or cell or column. Selection can be done through simple Mouse down or Arrow keys. To disable selection in the Grid, set the [allowSelection](#) to false.

The grid supports two types of selection that can be set by using the [selectionSettings.type](#). They are:

- **Single** - The **Single** value is set by default. Allows you to select only a single row or cell or column.
- [Link to the Video](#) - Allows you to select multiple rows or cells or columns. To perform the multi-selection, press and hold CTRL key and click the desired rows or cells or columns. To select range of rows or cells or columns, press and hold the SHIFT key and click the rows or cells or columns.

To get start quickly with Selection Options, you can check on this video:

### APP.VUE

```
<template>
  <div id="app">
    <ejs-grid :dataSource='data' :selectionSettings='selectionOptions'
height='315px'>
      <e-columns>
        <e-column field='OrderID' headerText='Order ID'
textAlign='Right' width=120></e-column>
        <e-column field='CustomerID' headerText='Customer ID'
width=150></e-column>
        <e-column field='ShipCity' headerText='Ship City'
width=150></e-column>
        <e-column field='ShipName' headerText='Ship Name'
width=150></e-column>
      </e-columns>
    </ejs-grid>
  </div>
</template>
<script>
import Vue from "vue";
import { GridPlugin } from "@syncfusion/ej2-vue-grids";
import { data } from './datasource.js';
Vue.use(GridPlugin);
export default {
  data() {
    return {
      data: data,
      selectionOptions: { type: 'Multiple' }
    };
  }
}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
</style>
```

{% previewsample "page.domainurl/code-snippet/grid/select/default-cs13" %}

*Selection mode*

Grid supports three types of selection mode which can be set by using [selectionSettings.mode](#). They are:

- **Row** - The **row** value is set by default. Allows you to select rows only.
- **Cell** - Allows you to select cells only.
- **Both** - Allows you to select rows and cells at the same time.



**APP.VUE**

```
<template>
  <div id="app">
    <ejs-grid :dataSource='data' :selectionSettings='selectionOptions'
height='315px'>
      <e-columns>
        <e-column field='OrderID' headerText='Order ID'
textAlign='Right' width=120></e-column>
        <e-column field='CustomerID' headerText='Customer ID'
width=150></e-column>
        <e-column field='ShipCity' headerText='Ship City'
width=150></e-column>
        <e-column field='ShipName' headerText='Ship Name'
width=150></e-column>
      </e-columns>
    </ejs-grid>
  </div>
</template>
<script>
import Vue from "vue";
import { GridPlugin } from "@syncfusion/ej2-vue-grids";
import { data } from './datasource.js';
Vue.use(GridPlugin);
export default {
  data() {
    return {
      data: data,
      selectionOptions: { mode: 'Both' }
    };
  }
}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
</style>
```

{% previewsample "page.domainurl/code-snippet/grid/select/default-cs14" %}

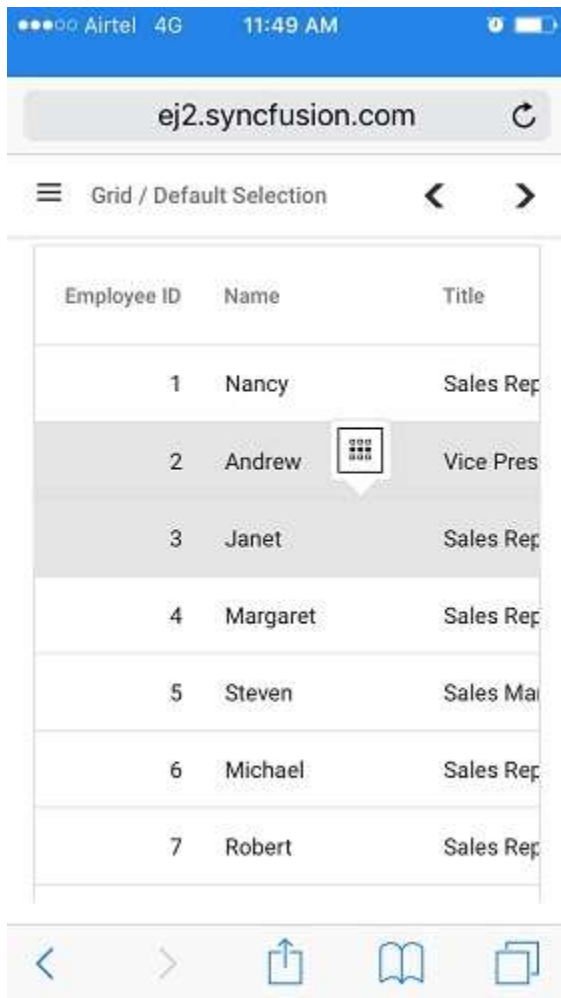
*Touch interaction*

When you tap Grid row on touch screen devices, then the tapped row is selected. Also, it will show a

popup  for multi-row-selection. To select multiple rows or cells, tap the popup  and then tap the desired rows or cells.

For multi-selection, It requires the selection [type](#) to be **Multiple**.

The following screenshot represents a Grid touch selection in the device.



*See Also*

- [How to enable/disable the Grid toolbar and contextMenu items based on the Grid row selection in Vue Grid](#)

Row selection in Vue Grid component

*Select row at initial rendering*

To select a row at initial rendering, set the [selectedRowIndex](#) value.

#### APP.VUE

```
<template>
  <div id="app">
    <ejs-grid :dataSource='data' :selectedRowIndex=1 height='315px'>
      <e-columns>
        <e-column field='OrderID' headerText='Order ID'
textAlign='Right' width=120></e-column>
        <e-column field='CustomerID' headerText='Customer ID'
width=150></e-column>
      </e-columns>
    </ejs-grid>
  </div>
</template>
```

```

        <e-column field='ShipCity' headerText='Ship City'
width=150></e-column>
        <e-column field='ShipName' headerText='Ship Name'
width=150></e-column>
    </e-columns>
</ejs-grid>
</div>
</template>
<script>
import Vue from "vue";
import { GridPlugin } from "@syncfusion/ej2-vue-grids";
import { data } from './datasource.js';
Vue.use(GridPlugin);
export default {
    data() {
        return {
            data: data
        };
    }
}
</script>
<style>
    @import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/grid/select/default-cs7" %}

#### [Get selected row indexes](#)

You can get the selected row indexes by using the [getSelectedRowIndexes](#) method.

#### **APP.VUE**

```

<template>
    <div id="app">
        <ejs-grid ref='grid' :dataSource='data' height='315px'
:rowSelected='rowSelected'>
            <e-columns>
                <e-column field='OrderID' headerText='Order ID'
textAlign='Right' width=120></e-column>
                <e-column field='CustomerID' headerText='Customer ID'
width=150></e-column>
                <e-column field='ShipCity' headerText='Ship City'
width=150></e-column>
                <e-column field='ShipName' headerText='Ship Name'
width=150></e-column>
            </e-columns>
        </ejs-grid>
    </div>
</template>
<script>
import Vue from "vue";
import { GridPlugin } from "@syncfusion/ej2-vue-grids";
import { data } from './datasource.js';
Vue.use(GridPlugin);
export default {
    data() {

```

```

    return {
      data: data
    };
  },
  methods: {
    rowSelected: function(args) {
      let selectedrowindex = this.$refs.grid.getSelectedRowIndex(); //
Get the selected row indexes.
      alert(selectedrowindex); // To alert the selected row indexes.
      let selectedrecords = this.$refs.grid.getSelectedRecords(); // Get
the selected records.
    }
  }
}
</script>
<style>
  @import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/grid/select/default-cs8" %}

#### *Simple multiple row selection*

You can select multiple rows by clicking on rows one by one. This will not deselect the previously selected rows. To deselect the previously selected row, you can click on the selected row. You can enable this behavior by using [selectionSettings.enableSimpleMultiRowSelection](#) property.

#### **APP.VUE**

```

<template>
  <div id="app">
    <ejs-grid :dataSource='data' :selectionSettings='selectionOptions'
height='315px'>
      <e-columns>
        <e-column field='OrderID' headerText='Order ID'
textAlign='Right' width=120></e-column>
        <e-column field='CustomerID' headerText='Customer ID'
width=150></e-column>
        <e-column field='ShipCity' headerText='Ship City'
width=150></e-column>
        <e-column field='ShipName' headerText='Ship Name'
width=150></e-column>
      </e-columns>
    </ejs-grid>
  </div>
</template>
<script>
import Vue from "vue";
import { GridPlugin } from "@syncfusion/ej2-vue-grids";
import { data } from './datasource.js';
Vue.use(GridPlugin);
export default {
  data() {
    return {
      data: data,
      selectionOptions: { type: 'Multiple', enableSimpleMultiRowSelection:
true }
    }
  }
}

```



```

    };
  }
}
</script>
<style>
  @import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/grid/select/default-cs9" %}

### *Toggle selection*

The Toggle selection allows to perform selection and unselection of the particular row or cell or column. To enable toggle selection, set [enableToggle](#) property of the selectionSettings as true. If you click on the selected row or cell or column then it will be unselected and vice versa.

### **APP.VUE**

```

<template>
  <div id="app">
    <ejs-grid :dataSource='data' :selectionSettings='selectionOptions'
height='315px'>
      <e-columns>
        <e-column type='checkbox' width='50'></e-column>
        <e-column field='OrderID' headerText='Order ID'
textAlign='Right' width=120></e-column>
        <e-column field='CustomerID' headerText='Customer ID'
width=150></e-column>
        <e-column field='ShipCity' headerText='Ship City'
width=150></e-column>
        <e-column field='ShipName' headerText='Ship Name'
width=150></e-column>
      </e-columns>
    </ejs-grid>
  </div>
</template>
<script>
import Vue from "vue";
import { GridPlugin } from "@syncfusion/ej2-vue-grids";
import { data } from '../datasource.js';
Vue.use(GridPlugin);
export default {
  data() {
    return {
      data: data,
      selectionOptions: { enableToggle: true }
    };
  }
}
</script>
<style>
  @import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/grid/select/default-cs10" %}

If multi selection is enabled, then first click on any selected row (without pressing Ctrl key), it will clear the multi selection and in second click on the same row, it will be unselected.

#### *Clear selection programmatically*

You can clear the Grid selection programmatically by using the [clearSelection](#) method.

In the demo below, we initially selected the third row using [selectedRowIndex](#). You can clear this selection by calling the [clearSelection](#) method in the button click event.

#### **APP.VUE**

```
<template>
  <div id="app">
    <ejs-button id='clear' cssClass='e-flat' @click.native='clear'>Clear
    Selection</ejs-button>
    <ejs-grid ref='grid' :dataSource='data' :allowPaging='true'
    :allowSelection='true' :pageSettings='pageSettings' :selectedRowIndex='2'
    :selectionSettings='selectionOptions'>
      <e-columns>
        <e-column field='OrderID' headerText='Order ID'
        :isPrimaryKey='true' textAlign='Right' width=120></e-column>
        <e-column field='CustomerID' headerText='Customer ID'
        width=120></e-column>
        <e-column field='ShipCountry' headerText='Ship Country'
        width=130></e-column>
        <e-column field='Freight' headerText='Freight' format='C2'
        width=100></e-column>
      </e-columns>
    </ejs-grid>
  </div>
</template>
<script>
import Vue from "vue";
import { GridPlugin, Page, Selection } from "@syncfusion/ej2-vue-grids";
import { ButtonPlugin } from "@syncfusion/ej2-vue-buttons";
import { cdata } from './datasource.js';
Vue.use(GridPlugin);
Vue.use(ButtonPlugin);
export default {
  data() {
    return {
      data: cdata,
      selectionOptions: { type: 'Multiple' },
      pageSettings: { pageSize: 5 }
    };
  },
  methods: {
    clear: function (args) {
      var gridObj = this.$refs.grid.ej2Instances;
      gridObj.clearSelection();
    }
  },
  provide: {
    grid: [Page, Selection]
  }
}
</script>
```

```
<style>
@import
"https://ej2.syncfusion.com/vue/documentation./node_modules/@syncfusion/ej2-
vue-grids/styles/material.css";
</style>
```

{% previewsample "page.domainurl/code-snippet/grid/select/default-cs11" %}

*Get selected records on various pages*

Enabling the `selectionSettings.persistSelection` property will persist the selection in all Grid operations.

So the selection will be maintained on every page even after navigating to another page.

You can get the selected records using the [getSelectedRecords](#) method.

### APP.VUE

```
<template>
  <div id="app">
    <ejs-button id='select' cssClass='e-flat'
    @click.native='select'>Selected Records</ejs-button>
    <ejs-grid ref='grid' :dataSource='data' :allowPaging='true'
    :pageSettings='pageSettings' :selectedRowIndex='2'
    :selectionSettings='selectionOptions'>
      <e-columns>
        <e-column type='checkbox' headerText='Check Box'
width=50></e-column>
        <e-column field='OrderID' headerText='Order ID'
:isPrimaryKey='true' textAlign='Right' width=120></e-column>
        <e-column field='CustomerID' headerText='Customer ID'
width=120></e-column>
        <e-column field='ShipCountry' headerText='Ship Country'
width=130></e-column>
        <e-column field='Freight' headerText='Freight' format='C2'
width=100></e-column>
      </e-columns>
    </ejs-grid>
  </div>
</template>
<script>
import Vue from "vue";
import { GridPlugin, Page, Selection } from "@syncfusion/ej2-vue-grids";
import { ButtonPlugin, CheckBoxPlugin } from "@syncfusion/ej2-vue-buttons";
import { cdata } from './datasource.js';
Vue.use(GridPlugin);
Vue.use(ButtonPlugin, CheckBoxPlugin);
export default {
  data() {
    return {
      data: cdata,
      selectionOptions: { persistSelection: true },
      pageSettings: { pageSize: 5 }
    };
  },
  methods: {
    select: function (args) {
      var gridObj = this.$refs.grid.ej2Instances;
```

```

        let selectedrecords = gridObj.getSelectedRecords(); // get the
selected records.
        let selectedRecordsCount = selectedrecords.length;
        alert(selectedRecordsCount); // to alert the selected records count.
    }
},
provide: {
    grid: [Page, Selection]
}
}
</script>
<style>
@import
"https://ej2.syncfusion.com/vue/documentation./node_modules/@syncfusion/ej2-
vue-grids/styles/material.css";
@import '../node_modules/@syncfusion/ej2-buttons/styles/material.css';
</style>

```

{% previewsample "page.domainurl/code-snippet/grid/select/default-cs12" %}

To persist the grid selection, it is necessary to define any one of the columns as a primary key using the `columns.isPrimaryKey` property.

#### Cell selection in Vue Grid component

Cell Selection can be done through simple Mouse down or Arrow keys(up, down, left and right).

Grid supports two types of cell selection mode which can be set by using

[selectionSettings.cellSelectionMode](#). They are:

- **Flow** - The **Flow** value is set by default. Select range of cells between the start index and end index which includes in between cells of rows.
- **Box** - Select range of cells within the start and end column indexes which includes in between cells of rows within the range.

#### APP.VUE

```

<template>
  <div id="app">
    <ejs-grid :dataSource='data' :selectionSettings='selectionOptions'
height='315px'>
      <e-columns>
        <e-column field='OrderID' headerText='Order ID'
textAlign='Right' width=120></e-column>
        <e-column field='CustomerID' headerText='Customer ID'
width=150></e-column>
        <e-column field='ShipCity' headerText='Ship City'
width=150></e-column>
        <e-column field='ShipName' headerText='Ship Name'
width=150></e-column>
      </e-columns>
    </ejs-grid>
  </div>
</template>
<script>

```

```
import Vue from "vue";
import { GridPlugin } from "@syncfusion/ej2-vue-grids";
import { data } from './datasource.js';
Vue.use(GridPlugin);
export default {
  data() {
    return {
      data: data,
      selectionOptions: { cellSelectionMode: 'Box', type: 'Multiple', mode:
'Cell' }
    };
  }
}
</script>
<style>
  @import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
</style>
```

{% previewsample "page.domainurl/code-snippet/grid/select/default-cs1" %}

Cell Selection requires the [selectionSettings.mode](#) to be **Cell** or **Both** and [type](#) should be **Multiple**.

#### *Toggle selection*

The Toggle selection allows to perform selection and unselection of the particular row or cell or column. To enable toggle selection, set [enableToggle](#) property of the selectionSettings as true. If you click on the selected row or cell or column then it will be unselected and vice versa.

#### **APP.VUE**

```
<template>
  <div id="app">
    <ejs-grid :dataSource='data' :selectionSettings='selectionOptions'
height='315px'>
      <e-columns>
        <e-column type='checkbox' width='50'></e-column>
        <e-column field='OrderID' headerText='Order ID'
textAlign='Right' width=120></e-column>
        <e-column field='CustomerID' headerText='Customer ID'
width=150></e-column>
        <e-column field='ShipCity' headerText='Ship City'
width=150></e-column>
        <e-column field='ShipName' headerText='Ship Name'
width=150></e-column>
      </e-columns>
    </ejs-grid>
  </div>
</template>
<script>
import Vue from "vue";
import { GridPlugin } from "@syncfusion/ej2-vue-grids";
import { data } from './datasource.js';
Vue.use(GridPlugin);
export default {
  data() {
    return {
      data: data,
```

```

        selectionOptions: { enableToggle: true }
    };
}
</script>
<style>
    @import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/grid/select/default-cs2" %}

If multi selection is enabled, then first click on any selected row (without pressing Ctrl key), it will clear the multi selection and in second click on the same row, it will be unselected.

### Column selection in Vue Grid component

Column selection can be done through simple mouse down or arrow keys.

You can enable column selection by setting the [selectionSettings.allowColumnSelection](#) property as true.

### APP.VUE

```

<template>
    <div id="app">
        <ejs-grid :dataSource='data' :selectionSettings='selectionOptions'
height='315px'>
            <e-columns>
                <e-column field='OrderID' headerText='Order ID'
textAlign='Right' width=120></e-column>
                <e-column field='CustomerID' headerText='Customer ID'
width=150></e-column>
                <e-column field='ShipCity' headerText='Ship City'
width=150></e-column>
                <e-column field='ShipName' headerText='Ship Name'
width=150></e-column>
            </e-columns>
        </ejs-grid>
    </div>
</template>
<script>
import Vue from "vue";
import { GridPlugin } from "@syncfusion/ej2-vue-grids";
import { data } from './datasource.js';
Vue.use(GridPlugin);
export default {
    data() {
        return {
            data: data,
            selectionOptions: { allowColumnSelection: true, type: 'Multiple' }
        };
    }
}
</script>
<style>
    @import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
</style>

```

```
{% previewsample "page.domainurl/code-snippet/grid/select/default-cs6" %}
```

### Check box selection in Vue Grid component

Checkbox Selection provides an option to select multiple Grid records with help of checkbox in each row.

To render checkbox in each grid row, you need to use checkbox column with type as **CheckBox** using column [type](#) property.

#### APP.VUE

```
<template>
  <div id="app">
    <ejs-grid :dataSource='data' height='315px'>
      <e-columns>
        <e-column type='checkbox' width='50'></e-column>
        <e-column field='OrderID' headerText='Order ID'
textAlign='Right' width=120></e-column>
        <e-column field='CustomerID' headerText='Customer ID'
width=150></e-column>
        <e-column field='ShipCity' headerText='Ship City'
width=150></e-column>
        <e-column field='ShipName' headerText='Ship Name'
width=150></e-column>
      </e-columns>
    </ejs-grid>
  </div>
</template>
<script>
import Vue from "vue";
import { GridPlugin } from "@syncfusion/ej2-vue-grids";
import { data } from './datasource.js';
Vue.use(GridPlugin);
export default {
  data() {
    return {
      data: data
    };
  }
}
</script>
<style>
  @import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
</style>
```

```
{% previewsample "page.domainurl/code-snippet/grid/select/default-cs3" %}
```

By default selection is allowed by clicking a grid row or checkbox in that row. To allow Selection only through checkbox, you can set [selectionSettings.checkboxOnly](#) property to true.

Selection can be persisted on all the operations using [selectionSettings.persistSelection](#) property. For persisting selection on the Grid, any one of the column should be defined as a primary key using [columns.isPrimaryKey](#) property.

*Checkbox selection mode*

In checkbox selection, selection can also be done by clicking on rows. This selection provides two types of Checkbox Selection mode which can be set by using the following API, [selectionSettings.checkboxMode](#). The modes are;

- **Default:** This is the default value of the checkboxMode. In this mode, user can select multiple rows by clicking rows one by one.
- **ResetOnRowClick:** In ResetOnRowClick mode, when user clicks on a row it will reset previously selected row. Also you can perform multiple-selection in this mode by press and hold CTRL key and click the desired rows. To select range of rows, press and hold the SHIFT key and click the rows.

**APP.VUE**

```
<template>
  <div id="app">
    <ejs-grid :dataSource='data' :selectionSettings='selectionOptions'
height='315px'>
      <e-columns>
        <e-column type='checkbox' width='50'></e-column>
        <e-column field='OrderID' headerText='Order ID'
textAlign='Right' width=120></e-column>
        <e-column field='CustomerID' headerText='Customer ID'
width=150></e-column>
        <e-column field='ShipCity' headerText='Ship City'
width=150></e-column>
        <e-column field='ShipName' headerText='Ship Name'
width=150></e-column>
      </e-columns>
    </ejs-grid>
  </div>
</template>
<script>
import Vue from "vue";
import { GridPlugin } from "@syncfusion/ej2-vue-grids";
import { data } from './datasource.js';
Vue.use(GridPlugin);
export default {
  data() {
    return {
      data: data,
      selectionOptions: { checkboxMode: 'ResetOnRowClick' }
    };
  }
}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
</style>
```

{% previewsample "page.domainurl/code-snippet/grid/select/default-cs4" %}



### Prevent specific rows from being selected in checkbox selection

You can prevent specific rows from being selected in the checkbox selection mode by hiding the checkboxes using the [rowDataBound](#) event. You achieve this by setting the [isSelectable](#) argument as false in the [rowDataBound](#) event based on certain conditions as per the needs of the application.

In the following sample, the selection of specific rows has been prevented based on the [isSelectable](#) argument in the [rowDataBound](#) event.

#### APP.VUE

```
<template>
  <div id="app">
    <ejs-grid :dataSource='data' :allowPaging="true"
      :allowFiltering='true' :allowSelection="true"
      :selectionSettings='selectionOptions' :editSettings='editSettings'
      :pageSettings='pageSettings' :toolbar='toolbar'
      :filterSettings='filterOptions' :rowDataBound='rowDataBound' height='185px'>
      <e-columns>
        <e-column type='checkbox' width='120'></e-column>
        <e-column field='List' headerText='List' width=120></e-
column>
        <e-column field='OrderID' headerText='Order ID'
textAlign='Right' :isPrimaryKey='true' width=120></e-column>
        <e-column field='CustomerID' headerText='Customer ID'
width=150></e-column>
        <e-column field='EmployeeID' headerText='Employee ID'
width=150></e-column>
        <e-column field='ShipCity' headerText='Ship City'
width=150></e-column>
      </e-columns>
    </ejs-grid>
  </div>
</template>
<script>
import Vue from "vue";
import { GridPlugin, Edit, Page, Toolbar, Filter } from "@syncfusion/ej2-
vue-grids";
import { cdata } from './datasource.js';
Vue.use(GridPlugin);
for (let i = 0; i < cdata.length; i++) {
  cdata[i]["List"] = i + 1;
}
export default {
  data() {
    return {
      data: cdata,
      selectionOptions: { persistSelection: true },
      editSettings: { allowEditing: true, allowAdding: true, allowDeleting:
true, mode: 'Normal' },
      pageSettings: { pageSize: 5 },
      toolbar: ['Add', 'Edit', 'Delete', 'Update', 'Cancel', 'Search'],
      filterOptions: {
        type: 'CheckBox'
      }
    }
  };
},
```

```

methods: {
  rowDataBound(args) {
    args.isSelectable = args.data.List % 5 === 0;
  }
},
provide: {
  grid: [Edit, Page, Toolbar, Filter]
}
}
</script>
<style>
  @import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/grid/select/default-cs5" %}

## Aggregates

### Aggregates in Vue Grid component

Aggregate values are displayed in the footer, group footer and group caption of Grid. It can be configured through `e-aggregates` directive. The [field](#) and [type](#) are the minimum properties required to represent an aggregate column.

To use aggregate feature, you need to inject the `Aggregate` module into the `provide` section.

By default, the aggregate value can be displayed in footer, group and caption cells, to show the aggregate value in any of these cells, use the [footerTemplate](#), [groupFooterTemplate](#) and [Link to the Video](#) properties.

To get start quickly with Aggregate Options, you can check on this video:

### Built-in aggregate types

Aggregate type must be specified in [type](#) property to configure an aggregate column.

The built-in aggregates are,

- Sum
- Average
- Min
- Max
- Count
- TrueCount
- FalseCount

\* Multiple aggregates can be used for an aggregate column by setting the [type](#) property with an array of aggregate type.

\* Multiple types for a column is supported only when one of the aggregate templates is used.

### Footer aggregate in Vue Grid component

Footer aggregate value is calculated from all the rows and it can be displayed in footer cells. Use [footerTemplate](#) to render the aggregate value in footer cells.

### **APP.VUE**

```

<template>
  <div id="app">
    <ejs-grid :dataSource='data' height='210px'>
      <e-columns>
        <e-column field='OrderID' headerText='Order ID'
textAlign='right' width=120></e-column>
        <e-column field='CustomerID' headerText='Customer ID'
width=150></e-column>
        <e-column field='Freight' width=150></e-column>
        <e-column field='ShipName' headerText='Ship Name'
width=150></e-column>
      </e-columns>
      <e-aggregates>
        <e-aggregate>
          <e-columns>
            <e-column type="Sum" field="Freight"
:footerTemplate='footerSum'></e-column>
          </e-columns>
        </e-aggregate>
        <e-aggregate>
          <e-columns>
            <e-column type="Max" field="Freight"
:footerTemplate='footerMax'></e-column>
          </e-columns>
        </e-aggregate>
      </e-aggregates>
    </ejs-grid>
  </div>
</template>
<script>
import Vue from "vue";
import { GridPlugin, Aggregate } from "@syncfusion/ej2-vue-grids";
import { data } from './datasource.js';
Vue.use(GridPlugin);
export default {
  data() {
    return {
      data: data,
      footerSum: function () {
        return { template : Vue.component('sumTemplate', {
          template: `<span>Sum: {{data.Sum}}</span>`,
          data () {return { data: {}}};
        })
      },
      footerMax: function () {
        return { template : Vue.component('maxTemplate', {
          template: `<span>Max: {{data.Max}}</span>`,
          data () {return { data: {}}};
        })
      }
    }
  },
  provide: {
    grid: [Aggregate]
  }
};

```

```

}
</script>
<style>
  @import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/grid/aggregates/default-cs2" %}

The aggregate values must be accessed inside the template using their corresponding [type](#) name.

#### *How to format aggregate value*

You can format the aggregate value result by using the [format](#) property.

### **APP.VUE**

```

<template>
  <div id="app">
    <ejs-grid :dataSource='data' height='210px'>
      <e-columns>
        <e-column field='OrderID' headerText='Order ID'
textAlign='right' width=120></e-column>
        <e-column field='CustomerID' headerText='Customer ID'
width=150></e-column>
        <e-column field='Freight' width=150></e-column>
        <e-column field='ShipName' headerText='Ship Name'
width=150></e-column>
      </e-columns>
      <e-aggregates>
        <e-aggregate>
          <e-columns>
            <e-column type="Sum" field="Freight" format="C2"
:footerTemplate='footerSum'></e-column>
          </e-columns>
        </e-aggregate>
        <e-aggregate>
          <e-columns>
            <e-column type="Max" field="Freight" format="C2"
:footerTemplate='footerMax'></e-column>
          </e-columns>
        </e-aggregate>
      </e-aggregates>
    </ejs-grid>
  </div>
</template>
<script>
import Vue from "vue";
import { GridPlugin, Aggregate } from "@syncfusion/ej2-vue-grids";
import { data } from './datasource.js';
Vue.use(GridPlugin);
export default {
  data() {
    return {
      data: data,
      footerSum: function () {
        return { template : Vue.component('sumTemplate', {
          template: `<span>Sum: {{data.Sum}}</span>`,
          data () {return { data: {}}};

```

```

        })
      }
    },
    footerMax: function () {
      return { template : Vue.component('maxTemplate', {
        template: `<span>Max: {{data.Max}}</span>`,
        data () {return { data: {}}};
      })
    }
  }
};
},
provide: {
  grid: [Aggregate]
}
}
</script>
<style>
  @import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/grid/aggregates/default-cs3" %}

#### *How to place aggregates on top of the Grid*

By default, the aggregated values are placed at the bottom of the footer section. It is possible to place the aggregated values at the top of the header. This is achieved by using the [dataBound](#) event, [getHeaderContent](#), and [getFooterContent](#) methods of the Grid.

In the following sample, the footer element is appended to the header element using the [getHeaderContent](#) and [getFooterContent](#) methods in the [dataBound](#) event.

#### **APP.VUE**

```

<template>
  <div id="app">
    <ejs-grid ref='grid' :dataSource='data' height='210px'
    :dataBound="dataBound">
      <e-columns>
        <e-column field='OrderID' headerText='Order ID'
        textAlign='right' width=120></e-column>
        <e-column field='CustomerID' headerText='Customer ID'
        width=150></e-column>
        <e-column field='Freight' width=150></e-column>
        <e-column field='ShipName' headerText='Ship Name'
        width=150></e-column>
      </e-columns>
      <e-aggregates>
        <e-aggregate>
          <e-columns>
            <e-column type="Sum" field="Freight"
            :footerTemplate='footerSum'></e-column>
          </e-columns>
        </e-aggregate>
        <e-aggregate>
          <e-columns>

```

```

        <e-column type="Max" field="Freight"
:footerTemplate='footerMax'></e-column>
    </e-columns>
  </e-aggregate>
</e-aggregates>
</ejs-grid>
</div>
</template>
<script>
import Vue from "vue";
import { GridPlugin, Aggregate } from "@syncfusion/ej2-vue-grids";
import { data } from '../datasource.js';
Vue.use(GridPlugin);
export default {
  data() {
    return {
      data: data,
      footerSum: function () {
        return { template : Vue.component('sumTemplate', {
          template: `<span>Sum: {{data.Sum}}</span>`,
          data () {return { data: {}}};
        })
      },
      footerMax: function () {
        return { template : Vue.component('maxTemplate', {
          template: `<span>Max: {{data.Max}}</span>`,
          data () {return { data: {}}};
        })
      }
    };
  },
  methods: {
    dataBound: function (args) {

this.$refs.grid.ej2Instances.getHeaderContent().append(this.$refs.grid.ej2In
stances.getFooterContent());
    },
    provide: {
      grid: [Aggregate]
    }
  }
</script>
<style>
@import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/grid/aggregates/default-cs4" %}

### Group and caption aggregate in Vue Grid component

Group and caption aggregate values are calculated from the current group items. If

[groupFooterTemplate](#) is provided then the aggregate values can be displayed in the group footer cells

and if [groupCaptionTemplate](#) is provided then aggregate values can be displayed in the group caption cells.

### APP.VUE

```
<template>
  <div id="app">
    <ejs-grid :dataSource='data' height='290px' :allowGrouping="true"
    :groupSettings="groupOptions">
      <e-columns>
        <e-column field='OrderID' headerText='Order ID'
        textAlign='right' width=120></e-column>
        <e-column field='CustomerID' headerText='Customer ID'
        width=150></e-column>
        <e-column field='OrderDate' headerText='Order Date'
        format='yMd' width=120 type='date'></e-column>
        <e-column field='Freight' format='C2' width=150></e-column>
        <e-column field='ShipCountry' headerText='Ship Country'
        width=150></e-column>
      </e-columns>
      <e-aggregates>
        <e-aggregate>
          <e-columns>
            <e-column type="Sum" field="Freight" format="C2"
            :groupFooterTemplate = 'footerSum'></e-column>
          </e-columns>
        </e-aggregate>
        <e-aggregate>
          <e-columns>
            <e-column type="Average" field="Freight" format="C2"
            :groupCaptionTemplate = 'footerAvg'></e-column>
          </e-columns>
        </e-aggregate>
      </e-aggregates>
    </ejs-grid>
  </div>
</template>
<script>
import Vue from "vue";
import { GridPlugin, Group, Aggregate } from "@syncfusion/ej2-vue-grids";
import { data } from './datasource.js';
Vue.use(GridPlugin);
export default {
  data() {
    return {
      data: data,
      groupOptions: {showDropArea: false, columns: ['ShipCountry'] },
      footerSum: function () {
        return { template : Vue.component('sumTemplate', {
          template: `<span>Sum: {{data.Sum}}</span>`,
          data () {return { data: {}}};
        })
      },
      footerAvg: function () {
        return { template : Vue.component('maxTemplate', {
          template: `<span>Average: {{data.Average}}</span>`,
```

```

        data () {return { data: {} };}
    })
}
};
},
provide : {
    grid: [Group, Aggregate]
}
}
</script>
<style>
    @import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/grid/aggregates/default-cs5" %}

\* Use the template reference property name as `groupFooterTemplate` to specify the group footer template and as `groupCaptionTemplate` to specify the group caption template.

\* The aggregate values must be accessed inside the template using their corresponding [type](#) name.

#### Custom aggregate in Vue Grid component

Sometimes you can have a scenario to calculate aggregate value using your own aggregate function, we can achieve this behavior using the custom aggregate option. To use custom aggregation, specify the [type](#) as `Custom` and provide the custom aggregate function in the [customAggregate](#) property.

The custom aggregate function will be invoked with different arguments for Total and Group aggregations.

- **Total aggregation** - the custom aggregate function will be called with whole data and the current [AggregateColumn](#) object.
- **Group aggregation** - it will be called with current group details and the [AggregateColumn](#) object.

#### APP.VUE

```

<template>
  <div id="app">
    <ejs-grid :dataSource='data' height='268px'>
      <e-columns>
        <e-column field='OrderID' headerText='Order ID'
textAlign='right' width=120></e-column>
        <e-column field='CustomerID' headerText='Customer ID'
width=150></e-column>
        <e-column field='Freight' format='C2' width=150></e-column>
        <e-column field='ShipCountry' headerText='Ship Country'
width=150></e-column>
      </e-columns>
      <e-aggregates>
        <e-aggregate>
          <e-columns>
            <e-column columnName="ShipCountry" type="Custom"
:customAggregate="customAggregateFn" :footerTemplate='footerTemp'></e-
column>
          </e-columns>
        </e-aggregate>
      </e-aggregates>
    </ejs-grid>
  </div>
</template>

```



```

        </e-aggregate>
    </e-aggregates>
</ejs-grid>
</div>
</template>
<script>
import Vue from "vue";
import { GridPlugin, Aggregate } from "@syncfusion/ej2-vue-grids";
import { data } from './datasource.js';
Vue.use(GridPlugin);
export default {
    data() {
        return {
            data: data,
            footerTemp: function () {
                return { template : Vue.component('footerTemplate', {
                    template: `<span>Brazil Count: {{data.Custom}}</span>`,
                    data () {return { data: {}}};
                })
            }
        };
    },
    methods: {
        customAggregateFn : function (data) {
            return data.result.filter((item) => item.ShipCountry ===
'Brazil').length;
        },
        provide: {
            grid: [Aggregate]
        }
    }
}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/grid/aggregates/default-cs1" %}

To access the custom aggregate value inside template, use the key as **Custom**.

### Reactive aggregate in Vue Grid component

#### [Auto update aggregate value in batch editing](#)

When using batch editing, the aggregate values will be refreshed on every cell save. The footer, group footer, and group caption aggregate values will be refreshed.

Adding a new record to the grouped grid will not refresh the aggregate values.

### **APP.VUE**

```

<template>
    <div id="app">
        <ejs-grid :dataSource='data' height='290px' allowPaging='true'
allowGrouping='true' :groupSettings='groupOptions' :toolbar='toolbarOptions'
:editSettings='editSettings'>

```

```

        <e-columns>
            <e-column field='OrderID' headerText='Order ID'
isPrimaryKey='true' textAlign='right' width=120></e-column>
            <e-column field='CustomerID' headerText='Customer ID'
width=150></e-column>
            <e-column field='OrderDate' headerText='Order Date'
format='yMd' width=120 type='date'></e-column>
            <e-column field='Freight' format='C2' editType=
'numericedit' width=150 ></e-column>
            <e-column field='ShipCountry' headerText='Ship Country'
width=150></e-column>
        </e-columns>
        <e-aggregates>
            <e-aggregate>
                <e-columns>
                    <e-column type="Sum" field="Freight" format="C2"
:footerTemplate = 'footerSum'></e-column>
                </e-columns>
            </e-aggregate>
            <e-aggregate>
                <e-columns>
                    <e-column type="Sum" field="Freight" format="C2"
:groupFooterTemplate = 'groupFooterSum'></e-column>
                </e-columns>
            </e-aggregate>
            <e-aggregate>
                <e-columns>
                    <e-column type="Average" field="Freight" format="C2"
:groupCaptionTemplate = 'footerAvg'></e-column>
                </e-columns>
            </e-aggregate>
        </e-aggregates>
    </ejs-grid>
</div>
</template>
<script>
import Vue from "vue";
import { GridPlugin, Page, Group, Aggregate, Toolbar, Edit } from
"@syncfusion/ej2-vue-grids";
import { data } from './datasource.js';
Vue.use(GridPlugin);
export default {
    data() {
        return {
            data: data,
            groupOptions: { showDropArea: false, columns: ['ShipCountry'] },
            toolbarOptions : ['Delete', 'Update', 'Cancel'],
            editSettings : { allowEditing: true, allowDeleting: true, mode:
'Batch' },
            footerSum: function () {
                return { template : Vue.component('sumTemplate', {
                    template: `<span>Sum: {{data.Sum}}</span>`,
                    data () {return { data: {}}};
                })
            },
            groupFooterSum: function () {

```

```

        return { template : Vue.component('sumTemplate', {
            template: `<span>Sum: {{data.Sum}}</span>`,
            data () {return { data: {}}};
        })
    },
    footerAvg: function () {
        return { template : Vue.component('maxTemplate', {
            template: `<span>Average: {{data.Average}}</span>`,
            data () {return { data: {}}};
        })
    }
    },
    };
    },
    provide : {
        grid: [Page, Group, Aggregate, Edit, Toolbar]
    }
}
</script>
<style>
    @import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/grid/aggregates/default-cs6" %}

#### *Refresh aggregate values in inline editing*

By default, reactive aggregate update is not supported by inline and dialog edit modes as it is not feasible to anticipate the value change event for every editor. But, you can refresh the aggregates manually in the inline edit mode using the refresh method of aggregate module.

In the following code, the input event for the Freight column editor has been registered and the aggregate value has been refreshed manually.

#### **APP.VUE**

```

<template>
    <div id="app">
        <ejs-grid ref='grid' :dataSource='data' height='290px'
        allowPaging='true' :toolbar='toolbarOptions' :editSettings='editSettings'
        :actionBegin='actionBegin'>
            <e-columns>
                <e-column field='OrderID' headerText='Order ID'
                isPrimaryKey='true' textAlign='right' width=120></e-column>
                <e-column field='CustomerID' headerText='Customer ID'
                width=150></e-column>
                <e-column field='Freight' format='C2' editType=
                'numericedit' :edit='numericParams' width=150 ></e-column>
                <e-column field='ShipCountry' headerText='Ship Country'
                width=150></e-column>
            </e-columns>
            <e-aggregates>
                <e-aggregate>
                    <e-columns>
                        <e-column type="Sum" field="Freight" format="C2"
                        :footerTemplate = 'footerSum'></e-column>
                    </e-columns>
                </e-aggregate>
            </e-aggregates>
        </ejs-grid>
    </div>
</template>

```

```

        </e-columns>
    </e-aggregate>
</e-aggregates>
</ejs-grid>
</div>
</template>
<script>
import Vue from "vue";
import { GridPlugin, Page, Aggregate, Toolbar, Edit } from "@syncfusion/ej2-vue-grids";
import { data } from './datasource.js';
Vue.use(GridPlugin);
let selectedRecord = {};
export default {
    data() {
        return {
            data: data,
            numericParams: { params: { change: this.changeFn } },
            toolbarOptions : ['Delete', 'Update', 'Cancel'],
            editSettings : { allowEditing: true, allowDeleting: true, mode:
'Inline' },
            footerSum: function () {
                return { template : Vue.component('sumTemplate', {
                    template: `<span>Sum: {{data.Sum}}</span>`,
                    data () {return { data: {}}};
                })
            }
        },
    },
    methods: {
        actionBegin: function(args){
            if(args.requestType === 'beginEdit'){
                selectedRecord = {};
                selectedRecord = args.rowData;
            };
        },
        changeFn: function(args){
            selectedRecord['Freight'] = args.value;
            let gridObj = this.$refs.grid.ej2Instances;
            gridObj.aggregateModule.refresh(selectedRecord);
        }
    },
    provide : {
        grid: [Page, Aggregate, Edit, Toolbar]
    }
}
</script>
<style>
    @import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/grid/aggregates/default-cs7" %}

## Print in Vue Grid component

To print the Grid, use the [Print](#) method from grid instance. The print option can be displayed on the [toolbar](#) by adding the **Print** toolbar item.

### APP.VUE

```
<template>
  <div id="app">
    <ejs-grid :dataSource='data' :toolbar='toolbarOptions'
height='272px'>
      <e-columns>
        <e-column field='OrderID' headerText='Order ID'
textAlign='Right' width=120></e-column>
        <e-column field='CustomerID' headerText='Customer ID'
width=150></e-column>
        <e-column field='ShipCity' headerText='Ship City'
width=150></e-column>
        <e-column field='ShipName' headerText='Ship Name'
width=150></e-column>
      </e-columns>
    </ejs-grid>
  </div>
</template>
<script>
import Vue from "vue";
import { GridPlugin, Toolbar } from "@syncfusion/ej2-vue-grids";
import { data } from './datasource.js';
Vue.use(GridPlugin);
export default {
  data() {
    return {
      data: data,
      toolbarOptions: ['Print']
    };
  },
  provide: {
    grid: [Toolbar]
  }
}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
</style>
```

{% previewsample "page.domainurl/code-snippet/grid/print/default-cs2" %}

### Page Setup

Some of the print options cannot be configured through JavaScript code. So, you have to customize the layout, paper size, and margin options using the browser page setup dialog. Please refer to the following links to know more about the browser page setup:

- [Chrome](#)
- [Firefox](#)
- [Safari](#)

- [IE](#)

### Print by external button

To print the grid from an external button, invoke the [print](#) method.

#### APP.VUE

```
<template>
  <div id="app">
    <ejs-button id='print' @click.native='print'>Print</ejs-button>
    <ejs-grid ref='grid' :dataSource='data' height='280px'>
      <e-columns>
        <e-column field='OrderID' headerText='Order ID'
textAlign='Right' width=120></e-column>
        <e-column field='CustomerID' headerText='Customer ID'
width=150></e-column>
        <e-column field='ShipCity' headerText='Ship City'
width=150></e-column>
        <e-column field='ShipName' headerText='Ship Name'
width=150></e-column>
      </e-columns>
    </ejs-grid>
  </div>
</template>
<script>
import Vue from "vue";
import { GridPlugin } from "@syncfusion/ej2-vue-grids";
import { ButtonPlugin } from "@syncfusion/ej2-vue-buttons";
import { data } from './datasource.js';
Vue.use(GridPlugin);
Vue.use(ButtonPlugin);
export default {
  data() {
    return {
      data: data
    };
  },
  methods: {
    print: function() {
      this.$refs.grid.print();
    }
  }
}
</script>
<style>
  @import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
</style>
```

```
{% previewsample "page.domainurl/code-snippet/grid/print/default-cs3" %}
```

### Print visible Page

By default, the grid prints all the pages. To print the current page alone, set the [printMode](#) to `CurrentPage`.

#### APP.VUE

```

<template>
  <div id="app">
    <ejs-grid :dataSource='data' printMode='CurrentPage'
:allowPaging='true' :pageSettings='pageOptions' :toolbar='toolbarOptions'>
      <e-columns>
        <e-column field='OrderID' headerText='Order ID'
textAlign='Right' width=120></e-column>
        <e-column field='CustomerID' headerText='Customer ID'
width=150></e-column>
        <e-column field='ShipCity' headerText='Ship City'
width=150></e-column>
        <e-column field='ShipName' headerText='Ship Name'
width=150></e-column>
      </e-columns>
    </ejs-grid>
  </div>
</template>
<script>
import Vue from "vue";
import { GridPlugin, Page, Toolbar } from "@syncfusion/ej2-vue-grids";
import { data } from './datasource.js';
Vue.use(GridPlugin);
export default {
  data() {
    return {
      data: data,
      toolbarOptions: ['Print'],
      pageOptions: { pageSize: 6 }
    };
  },
  provide: {
    grid: [Page, Toolbar]
  }
}
</script>
<style>
@import
"https://ej2.syncfusion.com/vue/documentation/node_modules/@syncfusion/ej2-
vue-grids/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/grid/print/default-cs4" %}

### Print the hierarchy grid

By default, the grid will be print the master and expanded child grids alone. you can change the print option by using the [hierarchyPrintMode](#) property. The available options are,

- |          |                                                   |  |
|----------|---------------------------------------------------|--|
| Mode     | Behavior                                          |  |
| -----    | -----                                             |  |
| Expanded | Prints the master grid with expanded child grids. |  |
| All      | Prints the master grid with all the child grids.  |  |
| None     | Prints the master grid alone.                     |  |

**APP.VUE**

```

<template>
  <div id="app">
    <ejs-grid ref='grid' :dataSource='parentData' :childGrid='childGrid'
:toolbar='["Print"]' hierarchyPrintMode='All'>
      <e-columns>
        <e-column field='EmployeeID' headerText='Employee ID'
textAlign='Right' width=120></e-column>
        <e-column field='FirstName' headerText='FirstName'
width=150></e-column>
        <e-column field='LastName' headerText='Last Name'
width=150></e-column>
        <e-column field='City' headerText='City' width=150></e-
column>
      </e-columns>
    </ejs-grid>
  </div>
</template>
<script>
import Vue from "vue";
import { GridPlugin, DetailRow, Toolbar, Page, printGridInit,
getPrintGridModel } from "@syncfusion/ej2-vue-grids";
import { data, employeeData } from './datasource.js';
import { extend } from '@syncfusion/ej2-base';
Vue.use(GridPlugin);
export default {
  data() {
    return {
      parentData: employeeData,
      childGrid: {
        dataSource: data,
        queryString: 'EmployeeID',
        columns: [
          { field: 'OrderID', headerText: 'Order ID', textAlign: 'Right',
width: 120 },
          { field: 'CustomerID', headerText: 'Customer ID', width: 150 },
          { field: 'ShipCity', headerText: 'Ship City', width: 150 },
          { field: 'ShipName', headerText: 'Ship Name', width: 150 }
        ]
      }
    },
    provide: {
      grid: [DetailRow, Toolbar]
    }
  }
}
</script>
<style>
  @import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/grid/hierarchy-grid/default-cs10" %}



### Print the master detail grid

The Grid has the option to visualize details of a record in another Grid in a master and detailed manner. By default, Grid will print the master grid alone. Instead of this, it is possible to print both the master and detail grids by using the [beforePrint](#) event of the Grid.

In the following sample, the detail grid is added to the `element` argument of the `beforePrint` event, resulting in both the master and detail grids being printed on the page.

#### APP.VUE

```
<template>
  <div id="app">
    <ejs-grid :dataSource="data" :selectedRowIndex="1" :toolbar="toolbar"
    :rowSelected="rowSelected" :beforePrint="beforePrint">
      <e-columns>
        <e-column field="ContactName" headerText="Customer Name"
width="150"></e-column>
        <e-column field="CompanyName" headerText="Company Name"
width="150"></e-column>
        <e-column field="Address" headerText="Address" width="150"></e-
column>
        <e-column field="Country" headerText="Country" width="130"></e-
column>
      </e-columns>
    </ejs-grid>
    <div class="e-statustext">Showing orders of Customer: <b
id="key"></b></div>
    <ejs-grid ref="grid" :allowSelection="false">
      <e-columns>
        <e-column field="OrderID" headerText="Order ID" width="100"
textAlign="Right"></e-column>
        <e-column field="Freight" headerText="Freight" format="C2"
width="100" type="number"></e-column>
        <e-column field="ShipName" headerText="Ship Name"
width="200"></e-column>
        <e-column field="ShipCountry" headerText="Ship Country"
width="150"></e-column>
        <e-column field="ShipAddress" headerText="Ship Address"
width="200"></e-column>
      </e-columns>
    </ejs-grid>
  </div>
</template>
<script>
import Vue from "vue";
import { GridPlugin, Toolbar, Print } from "@syncfusion/ej2-vue-grids";
import { customerData, data } from "../datasource.js";
Vue.use(GridPlugin);
export default {
  data() {
    var names = ["AROUT", "BERGS", "BLONP", "CHOPS", "ERNSH"];
    return {
      toolbar: ["Print"],
      data: customerData.filter(function (e) {
        return names.indexOf(e.CustomerID) !== -1;
      })
    };
  }
};
```

```

    },
    methods: {
      rowSelected: function (args) {
        let selectedRecord = args.data;
        this.$refs.grid.ej2Instances.dataSource = data.filter((record)
=> record.CustomerName === selectedRecord.ContactName).slice(0, 5);
        document.getElementById("key").innerHTML =
selectedRecord.ContactName;
      },
      beforePrint: function (args) {
        let customEle = document.createElement("div");
        customEle.innerHTML =
document.getElementsByClassName("e-statustext")[0].innerHTML +
this.$refs.grid.ej2Instances.element.innerHTML;
        customEle.appendChild(document.createElement("br"));
        args.element.append(customEle);
      },
    },
  },
  provide: {
    grid: [Toolbar, Print],
  }
};
</script>
<style>
  @import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/grid/print/mastergrid-cs2" %}

### Print large number of columns

By default, the browser uses A4 as page size option to print pages and to adapt the size of the page the browser print preview will auto-hide the overflowed contents. Hence grid with large number of columns will cut off to adapt the print page.

To show large number of columns when printing, adjust the scale option from print option panel based on your content size.



### Show or Hide columns while Printing

You can show a hidden column or hide a visible column while printing the grid using [toolbarClick](#) and [printComplete](#) events.

In the `toolbarClick` event, based on `args.item.id` as `grid_print`. We can show or hide columns by setting `column.visible` property to `true` or `false` respectively.

In the `printComplete` event, We have reversed the state back to the previous state.

In the below example, we have `CustomerID` as a hidden column in the grid. While printing, we have changed `CustomerID` to visible column and `ShipCity` as hidden column.

### APP.VUE

```
<template>
  <div id="app">
    <ejs-grid ref='grid' :dataSource='data' :allowPaging="true"
    :pageSettings='pageSettings' :toolbar='toolbarOptions'
    :toolbarClick='toolbarClick' :printComplete='printComplete' height='272px'>
      <e-columns>
        <e-column field='OrderID' headerText='Order ID'
        textAlign='Right' width=120></e-column>
        <e-column field='CustomerID' headerText='Customer ID'
        :visible='false' width=150></e-column>
        <e-column field='ShipCity' headerText='Ship City'
        width=150></e-column>
        <e-column field='ShipName' headerText='Ship Name'
        width=150></e-column>
      </e-columns>
    </ejs-grid>
  </div>
</template>
<script>
import Vue from "vue";
import { GridPlugin, Toolbar, Page } from "@syncfusion/ej2-vue-grids";
import { data } from './datasource.js';
Vue.use(GridPlugin);
export default {
  data() {
    return {
      data: data,
      pageSettings: { pageSizes: true, pageSize: 6 },
      toolbarOptions: ['Print']
    };
  },
  methods: {
    toolbarClick: function() {
      for (var i = 0; i < this.$refs.grid.getColumns().length; i++) {
        if (this.$refs.grid.getColumns()[i].field == "CustomerID") {
          this.$refs.grid.getColumns()[i].visible = true;
        }
        else if (this.$refs.grid.getColumns()[i].field == "ShipCity") {
          this.$refs.grid.getColumns()[i].visible = false;
        }
      }
    },
    printComplete: function() {
```

```

        for (var i = 0; i < this.$refs.grid.getColumns().length; i++) {
            if (this.$refs.grid.getColumns()[i].field == "CustomerID") {
                this.$refs.grid.getColumns()[i].visible = false;
            }
            else if (this.$refs.grid.getColumns()[i].field == "ShipCity") {
                this.$refs.grid.getColumns()[i].visible = true;
            }
        }
    },
    provide: {
        grid: [Toolbar, Page]
    }
}
</script>
<style>
    @import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/grid/print/default-cs5" %}

### Limitations of Printing Large Data

When grid contains large number of data, printing all the data at once is not a best option for the browser performance. Because to render all the DOM elements in one page will produce performance issues in the browser. It leads to browser slow down or browser hang. Grid have option to handle large number of data by Virtualization. However while printing, it is not possible to use virtualization for rows and columns.

If printing of all the data is still needed, we suggest to Export the grid to **Excel** or **CSV** or **Pdf** and then print it from another non-web based application.

### Adaptive in Vue Grid component

The Grid user interface (UI) was redesigned to provide an optimal viewing experience and improve usability on small screens. This interface will render the filter, sort, column chooser, column menu(supports only when the **rowRenderingMode** as Horizontal) and edit dialogs adaptively and have an option to render the grid row elements in the vertical direction.

### Render adaptive dialogs

When we enable the [enableAdaptiveUI](#) property, the grid will render the filter, sort, and edit dialogs in full screen for a better user experience. This behavior is demonstrated in the below demo.

### APP.VUE

```

<template>
    <div id="app">
        <div class="e-adaptive-demo e-bigger">
            <div class="e-mobile-layout">
                <div class="e-mobile-content">
                    <ejs-grid ref='grid' id='adaptivebrowser' :dataSource="data"
height='100%' :enableAdaptiveUI='true' :allowPaging='true'
:allowSorting='true' :allowFiltering='true'
:editSettings='editSettings' :toolbar='toolbar'
:filterSettings='filterSettings' :load='load'>
                        <e-columns>

```

```

        <e-column field='SNO' headerText='S NO' width='150'
:isPrimaryKey='true' :validationRules='orderidrules'></e-column>
        <e-column field='Model' headerText='Model Name'
width='200' editType='dropdownedit' :validationRules='customeridrules'></e-
column>
        <e-column field='Developer' headerText='Developer'
width='200' :validationRules='customeridrules' :filter='menuFilter'></e-
column>
        <e-column field='ReleaseDate' headerText='Released Date'
width='200' type='date' format='yMMM' editType='datepickeredit'></e-column>
        <e-column field='AndroidVersion' headerText='Android
Version' width='200' :validationRules='customeridrules'
:filter='checkboxFilter'></e-column>
    </e-columns>
</ejs-grid>
</div>
</div>
<br></br>
<div class="datalink">Source:
    <a
href="https://en.wikipedia.org/wiki/List_of_Android_smartphones"
target="_blank">Wikipedia: List of Android smartphones</a>
    </div>
</div>
</template>
<script>
import Vue from "vue";
import { GridPlugin, Filter, Sort, Edit, Toolbar, Page } from
"@syncfusion/ej2-vue-grids";
import { data } from './datasource.js'
Vue.use(GridPlugin);
export default {
    data() {
        return {
            data: data,
            orderidrules: { required: true, number: true },
            customeridrules: { required: true },
            editSettings: { allowAdding: true, allowEditing: true, allowDeleting:
true, mode: 'Dialog' },
            toolbar: ['Add', 'Edit', 'Delete', 'Update', 'Cancel', 'Search'],
            filterSettings: { type: 'Excel' },
            menuFilter: {
                type : 'Menu'
            },
            checkboxFilter: {
                type : 'CheckBox'
            }
        };
    },
    methods: {
        load: function() {
            (this.$refs.grid as any).$el.ej2_instances[0].adaptiveDlgTarget =
document.getElementsByClassName('e-mobile-content')[0];
        }
    },
    provide: {

```

```

    grid: [Filter, Sort, Edit, Toolbar, Page]
  }
}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
.e-grid .custom {
  background-color: #f48fb1 !important; /* csslint allow: important */
  color: white;
}
.e-grid .custom {
  background-color: #fce4ec;
  color: white;
}
/* The device with borders */
.e-mobile-layout {
  position: relative;
  width: 360px;
  height: 640px;
  margin: auto;
  border: 16px #f4f4f4 solid;
  border-top-width: 60px;
  border-bottom-width: 60px;
  border-radius: 36px;
  box-shadow: 0 0px 2px rgb(144 144 144), 0 0px 10px rgb(0 0 0 / 16%);
}
/* The horizontal line on the top of the device */
.e-mobile-layout:before {
  content: '';
  display: block;
  width: 60px;
  height: 5px;
  position: absolute;
  top: -30px;
  left: 50%;
  transform: translate(-50%, -50%);
  background: #ebebeb;
  border-radius: 10px;
}
/* The circle on the bottom of the device */
.e-mobile-layout:after {
  content: '';
  display: block;
  width: 35px;
  height: 35px;
  position: absolute;
  left: 50%;
  bottom: -65px;
  transform: translate(-50%, -50%);
  background: #e8e8e8;
  border-radius: 50%;
}
/* The screen (or content) of the device */
.e-mobile-layout .e-mobile-content {
  overflow-x: hidden;
  height: 100%;
  background: white;

```

```

    border: 0px solid #dddddd;
  }
  .highcontrast .e-mobile-layout {
    border: 16px #000000 solid;
    border-top-width: 60px;
    border-bottom-width: 60px;
    box-shadow: -1px 2px white, -2px -2px white, 2px -2px white, 2px 1px
white;
  }
  .e-responsive-dialog {
    box-shadow: none;
    border: 1px solid #dddddd;
  }
  /* Render the mobile pager by default */
  @media (max-width: 3840px) {
    .e-adaptive-demo .e-pager {
      padding: 13px 0;
    }
    .e-adaptive-demo .e-pager div.e-parentmsgbar {
      box-sizing: border-box;
      display: inline-block;
      float: initial;
      padding-bottom: 0;
      padding-right: 0;
      padding-top: 0;
      text-align: center;
      vertical-align: top;
      width: calc(60% - 48px);
    }
    .e-adaptive-demo .e-pager .e-pagesizes {
      display: none;
    }
    .e-adaptive-demo .e-pager .e-pagecountmsg {
      display: none;
    }
    .e-adaptive-demo .e-pager .e-pagercontainer {
      display: none;
    }
    .e-adaptive-demo .e-pager .e-icons {
      font-size: 11px;
    }
    .e-adaptive-demo .e-pager .e-mfirst,
    .e-adaptive-demo .e-pager .e-mprev,
    .e-adaptive-demo .e-pager .e-mnext,
    .e-adaptive-demo .e-pager .e-mlast {
      border: 0;
      box-sizing: border-box;
      display: inline-block;
      padding: 1% 5%;
    }
    .e-adaptive-demo .e-pager .e-mfirst {
      margin-right: 4px;
      text-align: right;
      width: calc(10% + 11px);
    }
    .e-adaptive-demo .e-pager .e-mprev {
      margin: 0 4px;

```

```

        text-align: right;
        width: 10%;
    }
    .e-adaptive-demo .e-pager .e-mnext {
        margin: 0 4px;
        text-align: left;
        width: 10%;
    }
    .e-adaptive-demo .e-pager .e-mlast {
        margin-left: 4px;
        text-align: left;
        width: calc(10% + 11px);
    }
    .e-adaptive-demo .e-bigger .e-pager,
    .e-adaptive-demo .e-pager.e-bigger {
        padding: 19px 0;
    }
    .e-adaptive-demo .e-bigger .e-pager.e-rtl div.e-parentmsgbar,
    .e-adaptive-demo .e-pager.e-bigger.e-rtl div.e-parentmsgbar {
        margin-right: 0;
    }
    .e-adaptive-demo .e-bigger .e-pager div.e-parentmsgbar,
    .e-adaptive-demo .e-pager.e-bigger div.e-parentmsgbar {
        padding: 0;
    }
    }
    }

    .e-dlg-target.e-scroll-disabled {
        overflow: auto !important;
    }
}
</style>

```

{% previewsample "page.domainurl/code-snippet/grid/adaptive-cs1" %}

### Vertical row rendering

The grid will render the row elements in vertical order while setting the [rowRenderingMode](#) property value as **Vertical**.

### APP.VUE

```

<template>
    <div id="app">
        <div class="e-adaptive-demo e-bigger">
            <div class="e-mobile-layout">
                <div class="e-mobile-content">
                    <ejs-grid ref='grid' id='adaptivebrowser' :dataSource="data"
height='100%' :enableAdaptiveUI='true' :allowPaging='true'
:allowSorting='true' :allowFiltering='true' :editSettings='editSettings'
:toolbar='toolbar' :filterSettings='filterSettings' :load='load'>
                        <e-columns>
                            <e-column field='SNO' headerText='S NO' width='150'
:isPrimaryKey='true' :validationRules='orderidrules'></e-column>
                            <e-column field='Model' headerText='Model Name'
width='200' editType='dropdownedit' :validationRules='customeridrules'></e-
column>

```



```

        <e-column field='Developer' headerText='Developer'
width='200' :validationRules='customeridrules' :filter='menuFilter'></e-
column>
        <e-column field='ReleaseDate' headerText='Released Date'
width='200' type='date' format='yMMM' editType='datepickeredit'></e-column>
        <e-column field='AndroidVersion' headerText='Android
Version' width='200' :validationRules='customeridrules'
:filter='checkboxFilter'></e-column>
    </e-columns>
    <e-aggregates>
        <e-aggregate>
            <e-columns>
                <e-column type="Count" field="Model"
:footerTemplate="sumTemplate" >
            </e-column>
        </e-columns>
    </e-aggregate>
</e-aggregates>
</ejs-grid>
</div>
</div>
<br></br>
<div className="datalink">Source:
    <a
href="https://en.wikipedia.org/wiki/List_of_Android_smartphones"
target="_blank">Wikipedia: List of Android smartphones</a>
    </div>
</div>
</template>
<script>
import Vue from "vue";
import { GridPlugin, Filter, Sort, Edit, Toolbar, Page, Aggregate } from
"@syncfusion/ej2-vue-grids";
import { data } from './datasource.js'
Vue.use(GridPlugin);
export default {
    data() {
        return {
            data: data,
            orderidrules: { required: true, number: true },
            customeridrules: { required: true },
            editSettings: { allowAdding: true, allowEditing: true, allowDeleting:
true, mode: 'Dialog' },
            toolbar: ['Add', 'Edit', 'Delete', 'Update', 'Cancel', 'Search'],
            filterSettings: { type: 'Excel' },
            rowMode: 'Vertical',
            menuFilter: {
                type : 'Menu'
            },
            checkboxFilter: {
                type : 'CheckBox'
            },
            sumTemplate: function() {
                return {
                    template: Vue.component('sumTemplate', {
                        template: `<span>Total Models: {{data.Count}}</span>`,

```

```

        data: function () {return {data: {data: {}}};}
    })
  }
}
};
},
methods: {
  load: function() {
    (this.$refs.grid as any).$el.ej2_instances[0].adaptiveDlgTarget =
document.getElementsByClassName('e-mobile-content')[0];
  }
},
provide: {
  grid: [Filter, Sort, Edit, Toolbar, Page, Aggregate]
}
}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
.e-grid .custom {
  background-color: #f48fb1 !important; /* csslint allow: important */
  color: white;
}
.e-grid .custom {
  background-color: #fce4ec;
  color: white;
}
/* The device with borders */
.e-mobile-layout {
  position: relative;
  width: 360px;
  height: 640px;
  margin: auto;
  border: 16px #f4f4f4 solid;
  border-top-width: 60px;
  border-bottom-width: 60px;
  border-radius: 36px;
  box-shadow: 0 0px 2px rgb(144 144 144), 0 0px 10px rgb(0 0 0 / 16%);
}
/* The horizontal line on the top of the device */
.e-mobile-layout:before {
  content: '';
  display: block;
  width: 60px;
  height: 5px;
  position: absolute;
  top: -30px;
  left: 50%;
  transform: translate(-50%, -50%);
  background: #ebebeb;
  border-radius: 10px;
}
/* The circle on the bottom of the device */
.e-mobile-layout:after {
  content: '';
  display: block;
  width: 35px;

```

```

    height: 35px;
    position: absolute;
    left: 50%;
    bottom: -65px;
    transform: translate(-50%, -50%);
    background: #e8e8e8;
    border-radius: 50%;
  }
  /* The screen (or content) of the device */
  .e-mobile-layout .e-mobile-content {
    overflow-x: hidden;
    height: 100%;
    background: white;
    border: 0px solid #dddddd;
  }
  .highcontrast .e-mobile-layout {
    border: 16px #000000 solid;
    border-top-width: 60px;
    border-bottom-width: 60px;
    box-shadow: -1px 2px white, -2px -2px white, 2px -2px white, 2px 1px
white;
  }
  .e-responsive-dialog {
    box-shadow: none;
    border: 1px solid #dddddd;
  }
  /* Render the mobile pager by default */
  @media (max-width: 3840px) {
    .e-adaptive-demo .e-pager {
      padding: 13px 0;
    }
    .e-adaptive-demo .e-pager div.e-parentmsgbar {
      box-sizing: border-box;
      display: inline-block;
      float: initial;
      padding-bottom: 0;
      padding-right: 0;
      padding-top: 0;
      text-align: center;
      vertical-align: top;
      width: calc(60% - 48px);
    }
    .e-adaptive-demo .e-pager .e-pagesizes {
      display: none;
    }
    .e-adaptive-demo .e-pager .e-pagecountmsg {
      display: none;
    }
    .e-adaptive-demo .e-pager .e-pagercontainer {
      display: none;
    }
    .e-adaptive-demo .e-pager .e-icons {
      font-size: 11px;
    }
    .e-adaptive-demo .e-pager .e-mfirst,
    .e-adaptive-demo .e-pager .e-mprev,
    .e-adaptive-demo .e-pager .e-mnext,

```

```

.e-adaptive-demo .e-pager .e-mlast {
  border: 0;
  box-sizing: border-box;
  display: inline-block;
  padding: 1% 5%;
}
.e-adaptive-demo .e-pager .e-mfirst {
  margin-right: 4px;
  text-align: right;
  width: calc(10% + 11px);
}
.e-adaptive-demo .e-pager .e-mprev {
  margin: 0 4px;
  text-align: right;
  width: 10%;
}
.e-adaptive-demo .e-pager .e-mnext {
  margin: 0 4px;
  text-align: left;
  width: 10%;
}
.e-adaptive-demo .e-pager .e-mlast {
  margin-left: 4px;
  text-align: left;
  width: calc(10% + 11px);
}
.e-adaptive-demo .e-bigger .e-pager,
.e-adaptive-demo .e-pager.e-bigger {
  padding: 19px 0;
}
.e-adaptive-demo .e-bigger .e-pager.e-rtl div.e-parentmsgbar,
.e-adaptive-demo .e-pager.e-bigger.e-rtl div.e-parentmsgbar {
  margin-right: 0;
}
.e-adaptive-demo .e-bigger .e-pager div.e-parentmsgbar,
.e-adaptive-demo .e-pager.e-bigger div.e-parentmsgbar {
  padding: 0;
}
}

.e-dlg-target.e-scroll-disabled {
  overflow: auto !important;
}
</style>

```

{% previewsample "page.domainurl/code-snippet/grid/adaptive-cs2" %}

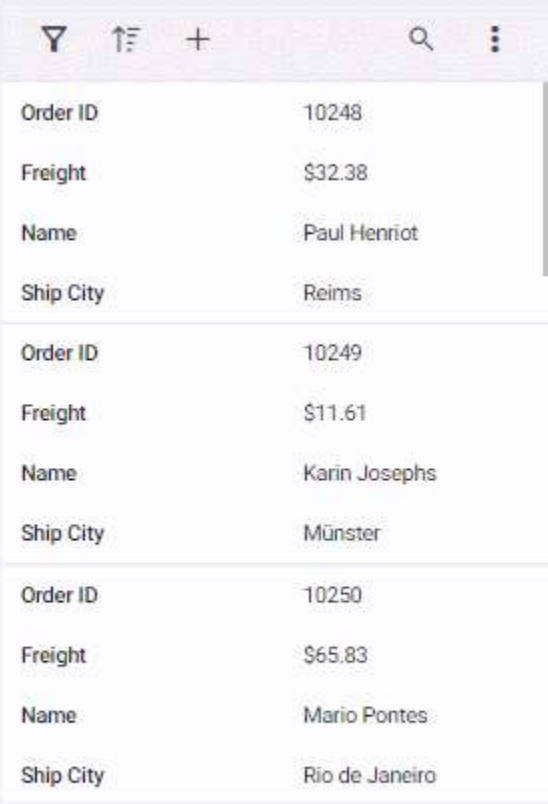
[enableAdaptiveUI](#) property must be enabled for vertical row rendering.

*Supported features by vertical row rendering*

The following features are only supported in vertical row rendering:

- Paging, including Page size dropdown
- Sorting
- Filtering

- Selection
- Dialog Editing
- Aggregate
- Infinite scroll
- Toolbar - Options like **Add**, **Filter**, **Sort**, **Edit**, **Delete**, **Search**, and **Toolbar template** are available when their respective features are enabled. The toolbar dynamically includes a three-dotted icon, containing additional features like **ColumnChooser**, **Print**, **PdfExport**, **ExcelExport**, or **CsvExport**, once these features are enabled. Please refer to the following snapshot.



Order ID	10248
Freight	\$32.38
Name	Paul Henriot
Ship City	Reims
Order ID	10249
Freight	\$11.61
Name	Karin Josepchs
Ship City	Münster
Order ID	10250
Freight	\$65.83
Name	Mario Pontes
Ship City	Rio de Janeiro

A snapshot of the adaptive grid displaying enabled paging along with a pager dropdown.

<div> <div></div> <div></div> <div></div> </div>	
Order ID	10248
Freight	\$32.38
Name	Vins et alcools Chevalier
Ship City	Reims
Order ID	10249
Freight	\$11.61
Name	Toms Spezialitäten
Ship City	Münster
Order ID	10250
Freight	\$65.83
<div> <div> <div></div> <div></div> <div></div> </div> <div>1 / 6</div> <div> <div></div> <div></div> </div> <div>Items per page</div> <div>5</div> </div>	

The Column Menu feature, which includes grouping, sorting, autofit, filter, and column chooser, is exclusively supported for the Grid in **Horizontal** [rowRenderingMode](#).

### Hierarchy grid in Vue Grid component

The Grid allows display of table data in a hierarchical structure to visualize relations between parent and child records. This feature is enabled by defining the [childGrid](#) and [childGrid.queryString](#).

The [childGrid](#) describes the options of grid and the [childGrid.queryString](#) describes the relation between parent and child grids.

To use hierarchical binding, inject the **DetailRow** in the **provide** section.

#### APP.VUE

```
<template>
  <div id="app">
    <ejs-grid :dataSource='parentData' height='315px'
    :childGrid='childGrid'>
      <e-columns>
        <e-column field='EmployeeID' headerText='Employee ID'
        textAlign='Right' width=120></e-column>
        <e-column field='FirstName' headerText='FirstName'
        width=150></e-column>
        <e-column field='LastName' headerText='Last Name'
        width=150></e-column>
      </e-columns>
    </ejs-grid>
  </div>
</template>
```

```

        <e-column field='City' headerText='City' width=150></e-
column>
    </e-columns>
</ejs-grid>
</div>
</template>
<script>
import Vue from "vue";
import { GridPlugin, DetailRow } from "@syncfusion/ej2-vue-grids";
import { data, employeeData } from './datasource.js';
Vue.use(GridPlugin);
export default {
  data() {
    return {
      parentData: employeeData,
      childGrid: {
        dataSource: data,
        queryString: 'EmployeeID',
        columns: [
          { field: 'OrderID', headerText: 'Order ID', textAlign: 'Right',
width: 120 },
          { field: 'CustomerID', headerText: 'Customer ID', width: 150 },
          { field: 'ShipCity', headerText: 'Ship City', width: 150 },
          { field: 'ShipName', headerText: 'Ship Name', width: 150 }
        ]
      }
    },
    provide: {
      grid: [DetailRow]
    }
  }
</script>
<style>
@import
"https://ej2.syncfusion.com/vue/documentation/node_modules/@syncfusion/ej2-
vue-grids/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/grid/hierarchy-grid/default-cs2" %}

\* Grid supports n level of child grids.

\* Hierarchical binding is not supported when [DetailTemplate](#) is enabled.

### ExpandAll by external button

By default, grid renders in collapsed state. You can expand all child grid rows by invoking the [expandAll](#) method, and collapse all grid rows by invoking the [collapseAll](#) through an external button.

### APP.VUE

```

<template>
  <div id="app">
    <ejs-button @click.native='expand'>Expand All</ejs-button>
    <ejs-button @click.native='collapse'>Collapse All</ejs-button>
    <ejs-grid ref='grid' :dataSource='parentData' height='265px'
:childGrid='childGrid'>

```

```

        <e-columns>
            <e-column field='EmployeeID' headerText='Employee ID'
textAlign='Right' width=120></e-column>
            <e-column field='FirstName' headerText='FirstName'
width=150></e-column>
            <e-column field='LastName' headerText='Last Name'
width=150></e-column>
            <e-column field='City' headerText='City' width=150></e-
column>
        </e-columns>
    </ejs-grid>
</div>
</template>
<script>
import Vue from "vue";
import { GridPlugin, DetailRow } from "@syncfusion/ej2-vue-grids";
import { ButtonPlugin } from "@syncfusion/ej2-vue-buttons";
import { data, employeeData } from './datasource.js';
Vue.use(GridPlugin);
Vue.use(ButtonPlugin);
export default {
    data() {
        return {
            parentData: employeeData,
            childGrid: {
                dataSource: data,
                queryString: 'EmployeeID',
                columns: [
                    { field: 'OrderID', headerText: 'Order ID', textAlign: 'Right',
width: 120 },
                    { field: 'CustomerID', headerText: 'Customer ID', width: 150 },
                    { field: 'ShipCity', headerText: 'Ship City', width: 150 },
                    { field: 'ShipName', headerText: 'Ship Name', width: 150 }
                ]
            }
        },
        methods: {
            expand: function() {
                this.$refs.grid.ej2Instances.detailRowModule.expandAll();
            },
            collapse: function() {
                this.$refs.grid.ej2Instances.detailRowModule.collapseAll();
            }
        },
        provide: {
            grid: [DetailRow]
        }
    }
}
</script>
<style>
    @import
    "https://ej2.syncfusion.com/vue/documentation/node_modules/@syncfusion/ej2-
vue-grids/styles/material.css";
</style>

```



{% previewsample "page.domainurl/code-snippet/grid/hierarchy-grid/default-cs3" %}

### Expand child grid initially

You can expand a particular child grid at initial rendering by invoking the [expand](#) method in the [dataBound](#) event.

#### APP.VUE

```
<template>
  <div id="app">
    <ejs-grid ref='grid' :dataSource='parentData' height='265px'
    :childGrid='childGrid' :dataBound='onDataBound'>
      <e-columns>
        <e-column field='EmployeeID' headerText='Employee ID'
        textAlign='Right' width=120></e-column>
        <e-column field='FirstName' headerText='First Name'
        width=150></e-column>
        <e-column field='LastName' headerText='Last Name'
        width=150></e-column>
        <e-column field='City' headerText='City' width=150></e-
        column>
      </e-columns>
    </ejs-grid>
  </div>
</template>
<script>
import Vue from "vue";
import { GridPlugin, DetailRow } from "@syncfusion/ej2-vue-grids";
import { data, employeeData } from './datasource.js';
Vue.use(GridPlugin);
export default {
  data() {
    return {
      parentData: employeeData,
      childGrid: {
        dataSource: data,
        queryString: 'EmployeeID',
        columns: [
          { field: 'OrderID', headerText: 'Order ID', textAlign: 'Right',
width: 120 },
          { field: 'CustomerID', headerText: 'Customer ID', width: 150 },
          { field: 'ShipCity', headerText: 'Ship City', width: 150 },
          { field: 'ShipName', headerText: 'Ship Name', width: 150 }
        ]
      }
    }
  },
  methods: {
    onDataBound: function() {
      this.$refs.grid.ej2Instances.detailRowModule.expand(2); // initially
2nd child Grid will expand.
    }
  },
  provide: {
    grid: [DetailRow]
  }
}
```

```

</script>
<style>
  @import
  "https://ej2.syncfusion.com/vue/documentation/node_modules/@syncfusion/ej2-
  vue-grids/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/grid/hierarchy-grid/default-cs4" %}

### Dynamically load child grid data

You can dynamically load child grid dataSource by using the [detailDataBound](#) event. This [detailDataBound](#) event triggers when the child grid is expanded for the first time.

### APP.VUE

```

<template>
  <div id="app">
    <ejs-grid ref='grid' :dataSource='parentData' height='265px'
    :childGrid='childGrid' :detailDataBound='onDetailDataBound'>
      <e-columns>
        <e-column field='EmployeeID' headerText='Employee ID'
        textAlign='Right' width=120></e-column>
        <e-column field='FirstName' headerText='FirstName'
        width=150></e-column>
        <e-column field='LastName' headerText='Last Name'
        width=150></e-column>
        <e-column field='City' headerText='City' width=150></e-
        column>
      </e-columns>
    </ejs-grid>
  </div>
</template>
<script>
import Vue from "vue";
import { GridPlugin, DetailRow } from "@syncfusion/ej2-vue-grids";
import { data, employeeData } from './datasource.js';
Vue.use(GridPlugin);
export default {
  data() {
    return {
      parentData: employeeData,
      childGrid: {
        queryString: 'EmployeeID',
        columns: [
          { field: 'OrderID', headerText: 'Order ID', textAlign: 'Right',
width: 120 },
          { field: 'CustomerID', headerText: 'Customer ID', width: 150 },
          { field: 'ShipCity', headerText: 'Ship City', width: 150 },
          { field: 'ShipName', headerText: 'Ship Name', width: 150 }
        ]
      }
    }
  },
  methods: {
    onDetailDataBound(args) {

```

```

        args.detailElement.querySelector('.e-grid').ej2_instances[0].dataSource = data; // assign data source for child grid.
    },
    provide: {
        grid: [DetailRow]
    }
}
</script>
<style>
    @import
    "https://ej2.syncfusion.com/vue/documentation/node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/grid/hierarchy-grid/default-cs5" %}

### Bind hierarchy grid with different field

By default, Parent and child grid relation will be maintained by [queryString](#) property. We should use the same field name to map both parent and child grid. To achieve parent and child relation with different fields, we need to change the mapping value in the child grid [load](#) event.

In the below sample, we have bound the child and parent grid with different fields. Parent grid field name as **EmployeeID** and the child grid field name as **ID**. We need to define the mapping value of **parentKeyFieldValue** from the parent row data in the child grid [load](#) event.

### APP.VUE

```

<template>
    <div id="app">
        <ejs-grid ref='grid' :dataSource='parentData' height='265px' :childGrid='childGrid'>
            <e-columns>
                <e-column field='EmployeeID' headerText='Employee ID'
                textAlign='Right' width=120></e-column>
                <e-column field='FirstName' headerText='First Name'
                width=150></e-column>
                <e-column field='LastName' headerText='Last Name'
                width=150></e-column>
                <e-column field='City' headerText='City' width=150></e-column>
            </e-columns>
        </ejs-grid>
    </div>
</template>
<script>
import Vue from "vue";
import { GridPlugin, DetailRow } from "@syncfusion/ej2-vue-grids";
import { data, employeeData, childdata } from './datasource.js';
Vue.use(GridPlugin);
export default {
    data() {
        return {
            parentData: employeeData,
            childGrid: {

```

```

        dataSource: childdata,
        queryString: 'ID',
        columns: [
            { field: 'ID', headerText: 'Order ID', textAlign: 'Right',
width: 120 },
            { field: 'CustomerID', headerText: 'Customer ID', width: 150 },
            { field: 'ShipCity', headerText: 'Ship City', width: 150 },
            { field: 'ShipName', headerText: 'Ship Name', width: 150 }
        ],
        load: function () {
            this.parentDetails.parentKeyFieldValue =
this.parentDetails.parentRowData['EmployeeID'];
        }
    },
    provide: {
        grid: [DetailRow]
    }
}
</script>
<style>
    @import
    "https://ej2.syncfusion.com/vue/documentation/node_modules/@syncfusion/ej2-
vue-grids/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/grid/hierarchy-grid/default-cs6" %}

### Adding Record in ChildGrid

Parent and child grid are related by [queryString](#) field value. To maintain this relation in newly added record, You need to set value for [\[queryString\]](#)(

<https://ej2.syncfusion.com/vue/documentation/api/grid/#querystring>) field in the added data by the [actionBegin](#) event.

In the below demo, **EmployeeID** field relates the parent and child grids. To add a new record in child grid, We have to set the **EmployeeID** field with parent record's [queryString](#) field value in the [actionBegin](#) event.

### APP.VUE

```

<template>
    <div id="app">
        <ejs-grid ref='grid' :dataSource='parentData' height='265px'
:childGrid='childGrid'>
            <e-columns>
                <e-column field='EmployeeID' headerText='Employee ID'
textAlign='Right' width=120></e-column>
                <e-column field='FirstName' headerText='FirstName'
width=150></e-column>
                <e-column field='LastName' headerText='Last Name'
width=150></e-column>
                <e-column field='City' headerText='City' width=150></e-
column>
            </e-columns>
        </ejs-grid>
    </div>
</template>

```

```

    </div>
</template>
<script>
import Vue from "vue";
import { GridPlugin, DetailRow, Edit, Toolbar } from "@syncfusion/ej2-vue-grids";
import { data, employeeData } from './datasource.js';
Vue.use(GridPlugin);
export default {
  data() {
    return {
      parentData: employeeData,
      childGrid: {
        dataSource: data,
        queryString: 'EmployeeID',
        toolbar: ['Add', 'Edit', 'Delete', 'Update', 'Cancel'],
        editSettings: { allowEditing: true, allowAdding: true,
allowDeleting: true },
        columns: [
          { field: 'OrderID', headerText: 'Order ID', isPrimaryKey:true,
textAlign: 'Right', width: 120 },
          { field: 'EmployeeID', headerText: 'Employee ID', textAlign:
'Right', allowEditing:false, width: 120 },
          { field: 'ShipCity', headerText: 'Ship City', width: 150 },
          { field: 'ShipName', headerText: 'Ship Name', width: 150 }
        ],
        actionBegin: function(args) {
          if (args.requestType === "add") {
            // `parentKeyFieldValue` refers to the queryString field value of
the parent record.
            args.data.EmployeeID = this.parentDetails.parentKeyFieldValue;
          }
        }
      }
    },
    provide: {
      grid: [DetailRow, Edit, Toolbar]
    }
  }
}
</script>
<style>
@import
"https://ej2.syncfusion.com/vue/documentation/node_modules/@syncfusion/ej2-
vue-grids/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/grid/hierarchy-grid/default-cs7" %}

### Dynamically bind data to child grid based on parent row Data

By default, the [childGrid.queryString](#) describes the relationship between parent and child grids and visualizes the data in a hierarchical structure. Instead of the `queryString` property, we can dynamically bind the datasource to the `childGrid` based on the parent row data using the [detailDataBound](#) event of the grid.

While expanding the child Grid, the `detailDataBound` event will be triggered. In this event, based on the EmployeeID column value of parent row data, filter the equally matched data from the `childGrid` datasource using the `DataManager` plugin and bind the filtered data as a datasource to the `childGrid`. This can be demonstrated by the following sample.

#### APP.VUE

```
<template>
  <div id="app">
    <ejs-grid ref='grid' :dataSource='parentData' height='265px'
      :childGrid='childGrid' :detailDataBound='onDetailDataBound'>
      <e-columns>
        <e-column field='EmployeeID' headerText='Employee ID'
          textAlign='Right' width=120></e-column>
        <e-column field='FirstName' headerText='FirstName'
          width=150></e-column>
        <e-column field='LastName' headerText='Last Name'
          width=150></e-column>
        <e-column field='City' headerText='City' width=150></e-
column>
      </e-columns>
    </ejs-grid>
  </div>
</template>
<script>
import Vue from "vue";
import { GridPlugin, DetailRow } from "@syncfusion/ej2-vue-grids";
import { data, employeeData } from './datasource.js';
import { DataManager, Query } from '@syncfusion/ej2-data';
Vue.use(GridPlugin);
export default {
  data() {
    return {
      parentData: employeeData,
      childGrid: {
        columns: [
          { field: 'OrderID', headerText: 'Order ID', textAlign: 'Right',
width: 120 },
          { field: 'CustomerID', headerText: 'Customer ID', width: 150 },
          { field: 'ShipCity', headerText: 'Ship City', width: 150 },
          { field: 'ShipName', headerText: 'Ship Name', width: 150 }
        ]
      }
    }
  },
  methods: {
    onDetailDataBound(args) {
      var orderData = data;
      var empIdValue =
args.childGrid.parentDetails.parentRowData.EmployeeID;
      var matchedData = new DataManager(data).executeLocal(
        new Query().where("EmployeeID", "equal", empIdValue, true)
      );
      args.childGrid.query = new Query();
      args.childGrid.dataSource = matchedData;
    }
  }
}
```

```

    },
    provide: {
      grid: [DetailRow]
    }
  }
</script>
<style>
  @import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/grid/hierarchy-grid/default-cs8" %}

## State Persistence

### State persistence in Vue Grid component

State persistence refers to the Grid's state maintained in the browser's [localStorage](#) even if the browser is refreshed or if you move to the next page within the browser. State persistence stores grid's model object in the local storage when the [enablePersistence](#) is defined as true.

#### Restore initial Grid state

When the [enablePersistence](#) property is set to **true**, the Grid will keep its state even if the page is reloaded. In some cases, you may be required to retain the Grid in its initial state. The Grid will not retain its initial state now since the [enablePersistence](#) property has been enabled.

You can achieve this by destroying the grid after disabling the [enablePersistence](#) property and clearing the local storage data, as shown in the sample below.

### APP.VUE

```

<template>
  <div id="app">
    <button id="restore" @click="clickRestore">Restore to initial
    state</button>
    <br /><br />
    <ejs-grid ref="grid" :dataSource='data' :enablePersistence='true'
    :allowPaging='true' :allowFiltering='true' height='230px' id="Grid">
      <e-columns>
        <e-column field='OrderID' headerText='Order ID'
        textAlign='Right' width=120></e-column>
        <e-column field='CustomerID' headerText='Customer ID'
        width=150></e-column>
        <e-column field='ShipCity' headerText='Ship City'
        width=150></e-column>
        <e-column field='ShipName' headerText='Ship Name'
        width=150></e-column>
      </e-columns>
    </ejs-grid>
  </div>
</template>
<script>
import Vue from "vue";
import { GridPlugin, Page, Filter } from "@syncfusion/ej2-vue-grids";
import { data } from './datasource.js';
Vue.use(GridPlugin);
export default {
  data() {

```

```

    return {
      data: data
    };
  },
  methods: {
    clickRestore: function () {
      this.$refs.grid.ej2Instances.enablePersistence = false;
      window.localStorage.setItem("gridGrid", "");
      this.$refs.grid.ej2Instances.destroy();
      location.reload();
    }
  },
  provide: {
    grid: [Page, Filter]
  }
}
</script>
<style>
  @import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/grid/sort/default-cs8" %}

#### *Maintaining custom query in state persistence*

Grid does not maintain the query params after page load event when [enablePersistence](#) is set to true. This is because the Grid refreshes its query params for every page load. You can maintain the custom query params by resetting the `addParams` method in the [actionBegin](#) event.

#### **APP.VUE**

```

<template>
  <div id="app">
    <ejs-grid ref="grid" :dataSource='data' :allowSorting='true'
      :enablePersistence='true' :allowPaging='true' :allowFiltering='true'
      height='210px' :actionBegin='actionHandler'>
      <e-columns>
        <e-column field='OrderID' headerText='Order ID'
          textAlign='Right' width=120></e-column>
        <e-column field='CustomerID' headerText='Customer ID'
          width=150></e-column>
        <e-column field='ShipCity' headerText='Ship City'
          width=150></e-column>
        <e-column field='ShipName' headerText='Ship Name'
          width=150></e-column>
      </e-columns>
    </ejs-grid>
  </div>
</template>
<script>
import Vue from "vue";
import { GridPlugin, Sort, Page, Filter } from "@syncfusion/ej2-vue-grids";
import { data } from './datasource.js';
Vue.use(GridPlugin);
export default {
  data() {
    return {

```



```

        data: data
      };
    },
    methods: {
      actionHandler: function () {
        this.$refs.grid.ej2Instances.query.addParams('$filter', 'EmployeeID eq 1');
      }
    }
  },
  provide: {
    grid: [Sort, Page, Filter]
  }
}
</script>
<style>
  @import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/grid/sort/default-cs9" %}

#### Add a new column in persisted columns list

The Grid columns can be persisted when the [enablePersistence](#) property is set to true. If you want to add the new columns with the existing persist state, you can use the Grid inbuilt method such as `column.push` and call the [refreshColumns\(\)](#) method for UI changes. Please refer to the following sample for more information.

#### APP.VUE

```

<template>
  <div id="app">
    <button id="add" @click="clickAdd">Add Columns</button>
    <br /><br />
    <ejs-grid ref="grid" :dataSource='data' :enablePersistence='true'
    :allowPaging='true' height='230px' id="Grid">
      <e-columns>
        <e-column field='OrderID' headerText='Order ID'
        textAlign='Right' width=120></e-column>
        <e-column field='CustomerID' headerText='Customer ID'
        width=150></e-column>
        <e-column field='ShipCity' headerText='Ship City'
        width=150></e-column>
        <e-column field='ShipName' headerText='Ship Name'
        width=150></e-column>
      </e-columns>
    </ejs-grid>
  </div>
</template>
<script>
import Vue from "vue";
import { GridPlugin, Page } from "@syncfusion/ej2-vue-grids";
import { data } from './datasource.js';
Vue.use(GridPlugin);
export default {
  data() {
    return {
      data: data
    }
  }
}

```

```

    };
  },
  methods: {
    clickAdd: function () {
      let obj = { field: "Freight", headerText: 'Freight', width: 120 }
      this.$refs.grid.ej2Instances.columns.push(obj as any); //you can add
the columns by using the Grid columns method
      this.$refs.grid.ej2Instances.refreshColumns();
    }
  },
  provide: {
    grid: [Page]
  }
}
</script>
<style>
  @import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/grid/sort/default-cs10" %}

[See Also](#)

- [How to override Grid persistence in Vue Grid](#)

Get or set local storage value in Vue Grid component

If the [enablePersistence](#) property set as true, The Grid property value is saved in the `window.localStorage` for reference. You can get/set the `localStorage` value by using the `getItem/setItem` method in `window.localStorage`.

`ts

//get the Grid model.

let value = window.localStorage.getItem('gridGrid'); //"gridGrid" is component name + component id.

let model = JSON.parse(model);

,

`ts

//set the Grid model.

window.localStorage.setItem('gridGrid', JSON.stringify(model)); //"gridGrid" is component name + component id.

,

Prevent to persist in Vue Grid component

[Prevent columns from persisting](#)

When the [enablePersistence](#) property is set to true, the Grid properties such as [Grouping](#), [Paging](#), [Filtering](#), [Sorting](#), and [Columns](#) will persist. You can use the `addOnPersist` method to prevent these Grid properties from persisting.

The following example demonstrates how to prevent Grid columns from persisting. In the [dataBound](#) event of the Grid, you can override the `addOnPersist` method and remove the columns from the key list given for persistence.

**Note:** When the [enablePersistence](#) property is set to true, the Grid properties such as column template, column formatter, header text, and value accessor will not persist.

#### APP.VUE

```
<template>
  <div id="app">
    <button id="add" @click="clickAdd">Add Columns</button>
    <button id="remove" @click="clickRemove">Remove Columns</button>
    <br /><br />
    <ejs-grid ref="grid" :dataSource='data' :enablePersistence='true'
:allowPaging='true' height='230px' id="Grid" :dataBound='dataBound'>
      <e-columns>
        <e-column field='OrderID' headerText='Order ID'
textAlign='Right' width=120></e-column>
        <e-column field='CustomerID' headerText='Customer ID'
width=150></e-column>
        <e-column field='ShipCity' headerText='Ship City'
width=150></e-column>
        <e-column field='ShipName' headerText='Ship Name'
width=150></e-column>
      </e-columns>
    </ejs-grid>
  </div>
</template>
<script>
import Vue from "vue";
import { GridPlugin, Page } from "@syncfusion/ej2-vue-grids";
import { data } from './datasource.js';
Vue.use(GridPlugin);
export default {
  data() {
    return {
      data: data
    };
  },
  methods: {
    dataBound: function () {
      let cloned = this.$refs.grid.ej2Instances.addOnPersist;
      this.$refs.grid.ej2Instances.addOnPersist = function (key: any) {
        key = key.filter((item: string) => item !== "columns");
        return cloned.call(this, key);
      };
    },
    clickAdd: function () {
      let obj = { field: "Freight", headerText: 'Freight', width: 120 }
      this.$refs.grid.ej2Instances.columns.push(obj as any); //you can add
the columns by using the Grid columns method
      this.$refs.grid.ej2Instances.refreshColumns();
    },
    clickRemove: function () {
      this.$refs.grid.ej2Instances.columns.pop();
      this.$refs.grid.ej2Instances.refreshColumns();
    }
  }
}
```

```

    }
  },
  provide: {
    grid: [Page]
  }
}
</script>
<style>
  @import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/grid/sort/default-cs7" %}

### Add to persist in Vue Grid component

#### *Persist the column template, header template and header Text*

By default, the Grid properties such as column template, header text, header template, column formatter, and value accessor will not persist when [enablePersistence](#) is set to true. Because the column template and header text can be customized at the application level.

If you wish to restore all these column properties, then you can achieve it by cloning the grid's columns property using JavaScript Object's assign method and storing this along with the persist data manually. While restoring the settings, this column object must be assigned to the grid's columns property to restore the column settings as demonstrated in the following sample.

### **APP.VUE**

```

<template>
  <div id="app">
    <button id="restore" @click="clickRestore">Restore</button>
    <br /><br />
    <ejs-grid ref="grid" :dataSource='data' :enablePersistence='true'
:allowPaging='true' height='230px' id="Grid">
      <e-columns>
        <e-column field='OrderID' headerText='Order ID'
textAlign='Right' width=120></e-column>
        <e-column field='CustomerID' headerText='Customer ID'
width=150 :headerTemplate="hTemplate"></e-column>
        <e-column field='ShipName' headerText='Ship Name' width=150
:template='cTemplate"></e-column>
      </e-columns>
    </ejs-grid>
  </div>
</template>
<script id="template" type="text/x-template">
  <a rel='nofollow'
href=https://en.wikipedia.org/wiki/${ShipName}><span class="e-icons e-
column"></span></a>
</script>

  <script id="customertemplate" type="text/x-template">
    <span class="e-icons e-header" ></span>
    Customer ID
  </script>
<script>
import Vue from "vue";

```

```

import { GridPlugin, Page } from "@syncfusion/ej2-vue-grids";
import { data } from './datasource.js';
Vue.use(GridPlugin);
var headTemplate = Vue.component("header", {
  template: '<span class="e-icons e-header">CustomerID</span>',
  data() {
    return {
      data: {}
    };
  }
});
var columnTemplate = Vue.component("column", {
  template: '<a rel="nofollow" href="https://en.wikipedia.org/wiki/${ShipName}"><span class="e-icons e-column"></span></a>',
  data() {
    return {
      data: {}
    };
  }
});
export default {
  data() {
    return {
      data: data,
      hTemplate: function (e) {
        return {
          template: headTemplate
        };
      },
      cTemplate: function (e) {
        return {
          template: columnTemplate
        };
      }
    };
  },
  methods: {
    clickRestore: function () {
      let savedProperties =
JSON.parse(this.$refs.grid.ej2Instances.getPersistData());
      let gridColumnsState = Object.assign([],
this.$refs.grid.ej2Instances.getColumns());
      savedProperties.columns.forEach((col: {
        field: any;
        headerText: any;
        template: any;
        headerTemplate: any;
      }) => {
        let headerText = gridColumnsState.find((colColumnsState) =>
colColumnsState.field === col.field)['headerText'];
        let colTemplate = gridColumnsState.find((colColumnsState) =>
colColumnsState.field === col.field)['template'];
        let headerTemplate = gridColumnsState.find((colColumnsState) =>
colColumnsState.field === col.field)['headerTemplate'];
        col.headerText = 'Text Changed';
        col.template = colTemplate;
      });
    }
  }
};

```

```

        col.headerTemplate = headerTemplate; //likewise you can restore
        required column properties as per your wants.
    }
    );
    console.log(savedProperties);
    this.$refs.grid.ej2Instances.setProperties(savedProperties);
}
},
provide: {
    grid: [Page]
}
}
</script>
<style>
    .e-column:before {
        content: '\e815';
    }
    .e-header:before {
        content: '\ea9a';
    }
    @import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/grid/sort/default-cs6" %}

## Tool Bar

### Tool bar in Vue Grid component

The Grid provides Toolbar support to handle grid actions. The [toolbar](#) property accepts either the collection of built-in toolbar items and `ItemModel` objects for custom toolbar items or HTML element ID for toolbar template.

To use Toolbar, you need to inject `Toolbar` module in the `provide` section.

### Enable or disable toolbar items

You can enable/disable toolbar items by using the `enableItems` method.

## APP.VUE

```

<template>
    <div id="app">
        <ejs-button id='enable' cssClass='e-flat'
        @click.native='enable'>Enable</ejs-button>
        <ejs-button id='disable' cssClass='e-flat'
        @click.native='disable'>Disable</ejs-button>
        <ejs-grid id='Grid' ref='grid' :dataSource='data' height='200px'
        :allowGrouping='true' :groupSettings='groupOptions' :toolbar='toolbar'
        :toolbarClick='clickHandler'>
            <e-columns>
                <e-column field='OrderID' headerText='Order ID'
                textAlign='Right' width=120></e-column>
                <e-column field='CustomerID' headerText='Customer ID'
                width=150></e-column>
                <e-column field='ShipCity' headerText='Ship City'
                width=150></e-column>
            </e-columns>
        </ejs-grid>
    </div>

```

```

        <e-column field='ShipName' headerText='Ship Name'
width=150></e-column>
    </e-columns>
</ejs-grid>
</div>
</template>
<script>
import Vue from "vue";
import { GridPlugin, Toolbar, Group } from "@syncfusion/ej2-vue-grids";
import { ButtonPlugin } from "@syncfusion/ej2-vue-buttons";
import { data } from './datasource.js';
Vue.use(GridPlugin);
Vue.use(ButtonPlugin);
export default {
  data() {
    return {
      data: data,
      toolbar: ['Expand', 'Collapse'],
      groupOptions: { columns: ['CustomerID'] }
    };
  },
  methods: {
    clickHandler: function(args) {
      if (args.item.id === 'Grid_Collapse') { // Grid_Collapse is control
        id + '_' + toolbar value.
        this.$refs.grid.ej2Instances.groupModule.collapseAll();
      }
      if (args.item.id === "Grid_Expand") {
        this.$refs.grid.ej2Instances.groupModule.expandAll();
      }
    },
    enable: function() {
      this.$refs.grid.ej2Instances.toolbarModule.enableItems(['Grid_Collapse', 'Grid_Expand'], true); // Enable toolbar items.
    },
    disable: function() {
      this.$refs.grid.ej2Instances.toolbarModule.enableItems(['Grid_Collapse', 'Grid_Expand'], false); // Disable toolbar items.
    }
  },
  provide: {
    grid: [Toolbar]
  }
}
</script>
<style>
  @import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/grid/toolbar/default-cs6" %}

[Add toolbar at the bottom of Grid](#)

You can add the Grid toolbar component at the bottom of Grid using the ['created'](#) event.

**APP.VUE**

```

<template>
  <div id="app">
    <ejs-grid ref='grid' :dataSource='data' :toolbar='toolbar'
:created='created' height='175px'>
      <e-columns>
        <e-column field='OrderID' headerText='Order ID'
type='number' textAlign='Right' :isPrimaryKey='true'
:validationRules='orderIDRules' width='120'></e-column>
        <e-column field='CustomerID' headerText='Customer ID'
type='string' width='140'>
          </e-column>
        <e-column field='Freight' headerText='Freight' type='number'
textAlign='Right' format='C2' width='120'></e-column>
        <e-column field='OrderDate' headerText='OrderDate'
type='date' textAlign='Right' format='yMd' width='140'></e-column>
      </e-columns>
    </ejs-grid>
  </div>
</template>
<script>
import Vue from "vue";
import { GridPlugin, Toolbar } from "@syncfusion/ej2-vue-grids";
import { tbdata } from './datasource.js';
Vue.use(GridPlugin);
export default {
  data() {
    return {
      data: tbdata.slice(0, 8),
      toolbar: ['Print', 'Search'],
      orderIDRules: { required: true },
    };
  },
  methods: {
    created: function () {
      let toolbar = this.$refs.grid.ej2Instances.element.querySelector(".e-
toolbar");
      this.$refs.grid.ej2Instances.element.appendChild(toolbar);
    }
  },
  provide: {
    grid: [Toolbar]
  }
}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/grid/toolbar/default-cs7" %}

[See Also](#)

- [Toolbar Component](#)
- [How to apply overflowMode to grid toolbar in Vue Grid](#)



- [How to show the toolbar in both top and bottom of the Vue Grid](#)

Tool bar items in Vue Grid component

#### *Built-in toolbar items*

Built-in Toolbar Items execute standard actions of the Grid and it can be added by defining [toolbar](#) as a collection of built-in items. It renders the button with icon and text.

The following table shows Built-in toolbar items and its action.

Built-in Toolbar Items	Actions
----- -----	
Add	Add a new record.
Edit	Edit the selected record.
Update	Update edited record.
Delete	Delete the selected record.
Cancel	Cancel the edit state.
Search	Search the records by given key.
Print	Print the Grid.
ColumnChooser	Choose the column's visibility.
PdfExport	Exports grid to PDF document.
ExcelExport	Exports grid to Excel document.
CsvExport	Exports grid to CSV document.

#### **APP.VUE**

```

<template>
  <div id="app">
    <ejs-grid :dataSource='data' height='270px' :toolbar='toolbar'>
      <e-columns>
        <e-column field='OrderID' headerText='Order ID'
textAlign='Right' width=120></e-column>
        <e-column field='CustomerID' headerText='Customer ID'
width=150></e-column>
        <e-column field='ShipCity' headerText='Ship City'
width=150></e-column>
        <e-column field='ShipName' headerText='Ship Name'
width=150></e-column>
      </e-columns>
    </ejs-grid>
  </div>
</template>
<script>
import Vue from "vue";
import { GridPlugin, Toolbar } from "@syncfusion/ej2-vue-grids";
import { data } from './datasource.js';
Vue.use(GridPlugin);
export default {
  data() {

```

```

    return {
      data: data,
      toolbar: ['Print', 'Search']
    };
  },
  provide: {
    grid: [Toolbar]
  }
}
</script>
<style>
  @import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/grid/toolbar/default-cs1" %}

The [toolbar](#) has options to define both built-in and custom toolbar items.

#### Show only icons in built-in toolbar items

By default, the built-in toolbar items render as buttons with an icon and text. It is possible to hide the text and show only the icon using the following CSS style.

```

,
.e-toolbar .e-tbar-btn-text, .e-toolbar .e-toolbar-items .e-toolbar-item .e-tbar-btn-text {
display: none;
}
,

```

This is demonstrated in the following sample:

#### APP.VUE

```

<template>
  <div id="app">
    <ejs-grid :dataSource='data' :editSettings='editSettings'
height='270px' :toolbar='toolbar'>
      <e-columns>
        <e-column field='OrderID' headerText='Order ID'
textAlign='Right' width=120 isPrimaryKey="true"></e-column>
        <e-column field='CustomerID' headerText='Customer ID'
width=150></e-column>
        <e-column field='ShipCity' headerText='Ship City'
width=150></e-column>
        <e-column field='ShipName' headerText='Ship Name'
width=150></e-column>
      </e-columns>
    </ejs-grid>
  </div>
</template>
<script>
import Vue from "vue";
import { GridPlugin, Toolbar, Edit } from "@syncfusion/ej2-vue-grids";
import { data } from '../datasource.js';
Vue.use(GridPlugin);

```

```

export default {
  data() {
    return {
      data: data,
      editSettings: { allowEditing: true, allowAdding: true, allowDeleting:
true },
      toolbar: ['Add', 'Edit', 'Delete', 'Update', 'Cancel']
    };
  },
  provide: {
    grid: [Edit, Toolbar]
  }
}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
.e-toolbar .e-tbar-btn-text, .e-toolbar .e-toolbar-items .e-toolbar-item
.e-tbar-btn-text {
  display: none;
}
</style>

```

{% previewsample "page.domainurl/code-snippet/grid/toolbar/default-cs2" %}

#### Custom toolbar items

Custom toolbar items can be added by defining [toolbar](#) as a collection of `ItemModel` Actions for this customized toolbar items are defined in the [toolbarClick](#) event.

By default, Custom toolbar items are in position `Left`. You can change the position by using the `align` property. In the below sample, we have applied position `Right` for the `Collapse All` toolbar item.

#### APP.VUE

```

<template>
  <div id="app">
    <ejs-grid ref='grid' :dataSource='data' height='200px'
:allowGrouping='true' :groupSettings='groupOptions' :toolbar='toolbar'
:toolbarClick='clickHandler'>
      <e-columns>
        <e-column field='OrderID' headerText='Order ID'
textAlign='Right' width=120></e-column>
        <e-column field='CustomerID' headerText='Customer ID'
width=150></e-column>
        <e-column field='ShipCity' headerText='Ship City'
width=150></e-column>
        <e-column field='ShipName' headerText='Ship Name'
width=150></e-column>
      </e-columns>
    </ejs-grid>
  </div>
</template>
<script>
import Vue from "vue";
import { GridPlugin, Toolbar, Group } from "@syncfusion/ej2-vue-grids";
import { data } from './datasource.js';
Vue.use(GridPlugin);

```

```

export default {
  data() {
    return {
      data: data,
      toolbar: [{ text: 'Expand All', tooltipText: 'Expand All', prefixIcon:
'e-expand', id: 'expandall' }, { text: 'Collapse All', tooltipText:
'collection All', prefixIcon: 'e-collapse', id: 'collapseall' },
align: 'Right' ]},
      groupOptions: { columns: ['CustomerID'] }
    };
  },
  methods: {
    clickHandler: function(args) {
      if (args.item.id === 'expandall') {
        this.$refs.grid.ej2Instances.groupModule.expandAll();
      }
      if (args.item.id === "collapseall") {
        this.$refs.grid.ej2Instances.groupModule.collapseAll();
      }
    }
  },
  provide: {
    grid: [Toolbar, Group]
  }
}
</script>
<style>
  @import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/grid/toolbar/default-cs3" %}

\* The [toolbar](#) has options to define both built-in and custom toolbar items.

\* If a toolbar item does not match with built-in items, it will be treated as custom toolbar item.

#### *Both built-in and custom items in toolbar*

Grid have an option to use both built-in and custom toolbar items at same time.

In the below example, Add, Edit, Delete, Update, Cancel are built-in toolbar items and Click is custom toolbar item.

#### **APP.VUE**

```

<template>
  <div id="app">
    <ejs-grid ref='grid' :dataSource='data' height='200px'
:editSettings='editSettings' :toolbar='toolbar'
:toolbarClick='clickHandler'>
      <e-columns>
        <e-column field='OrderID' headerText='Order ID'
textAlign='Right' width=120></e-column>
        <e-column field='CustomerID' headerText='Customer ID'
width=150></e-column>
        <e-column field='ShipCity' headerText='Ship City'
width=150></e-column>
      </e-columns>
    </div>
  </template>

```

```

        <e-column field='ShipName' headerText='Ship Name'
width=150></e-column>
    </e-columns>
</ejs-grid>
</div>
</template>
<script>
import Vue from "vue";
import { GridPlugin, Toolbar, Edit } from "@syncfusion/ej2-vue-grids";
import { data } from './datasource.js';
Vue.use(GridPlugin);
export default {
  data() {
    return {
      data: data,
      editSettings: { allowEditing: true, allowAdding: true, allowDeleting:
true },
      toolbar: ['Add', 'Edit', 'Delete', 'Update', 'Cancel', 'Print',
'Search', { text: 'Click', tooltipText: 'Click', prefixIcon: 'e-expand', id:
'Click' }]
    };
  },
  methods: {
    clickHandler: function(args) {
      if (args.item.id === 'Click') {
        alert("Custom Toolbar Click...");
      }
    }
  },
  provide: {
    grid: [Toolbar, Edit]
  }
}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
.e-expand::before {
  content: '\e5b8';
}
</style>

```

{% previewsample "page.domainurl/code-snippet/grid/toolbar/default-cs4" %}

#### Custom toolbar component in a specific position

By default, Custom toolbar items are in the Left position. You can change the position by using the [align](#) property. In the following sample, we have applied the Right position for the Collapse All toolbar item, Left for the Expand All toolbar item, and Center for the Search toolbar item.

#### APP.VUE

```

<template>
  <div id="app">
    <ejs-grid ref='grid' :dataSource='data' height='200px'
:allowGrouping='true' :groupSettings='groupOptions' :toolbar='toolbar'
:toolbarClick='clickHandler'>
      <e-columns>

```

```

        <e-column field='EmployeeID' headerText='Employee ID'
textAlign='Right' width=120></e-column>
        <e-column field='FirstName' headerText='First Name'
width=150></e-column>
        <e-column field='Country' headerText='Country'
width=150></e-column>
        <e-column field='PostalCode' headerText='Postal Code'
width=150></e-column>
    </e-columns>
</ejs-grid>
</div>
</template>
<script>
import Vue from "vue";
import { GridPlugin, Toolbar, Group } from "@syncfusion/ej2-vue-grids";
import { employeeData } from './datasource.js';
Vue.use(GridPlugin);
export default {
    data() {
        return {
            data: employeeData,
            toolbar: [{ text: 'Expand All', tooltipText: 'Expand All', prefixIcon:
'e-expand', id: 'expandall' }, { text: 'Collapse All', tooltipText:
'collection All', prefixIcon: 'e-collapse', id: 'collapseall' },
align: 'Right' }, { text: 'Search', align: 'Center' } ],
            groupOptions: { columns: ['FirstName'] }
        };
    },
    methods: {
        clickHandler: function(args) {
            if (args.item.id === 'expandall') {
                this.$refs.grid.ej2Instances.groupModule.expandAll();
            }
            if (args.item.id === "collapseall") {
                this.$refs.grid.ej2Instances.groupModule.collapseAll();
            }
        }
    },
    provide: {
        grid: [Toolbar, Group]
    }
}
</script>
<style>
    @import "https://cdn.syncfusion.com/ej2/material.css";
    .e-icons.e-collapse::before {
        content: "\e834";
    }
    .e-icons.e-expand::before {
        content: "\e83d";
    }
</style>

```

{% previewsample "page.domainurl/code-snippet/grid/toolbar/default-cs5" %}

### Custom tool bar in Vue Grid component

Custom Toolbar is used to customize the whole toolbar. It can be added by defining **toolbarTemplate**. Actions for this toolbar template items are defined in the **clicked** event in toolbar.

#### APP.VUE

```
<template>
  <div id="app">
    <ejs-grid ref='grid' id='Grid' :dataSource='data'
:allowGrouping='true' :groupSettings='groupOptions'
:toolbarTemplate='toolbar'>
      <e-columns>
        <e-column field='OrderID' headerText='Order ID'
type='number' textAlign='Right' isPrimaryKey='true' width=120></e-column>
        <e-column field='CustomerID' headerText='Customer ID'
type='string' width=150></e-column>
        <e-column field='ShipCity' headerText='Ship City'
type='string' width=150></e-column>
        <e-column field='ShipName' headerText='Ship Name'
type='string' width=150></e-column>
      </e-columns>
    </ejs-grid>
  </div>
</template>
<script>
import Vue from "vue";
import { GridPlugin, Toolbar, Group } from "@syncfusion/ej2-vue-grids";
import { ToolbarPlugin } from "@syncfusion/ej2-vue-navigations";
import { cdata } from './datasource.js';
Vue.use(GridPlugin);
Vue.use(ToolbarPlugin);
export default {
  data() {
    return {
      data: cdata,
      groupOptions: { columns: ['CustomerID'] },
      toolbar: function () {
        return {
          template: Vue.component('custom-toolbar', {
            template: `<ejs-toolbar :clicked='clickHandler'>
              <e-items>
                <e-item id='collapse' prefixIcon='e-
collapse'></e-item>
              </e-items>
            </ejs-toolbar>`,
            data: function () {
              return {};
            },
            methods: {
              clickHandler(args) {
                let target = args.originalEvent.target.closest('button');
                //find clicked button
                var gridInstance =
document.getElementById("Grid").ej2_instances[0];
```

```

        if (target.id === 'collapse') {
            //collapse all expanded grouped row
            gridInstance.groupModule.collapseAll();
        }
    }
}
}))
};
}
};
},
provide: {
    grid: [Toolbar, Group]
}
}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
.e-icons.e-collapse::before {
    content: "\e834";
}
</style>

```

{% previewsample "page.domainurl/code-snippet/grid/toolbar/custom-cs1" %}

## Pdf Export

### Pdf export in Vue Grid component

PDF export allows exporting Grid data to PDF document. You need to use the [pdfExport](#) method for exporting. To enable PDF export in the grid, set the [allowPdfExport](#) as true.

To use PDF export, inject **PdfExport** module in the **provide** section.

### APP.VUE

```

<template>
    <div id="app">
        <ejs-grid ref='grid' id='Grid' :dataSource='data'
        :toolbar='toolbarOptions' height='272px' :allowPdfExport='true'
        :toolbarClick='toolbarClick'>
            <e-columns>
                <e-column field='OrderID' headerText='Order ID'
                textAlign='Right' width=120></e-column>
                <e-column field='CustomerID' headerText='Customer ID'
                width=150></e-column>
                <e-column field='ShipCity' headerText='Ship City'
                width=150></e-column>
                <e-column field='ShipName' headerText='Ship Name'
                width=150></e-column>
            </e-columns>
        </ejs-grid>
    </div>
</template>
<script>
import Vue from "vue";
import { GridPlugin, Toolbar, PdfExport } from "@syncfusion/ej2-vue-grids";
import { data } from './datasource.js';

```



```

Vue.use(GridPlugin);
export default {
  data() {
    return {
      data: data,
      toolbarOptions: ['PdfExport']
    };
  },
  methods: {
    toolbarClick(args) {
      if (args.item.id === 'Grid_pdfexport') { // 'Grid_pdfexport' -> Grid
        component id + _ + toolbar item name
        this.$refs.grid.pdfExport();
      }
    }
  },
  provide: {
    grid: [Toolbar, PdfExport]
  }
}
</script>
<style>
  @import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/grid/pdf/default-cs14" %}

#### Show spinner while exporting

You can show/ hide spinner component while exporting the grid using `showSpinner/ hideSpinner` methods. You can use `toolbarClick` event to show spinner before exporting and hide a spinner in the `pdfExportComplete` or `excelExportComplete` event after the exporting.

In the `toolbarClick` event, based on the parameter `args.item.id` as `Gridpdfexport` or `Gridexcelexport` we can call the `showSpinner` method from grid instance.

In the `pdfExportComplete` or `excelExportComplete` event, We can call the `hideSpinner` method.

In the below demo, we have rendered the default spinner component when exporting the grid.

#### APP.VUE

```

<template>
  <div id="app">
    <ejs-grid ref='grid' id='Grid' :dataSource='data'
      :toolbar='toolbarOptions' height='272px' :allowPdfExport='true'
      :allowExcelExport='true' :excelExportComplete='excelExportComplete'
      :pdfExportComplete='pdfExportComplete' :toolbarClick='toolbarClick'>
      <e-columns>
        <e-column field='OrderID' headerText='Order ID'
          textAlign='Right' width=120></e-column>
        <e-column field='CustomerID' headerText='Customer ID'
          :visible='false' width=150></e-column>
        <e-column field='ShipCity' headerText='Ship City'
          width=150></e-column>
        <e-column field='ShipName' headerText='Ship Name'
          width=150></e-column>
      </e-columns>
    </div>
  </div>

```

```

        </e-columns>
      </ejs-grid>
    </div>
  </template>
  <script>
import Vue from "vue";
import { GridPlugin, Toolbar, PdfExport, ExcelExport } from
"@syncfusion/ej2-vue-grids";
import { data } from './datasource.js';
Vue.use(GridPlugin);
export default {
  data() {
    return {
      data: data,
      toolbarOptions: ['PdfExport', 'ExcelExport']
    };
  },
  methods: {
    toolbarClick(args) {
      if (args.item.id === 'Grid_pdfexport') {
        this.$refs.grid.showSpinner();
        this.$refs.grid.pdfExport();
      }
      else if (args.item.id === 'Grid_excelexport') {
        this.$refs.grid.showSpinner();
        this.$refs.grid.excelExport();
      }
    }
    pdfExportComplete(args) {
      this.$refs.grid.hideSpinner();
    }
    excelExportComplete(args) {
      this.$refs.grid.hideSpinner();
    }
  },
  provide: {
    grid: [Toolbar, PdfExport, ExcelExport]
  }
}
  </script>
  <style>
    @import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
  </style>

```

{% previewsample "page.domainurl/code-snippet/grid/pdf/default-cs15" %}

#### Custom data source

PDF export provides an option to define datasource dynamically before exporting. To export data dynamically, define the **dataSource** in [pdfExportProperties](#).

#### APP.VUE

```

<template>
  <div id="app">

```

```

    <ejs-grid ref='grid' id='Grid' :dataSource='data'
:toolbar='toolbarOptions' height='272px' :allowPaging='true'
:allowPdfExport='true' :toolbarClick='toolbarClick'>
        <e-columns>
            <e-column field='OrderID' headerText='Order ID'
textAlign='Right' width=120></e-column>
            <e-column field='CustomerID' headerText='Customer ID'
width=150></e-column>
            <e-column field='ShipCity' headerText='Ship City'
width=150></e-column>
            <e-column field='ShipName' headerText='Ship Name'
width=150></e-column>
        </e-columns>
    </ejs-grid>
</div>
</template>
<script>
import Vue from "vue";
import { GridPlugin, Toolbar, PdfExport } from "@syncfusion/ej2-vue-grids";
import { data } from './datasource.js';
Vue.use(GridPlugin);
export default {
  data() {
    return {
      data: data,
      toolbarOptions: ['PdfExport']
    };
  },
  methods: {
    toolbarClick(args) {
      if (args.item.id === 'Grid_pdfexport') { // 'Grid_pdfexport' -> Grid
        component id + _ + toolbar item name
        let pdfExportProperties = {
          dataSource: data,
        };
        this.$refs.grid.pdfExport(pdfExportProperties);
      }
    }
  },
  provide: {
    grid: [Toolbar, PdfExport]
  }
}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/grid/pdf/default-cs16" %}

#### [Passing additional parameters to the server when exporting](#)

You can pass the additional parameter in the **query** property by invoking **addParams** method. In the **toolbarClick** event, you can define params as key and value pair so it will receive at the server side when exporting.

In the below example, we have passed `recordcount` as `12` using `addParams` method.

### APP.VUE

```
<template>
  <div id="app">
    <ejs-grid ref='grid' id='Grid' :dataSource='data'
:toolbar='toolbarOptions' height='272px' :allowPdfExport='true'
:allowExcelExport='true' :excelExportComplete='excelExportComplete'
:pdfExportComplete='pdfExportComplete' :toolbarClick='toolbarClick'>
      <e-columns>
        <e-column field='OrderID' headerText='Order ID'
textAlign='Right' width=120></e-column>
        <e-column field='CustomerID' headerText='Customer ID'
:visible='false' width=150></e-column>
        <e-column field='ShipCity' headerText='Ship City'
width=150></e-column>
        <e-column field='ShipName' headerText='Ship Name'
width=150></e-column>
      </e-columns>
    </ejs-grid>
  </div>
</template>
<script>
import Vue from "vue";
import { GridPlugin, Toolbar, PdfExport, ExcelExport } from
"@syncfusion/ej2-vue-grids";
import { Query } from "@syncfusion/ej2-data";
import { data } from './datasource.js';
Vue.use(GridPlugin);
export default {
  data() {
    return {
      data: data,
      toolbarOptions: ['PdfExport', 'ExcelExport'],
      queryClone: ""
    };
  },
  methods: {
    toolbarClick(args) {
      if (args.item.id === 'Grid_pdfexport') {
        debugger
        this.queryClone = this.$refs.grid.query;
        this.$refs.grid.query = new Query().addParams("recordcount",
"12")
        this.$refs.grid.pdfExport();
      }
      else if (args.item.id === 'Grid_excelexport') {
        this.queryClone = this.$refs.grid.query;
        this.$refs.grid.query = new Query().addParams("recordcount",
"12")
        this.$refs.grid.excelExport();
      }
    }
  },
  pdfExportComplete(args) {
    this.$refs.grid.query = this.queryClone();
  }
}
```

```

        excelExportComplete(args) {
            this.$refs.grid.query = this.queryClone();
        },
        provide: {
            grid: [Toolbar, PdfExport, ExcelExport]
        }
    }
</script>
<style>
    @import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/grid/pdf/default-cs17" %}

[See Also](#)

- [Exporting Grid in Cordova application](#)

### Export multiple grids in Vue Grid component

The PDF export provides an option to export multiple grids to the same or different pages of a PDF file. Each grid is identified by its unique ID. You can specify which grid to export by listing their IDs in the `exportGrids` property.

#### Same page

PDF exporting provides support for exporting multiple grids on the same page. To export the grids on the same page, define `multipleExport.type` as **AppendToPage** in `exportProperties`. It also has an option to provide blank space between the grids. This blank space can be defined by using `multipleExport.blankSpace` property.

### APP.VUE

```

<template>
    <div id="app">
        <p><b>First Grid:</b></p>
        <ejs-grid ref='grid1' id='FirstGrid' :dataSource='fData'
        :toolbar='toolbarOptions' :exportGrids='exportGrids' :allowPdfExport='true'
        :toolbarClick='toolbarClick'>
            <e-columns>
                <e-column field='OrderID' headerText='Order ID'
                textAlign='Right' width=120></e-column>
                <e-column field='CustomerID' headerText='Customer ID'
                width=150></e-column>
                <e-column field='ShipCity' headerText='Ship City'
                width=150></e-column>
                <e-column field='ShipName' headerText='Ship Name'
                width=150></e-column>
            </e-columns>
        </ejs-grid>
        <p><b>Second Grid:</b></p>
        <ejs-grid ref='grid2' id='SecondGrid' :dataSource='sData'
        :allowPdfExport='true'>
            <e-columns>

```

```

        <e-column field='EmployeeID' headerText='Employee ID'
textAlign='Right' width=120></e-column>
        <e-column field='FirstName' headerText='FirstName'
width=150></e-column>
        <e-column field='LastName' headerText='Last Name'
width=150></e-column>
        <e-column field='City' headerText='City' width=150></e-
column>
    </e-columns>
</ejs-grid>
</div>
</template>
<script>
import Vue from "vue";
import { GridPlugin, Toolbar, PdfExport } from "@syncfusion/ej2-vue-grids";
import { data, employeeData } from './datasource.js';
Vue.use(GridPlugin);
export default {
  data() {
    return {
      fData: data.slice(0, 5),
      sData: employeeData.slice(0, 5),
      toolbarOptions: ['PdfExport'],
      exportGrids: ['FirstGrid', 'SecondGrid'],
    };
  },
  methods: {
    toolbarClick(args) {
      if (args.item.id === 'FirstGrid_pdfexport') { // 'Grid_pdfexport' ->
Grid component id + _ + toolbar item name
        let appendPdfExportProperties = {
          multipleExport: { type: "AppendToPage", blankSpace: 10 }
        };
        this.$refs.grid1.pdfExport(appendPdfExportProperties, true);
      }
    },
  },
  provide: {
    grid: [Toolbar, PdfExport]
  }
}
</script>
<style>
  @import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/grid/pdf/multiple-cs1" %}

### [New page](#)

PDF export functionality enables the exporting of multiple grids into separate pages (each grid on a new page) within the PDF file. To achieve this, you can specify `multipleExport.type` as `NewPage` in `exportProperties`.

### APP.VUE

```
<template>
```

```

<div id="app">
  <p><b>First Grid:</b></p>
  <ejs-grid ref='grid1' id='FirstGrid' :dataSource='fData'
:toolbar='toolbarOptions' :exportGrids='exportGrids' :allowPdfExport='true'
:toolbarClick='toolbarClick'>
    <e-columns>
      <e-column field='OrderID' headerText='Order ID'
textAlign='Right' width=120></e-column>
      <e-column field='CustomerID' headerText='Customer ID'
width=150></e-column>
      <e-column field='ShipCity' headerText='Ship City'
width=150></e-column>
      <e-column field='ShipName' headerText='Ship Name'
width=150></e-column>
    </e-columns>
  </ejs-grid>
  <p><b>Second Grid:</b></p>
  <ejs-grid ref='grid2' id='SecondGrid' :dataSource='sData'
:allowPdfExport='true'>
    <e-columns>
      <e-column field='EmployeeID' headerText='Employee ID'
textAlign='Right' width=120></e-column>
      <e-column field='FirstName' headerText='FirstName'
width=150></e-column>
      <e-column field='LastName' headerText='Last Name'
width=150></e-column>
      <e-column field='City' headerText='City' width=150></e-
column>
    </e-columns>
  </ejs-grid>
</div>
</template>
<script>
import Vue from "vue";
import { GridPlugin, Toolbar, PdfExport } from "@syncfusion/ej2-vue-grids";
import { data, employeeData } from './datasource.js';
Vue.use(GridPlugin);
export default {
  data() {
    return {
      fData: data.slice(0, 5),
      sData: employeeData.slice(0, 5),
      toolbarOptions: ['PdfExport'],
      exportGrids: ['FirstGrid', 'SecondGrid'],
    };
  },
  methods: {
    toolbarClick(args) {
      if (args.item.id === 'FirstGrid_pdfexport') { // 'Grid_pdfexport' ->
Grid component id + _ + toolbar item name
        let appendPdfExportProperties = {
          multipleExport: { type: 'NewPage' }
        };
        this.$refs.grid1.pdfExport(appendPdfExportProperties, true);
      }
    }
  },
},

```

```

    provide: {
      grid: [Toolbar, PdfExport]
    }
  }
</script>
<style>
  @import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/grid/pdf/multiple-cs3" %}

## Pdf export options in Vue Grid component

### [Export current page](#)

PDF export provides an option to export the current page into PDF. To export current page, define the [exportType](#) to **CurrentPage**.

## APP.VUE

```

<template>
  <div id="app">
    <ejs-grid ref='grid' id='Grid' :dataSource='data'
      :toolbar='toolbarOptions' height='220px' :allowPaging='true'
      :allowPdfExport='true' :toolbarClick='toolbarClick'>
      <e-columns>
        <e-column field='OrderID' headerText='Order ID'
          textAlign='Right' width=120></e-column>
        <e-column field='CustomerID' headerText='Customer ID'
          width=150></e-column>
        <e-column field='ShipCity' headerText='Ship City'
          width=150></e-column>
        <e-column field='ShipName' headerText='Ship Name'
          width=150></e-column>
      </e-columns>
    </ejs-grid>
  </div>
</template>
<script>
import Vue from "vue";
import { GridPlugin, Toolbar, PdfExport, Page } from "@syncfusion/ej2-vue-grids";
import { data } from './datasource.js';
Vue.use(GridPlugin);
export default {
  data() {
    return {
      data: data,
      toolbarOptions: ['PdfExport']
    };
  },
  methods: {
    toolbarClick(args) {
      if (args.item.id === 'Grid_pdfexport') { // 'Grid_pdfexport' -> Grid
        component id + _ + toolbar item name
        let pdfExportProperties = {
          exportType: 'CurrentPage'
        };
      }
    }
  }
};

```



```

        this.$refs.grid.pdfExport(pdfExportProperties);
    }
},
provide: {
    grid: [Toolbar, PdfExport, Page]
}
}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/grid/pdf/default-cs7" %}

### *Export the selected records only*

You can export the selected records data by passing it to `exportProperties.dataSource` property in the `toolbarClick` event.

In the below exporting demo, We can get the selected records using `getSelectedRecords` method and pass the selected data to `PdfExport` or `excelExport` property.

### **APP.VUE**

```

<template>
  <div id="app">
    <ejs-grid ref='grid' :dataSource='data' :toolbar='toolbarOptions'
    :allowPaging='true' :allowFiltering='true' :allowPdfExport='true'
    :allowExcelExport='true' :pageSettings='pageSettings'
    :toolbarClick='toolbarClick' :selectionSettings='selectionOption'>
      <e-columns>
        <e-column field='OrderID' headerText='Order ID'
        textAlign='Right' width=100></e-column>
        <e-column field='CustomerID' headerText='Customer ID'
        width=150></e-column>
        <e-column field='ShipCity' headerText='Ship City'
        width=150></e-column>
      </e-columns>
    </ejs-grid>
  </div>
</template>
<script>
import Vue from "vue";
import { GridPlugin, Toolbar, PdfExport, Filter, Page, ExcelExport } from
"@syncfusion/ej2-vue-grids";
import { data } from './datasource.js';
import { DataManager } from "@syncfusion/ej2-data";
Vue.use(GridPlugin);
export default {
  data: () => {
    return {
      data: data,
      toolbarOptions: ['PdfExport', 'ExcelExport'],
      pageSettings: { pageSize: 5, pageCount: 5 },
      selectionOption: { type: 'Multiple' }
    };
  }
};

```

```

    },
    methods: {
        toolbarClick(args: ClickEventArgs) {
            if (args['item'].id.indexOf("pdfexport") !== -1) {
                let selectedRecords =
this.$refs.grid.getSelectedRecords();
                let exportProperties = {
                    dataSource: selectedRecords
                };
                this.$refs.grid.pdfExport(exportProperties);
            }
            else if (args['item'].id.indexOf("excelexport") !== -1) {
                let selectedRecords =
this.$refs.grid.getSelectedRecords();
                let exportProperties = {
                    dataSource: selectedRecords
                };
                this.$refs.grid.excelExport(exportProperties);
            }
        },
    },
    provide: {
        grid: [Toolbar, PdfExport, Filter, Page, ExcelExport]
    },
}
</script>
<style>
    @import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/grid/how-to/export-filtered-data-cs3" %}

### [Export filtered data only](#)

You can export the filtered data by defining the resulted data in `exportProperties.dataSource` before export.

In the below Pdf exporting demo, We have gotten the filtered data by applying filter query to the grid data and then defines the resulted data in `exportProperties.dataSource` and pass it to `pdfExport` method.

### **APP.VUE**

```

<template>
    <div id="app">
        <ejs-grid ref='grid' :dataSource='data' :toolbar='toolbarOptions'
:allowPaging='true' :allowFiltering='true' :allowPdfExport='true'
:pageSettings='pageSettings'
        :toolbarClick='toolbarClick'>
            <e-columns>
                <e-column field='OrderID' headerText='Order ID'
textAlign='Right' width=100></e-column>
                <e-column field='CustomerID' headerText='Customer ID'
width=150></e-column>
                <e-column field='ShipCity' headerText='Ship City'
width=150></e-column>
            </e-columns>
        </div>
    </template>

```

```

        </ejs-grid>
    </div>
</template>
<script>
import Vue from "vue";
import { GridPlugin, Toolbar, PdfExport, Filter, Page } from "@syncfusion/ej2-vue-grids";
import { data } from './datasource.js';
import { DataManager } from "@syncfusion/ej2-data";
Vue.use(GridPlugin);
export default {
    data: () => {
        return {
            data: data,
            toolbarOptions: ['PdfExport'],
            pageSettings: { pageSize: 5, pageCount: 5 }
        };
    },
    methods: {
        toolbarClick: function (args) {
            if (args['item'].id.indexOf("pdfexport") !== -1) {
                let pdfdata;
                let query =
this.$refs.grid.ej2Instances.renderModule.data.generateQuery(); // get grid
corresponding query
                for(var i=0; i<query.queries.length; i++){
                    if(query.queries[i].fn=='onPage'){
                        query.queries.splice(i,1); // remove page query to get all
records
                        break;
                    }
                }
                let fdata = new DataManager({ json:
data}).executeQuery(query).then((e) => {
                    pdfdata = e.result; // get all filtered records
                    let exportProperties = {
                        dataSource: pdfdata,
                    };
                    this.$refs.grid.pdfExport(exportProperties);
                }).catch((e) => true);
            }
        },
        provide: {
            grid: [Toolbar, PdfExport, Filter, Page]
        },
    }
}
</script>
<style>
    @import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/grid/how-to/export-filtered-data-cs4" %}

*Export hidden columns*

PDF export provides an option to export hidden columns of Grid by defining the [includeHiddenColumn](#) as **true**.

**APP.VUE**

```
<template>
  <div id="app">
    <ejs-grid ref='grid' id='Grid' :dataSource='data'
:toolbar='toolbarOptions' height='272px' :allowPaging='true'
:allowPdfExport='true' :toolbarClick='toolbarClick'>
      <e-columns>
        <e-column field='OrderID' headerText='Order ID'
textAlign='Right' width=120></e-column>
        <e-column field='CustomerID' headerText='Customer ID'
width=150></e-column>
        <e-column field='ShipCity' headerText='Ship City'
width=150 :visible='false'></e-column>
        <e-column field='ShipName' headerText='Ship Name'
width=150></e-column>
      </e-columns>
    </ejs-grid>
  </div>
</template>
<script>
import Vue from "vue";
import { GridPlugin, Toolbar, PdfExport } from "@syncfusion/ej2-vue-grids";
import { data } from './datasource.js';
Vue.use(GridPlugin);
export default {
  data() {
    return {
      data: data,
      toolbarOptions: ['PdfExport']
    };
  },
  methods: {
    toolbarClick(args) {
      if (args.item.id === 'Grid_pdfexport') { // 'Grid_pdfexport' -> Grid
        component id + _ + toolbar item name
        let pdfExportProperties = {
          includeHiddenColumn: true
        };
        this.$refs.grid.pdfExport(pdfExportProperties);
      }
    }
  },
  provide: {
    grid: [Toolbar, PdfExport]
  }
}
</script>
<style>
  @import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
</style>
```

```
{% previewsample "page.domainurl/code-snippet/grid/pdf/default-cs8" %}
```

### Show or hide columns

You can show a hidden column or hide a visible column while exporting the grid using [toolbarClick](#) and [pdfExportComplete](#) events.

In the [toolbarClick](#) event, based on **args.item.id** as **Grid\_pdfexport**. We can show or hide columns by setting [column.visible](#) property to **true** or **false** respectively.

In the pdfExportComplete event, We have reversed the state back to the previous state.

In the below example, we have **CustomerID** as a hidden column in the grid. While exporting, we have changed **CustomerID** to visible column and **ShipCity** as hidden column.

### APP.VUE

```
<template>
  <div id="app">
    <ejs-grid ref='grid' id='Grid' :dataSource='data'
:toolbar='toolbarOptions' height='272px' :allowPdfExport='true'
:pdfExportComplete='pdfExportComplete'
:toolbarClick='toolbarClick'>
      <e-columns>
        <e-column field='OrderID' headerText='Order ID'
textAlign='Right' width=120></e-column>
        <e-column field='CustomerID' headerText='Customer ID'
:visible='false' width=150></e-column>
        <e-column field='ShipCity' headerText='Ship City'
width=150></e-column>
        <e-column field='ShipName' headerText='Ship Name'
width=150></e-column>
      </e-columns>
    </ejs-grid>
  </div>
</template>
<script>
import Vue from "vue";
import { GridPlugin, Toolbar, PdfExport } from "@syncfusion/ej2-vue-grids";
import { data } from './datasource.js';
Vue.use(GridPlugin);
export default {
  data() {
    return {
      data: data,
      toolbarOptions: ['PdfExport']
    };
  },
  methods: {
    toolbarClick(args) {
      if (args.item.id === 'Grid_pdfexport') {
        this.$refs.grid.getColumns()[1].visible = true;
        this.$refs.grid.getColumns()[3].visible = false;
        this.$refs.grid.pdfExport();
      }
    },
    pdfExportComplete(args) {
      this.$refs.grid.getColumns()[1].visible = false;
      this.$refs.grid.getColumns()[3].visible = true;
    }
  }
}
```

```

    }
  },
  provide: {
    grid: [Toolbar, PdfExport]
  }
}
</script>
<style>
  @import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/grid/pdf/default-cs9" %}

#### [Change page orientation](#)

Page orientation can be changed Landscape(Default Portrait) for the exported document using the [pdfExportProperties](#).

### **APP.VUE**

```

<template>
  <div id="app">
    <ejs-grid ref='grid' id='Grid' :dataSource='data'
:toolbar='toolbarOptions' height='272px' :allowPaging='true'
:allowPdfExport='true' :toolbarClick='toolbarClick'>
      <e-columns>
        <e-column field='OrderID' headerText='Order ID'
textAlign='Right' width=120></e-column>
        <e-column field='CustomerID' headerText='Customer ID'
width=150></e-column>
        <e-column field='ShipCity' headerText='Ship City'
width=150></e-column>
        <e-column field='ShipName' headerText='Ship Name'
width=150></e-column>
      </e-columns>
    </ejs-grid>
  </div>
</template>
<script>
import Vue from "vue";
import { GridPlugin, Toolbar, PdfExport } from "@syncfusion/ej2-vue-grids";
import { data } from './datasource.js';
Vue.use(GridPlugin);
export default {
  data() {
    return {
      data: data,
      toolbarOptions: ['PdfExport']
    };
  },
  methods: {
    toolbarClick(args) {
      if (args.item.id === 'Grid_pdfexport') { // 'Grid_pdfexport' -> Grid
        component id + _ + toolbar item name
        let pdfExportProperties = {
          pageOrientation: 'Landscape',
        };
      }
    }
  }
}

```

```

        this.$refs.grid.pdfExport(pdfExportProperties);
    }
},
provide: {
    grid: [Toolbar, PdfExport]
}
}
</script>
<style>
    @import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/grid/pdf/default-cs10" %}

### [Change page size](#)

Page size can be customized for the exported document using the [pdfExportProperties](#). Supported page sizes are:

- Letter
- Note
- Legal
- A0
- A1
- A2
- A3
- A5
- A6
- A7
- A8
- A9
- B0
- B1
- B2
- B3
- B4
- B5
- Archa
- Archb
- Archc
- Archd
- Arche
- Flsa
- HalfLetter
- Letter11x17
- Ledger

### **APP.VUE**

```
<template>
```

```

<div id="app">
  <ejs-grid ref='grid' id='Grid' :dataSource='data'
:toolbar='toolbarOptions' height='272px' :allowPaging='true'
:allowPdfExport='true' :toolbarClick='toolbarClick'>
    <e-columns>
      <e-column field='OrderID' headerText='Order ID'
textAlign='Right' width=120></e-column>
      <e-column field='CustomerID' headerText='Customer ID'
width=150></e-column>
      <e-column field='ShipCity' headerText='Ship City'
width=150></e-column>
      <e-column field='ShipName' headerText='Ship Name'
width=150></e-column>
    </e-columns>
  </ejs-grid>
</div>
</template>
<script>
import Vue from "vue";
import { GridPlugin, Toolbar, PdfExport } from "@syncfusion/ej2-vue-grids";
import { data } from './datasource.js';
Vue.use(GridPlugin);
export default {
  data() {
    return {
      data: data,
      toolbarOptions: ['PdfExport']
    };
  },
  methods: {
    toolbarClick(args) {
      if (args.item.id === 'Grid_pdfexport') { // 'Grid_pdfexport' -> Grid
component id + _ + toolbar item name
        let pdfExportProperties = {
          pageSize: 'Letter'
        };
        this.$refs.grid.pdfExport(pdfExportProperties);
      }
    }
  },
  provide: {
    grid: [Toolbar, PdfExport]
  }
}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/grid/pdf/default-cs11" %}

### Define file name

You can assign the file name for the exported document by defining [fileName](#) property of [PdfExportProperties](#) in the `toolbarClick` event.

### APP.VUE



```

<template>
  <div id="app">
    <ejs-grid ref='grid' id='Grid' :dataSource='data'
:allowPaging='true' :allowPdfExport='true' :toolbar='toolbarOptions'
:toolbarClick='toolbarClick' height='258px'>
      <e-columns>
        <e-column field='OrderID' headerText='Order ID'
textAlign='Right' type='number' width=120></e-column>
        <e-column field='CustomerID' headerText='Customer ID'
type='string' width=140></e-column>
        <e-column field='Freight' headerText='Freight'
textAlign='Right' format='C2' width=120></e-column>
        <e-column field='OrderDate' headerText='Order Date'
textAlign='Right' type='date' format='yMd' width=150></e-column>
      </e-columns>
    </ejs-grid>
  </div>
</template>
<script>
import Vue from "vue";
import { GridPlugin, Toolbar, Page, PdfExport } from "@syncfusion/ej2-vue-
grids";
import { data } from './datasource.js';
Vue.use(GridPlugin);
export default {
  data() {
    return {
      data: data,
      toolbarOptions: ["PdfExport"],
      pageSettings: { pageSize: 7 }
    };
  },
  methods: {
    toolbarClick(args) {
      if (args["item"].id === "Grid_pdfexport") {
        let exportProperties = {
          fileName: "new.pdf"
        };
        this.$refs.grid.pdfExport(exportProperties);
      }
    }
  },
  provide: {
    grid: [Toolbar, Page, PdfExport]
  }
}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/grid/pdf/default-cs12" %}

*Font customization**Default fonts*

By default, grid uses **Helvetica** font in the exported document. You can change the default font by using [pdfExportProperties.theme](#) property. The available default fonts are,

- Helvetica
- TimesRoman
- Courier
- Symbol
- ZapfDingbats

The code example for changing default font,

```
`ts
import { PdfStandardFont, PdfFontFamily, PdfFontStyle } from '@syncfusion/ej2-pdf-export';
...
let pdfExportProperties = {
  theme: {
    header: {font: new PdfStandardFont(PdfFontFamily.TimesRoman, 11, PdfFontStyle.Bold)},
    caption: { font: new PdfStandardFont(PdfFontFamily.TimesRoman, 9) },
    record: { font: new PdfStandardFont(PdfFontFamily.TimesRoman, 10) }
  }
};
`
```

*Add custom font*

You can change the default font of Grid header, content and caption cells in the exported document by using [pdfExportProperties.theme](#) property.

In the following example, we have used Algeria font to export the grid.

**APP.VUE**

```
<template>
  <div id="app">
    <ejs-grid ref='grid' id='Grid' :dataSource='data'
      :toolbar='toolbarOptions' height='272px' :allowPaging='true'
      :allowPdfExport='true' :toolbarClick='toolbarClick'>
      <e-columns>
        <e-column field='OrderID' headerText='Order ID'
          textAlign='Right' width=120></e-column>
        <e-column field='CustomerID' headerText='Customer ID'
          width=150></e-column>
        <e-column field='ShipCity' headerText='Ship City'
          width=150></e-column>
        <e-column field='ShipName' headerText='Ship Name'
          width=150></e-column>
      </e-columns>
    </div>
  </template>
```

```

        </ejs-grid>
    </div>
</template>
<script>
import Vue from "vue";
import { GridPlugin, Toolbar, PdfExport } from "@syncfusion/ej2-vue-grids";
import { data, adventProFont } from '../datasource.js';
import { PdfTrueTypeFont } from '@syncfusion/ej2-pdf-export';
Vue.use(GridPlugin);
export default {
  data() {
    return {
      data: data,
      toolbarOptions: ['PdfExport']
    };
  },
  methods: {
    toolbarClick(args) {
      if (args.item.id === 'Grid_pdfexport') { // 'Grid_pdfexport' -> Grid
        component id + _ + toolbar item name
        let pdfExportProperties = {
          theme: {
            header: { font: new PdfTrueTypeFont(adventProFont, 12)
            },
            caption: { font: new PdfTrueTypeFont(adventProFont, 10)
            },
            record: { font: new PdfTrueTypeFont(adventProFont, 9) }
          }
        };
        this.$refs.grid.pdfExport(pdfExportProperties);
      }
    },
  },
  provide: {
    grid: [Toolbar, PdfExport]
  }
}
</script>
<style>
  @import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/grid/pdf/default-cs13" %}

**PdfTrueTypeFont** accepts base 64 format of the Custom Font.

#### [Export grid as blob](#)

The Grid offers an option to export the data as a **Blob** instead of downloading it as a file in the browser. To export the grid as a Blob, set the **isBlob** parameter to **true** in the [pdfExport](#) method. The grid returns the promise of a blob in the [pdfExportComplete](#) event.

The following example demonstrates how to obtain the blob data of the exported grid by executing the promise in the **pdfExportComplete** event.

#### **APP.VUE**

```

<template>
  <div id="app">
    <ejs-grid ref='grid' id='Grid' :dataSource='data'
:toolbar='toolbarOptions' height='220px' :allowPaging='true'
:allowPdfExport='true' :toolbarClick='toolbarClick'
:pdfExportComplete='pdfExportComplete">
      <e-columns>
        <e-column field='OrderID' headerText='Order ID'
textAlign='Right' width=120></e-column>
        <e-column field='CustomerID' headerText='Customer ID'
width=150></e-column>
        <e-column field='ShipCity' headerText='Ship City'
width=150></e-column>
        <e-column field='ShipName' headerText='Ship Name'
width=150></e-column>
      </e-columns>
    </ejs-grid>
  </div>
</template>
<script>
import Vue from "vue";
import { GridPlugin, Toolbar, PdfExport, Page } from "@syncfusion/ej2-vue-
grids";
import { data } from './datasource.js';
Vue.use(GridPlugin);
export default {
  data() {
    return {
      data: data,
      toolbarOptions: ['PdfExport']
    };
  },
  methods: {
    toolbarClick(args) {
      if (args.item.id === 'Grid_pdfexport') {
        // pass fourth parameter as true to get the blob data of exported
grid
        this.$refs.grid.pdfExport(null, null, null, true);
      }
    },
    pdfExportComplete(args) {
      // execute the promise to get the blob data
      args.promise.then((e) => {
        console.log(e.blobData);
      });
    },
  },
  provide: {
    grid: [Toolbar, PdfExport, Page]
  }
}
</script>
<style>
  @import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
</style>

```

```
{% previewsample "page.domainurl/code-snippet/grid/pdf/export-grid-as-blob-cs1" %}
```

## Pdf cell style customization in Vue Grid component

### Conditional cell formatting

Grid cells in the exported PDF can be customized or formatted using [pdfQueryCellInfo](#) event. In this event, we can format the grid cells of exported PDF document based on the column cell value.

In the below sample, we have set the background color for **Freight** column in the exported document by **args.cell** and **backgroundColor** property.

### APP.VUE

```
<template>
  <div id="app">
    <ejs-grid ref='grid' id='Grid' :dataSource='data'
:toolbar='toolbarOptions' height='272px' :allowPdfExport='true'
:queryCellInfo='queryCellInfo'
:pdfQueryCellInfo='pdfQueryCellInfo' :toolbarClick='toolbarClick'>
      <e-columns>
        <e-column field='OrderID' headerText='Order ID'
textAlign='Right' width=120></e-column>
        <e-column field='CustomerID' headerText='Customer ID'
width=150></e-column>
        <e-column field='Freight' headerText='Freight'
width=150></e-column>
        <e-column field='ShipName' headerText='Ship Name'
width=150></e-column>
      </e-columns>
    </ejs-grid>
  </div>
</template>
<script>
import Vue from "vue";
import { GridPlugin, Toolbar, PdfExport } from "@syncfusion/ej2-vue-grids";
import { data } from './datasource.js';
Vue.use(GridPlugin);
export default {
  data() {
    return {
      data: data,
      toolbarOptions: ['PdfExport']
    };
  },
  methods: {
    toolbarClick(args) {
      if (args.item.id === 'Grid_pdfexport') {
        this.$refs.grid.pdfExport();
      }
    },
    pdfQueryCellInfo(args) {
      if (args.column.field === 'Freight') {
        if (args.value < 30) {
          args.style = { backgroundColor: '#99ffcc' };
        }
        else if (args.value < 60) {
          args.style = { backgroundColor: '#ffffb3' };
        }
      }
    }
  }
}
```

```

        else {
            args.style = {backgroundColor: '#ff704d'};
        }
    },
    queryCellInfo(args) {
        if(args.column.field == 'Freight') {
            if(args.data['Freight'] < 30) {
                args.cell.bgColor = '#99ffcc';
            }
            else if(args.data['Freight'] < 60) {
                args.cell.bgColor = '#ffffb3';
            }
            else {
                args.cell.bgColor = '#ff704d';
            }
        }
    }
},
provide: {
    grid: [Toolbar, PdfExport]
}
}
</script>
<style>
    @import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/grid/pdf/default-cs5" %}

### Theme

PDF export provides an option to include theme for exported PDF document.

To apply theme in exported PDF, define the [Link to the Video](#) in [pdfExportProperties](#).

### APP.VUE

```

<template>
    <div id="app">
        <ejs-grid ref='grid' id='Grid' :dataSource='data'
        :toolbar='toolbarOptions' height='272px' :allowPaging='true'
        :allowPdfExport='true' :toolbarClick='toolbarClick'>
            <e-columns>
                <e-column field='OrderID' headerText='Order ID'
                textAlign='Right' width=120></e-column>
                <e-column field='CustomerID' headerText='Customer ID'
                width=150></e-column>
                <e-column field='ShipCity' headerText='Ship City'
                width=150></e-column>
                <e-column field='ShipName' headerText='Ship Name'
                width=150></e-column>
            </e-columns>
        </ejs-grid>
    </div>
</template>
<script>
    import Vue from "vue";

```

```

import { GridPlugin, Toolbar, PdfExport } from "@syncfusion/ej2-vue-grids";
import { data } from './datasource.js';
Vue.use(GridPlugin);
export default {
  data() {
    return {
      data: data,
      toolbarOptions: ['PdfExport']
    };
  },
  methods: {
    toolbarClick(args) {
      if (args.item.id === 'Grid_pdfexport') { // 'Grid_pdfexport' -> Grid
        component id + _ + toolbar item name
        let pdfExportProperties = {
          theme: {
            header: {
              fontColor: '#64FA50', fontName: 'Calibri', fontSize:
17, bold: true, border: { color: '#64FA50', lineStyle: 'Thin' }
            },
            record: {
              fontColor: '#64FA50', fontName: 'Calibri', fontSize:
17, bold: true
            },
            caption: {
              fontColor: '#64FA50', fontName: 'Calibri', fontSize:
17, bold: true
            }
          }
        };
        this.$refs.grid.pdfExport(pdfExportProperties);
      }
    },
    provide: {
      grid: [Toolbar, PdfExport]
    }
  }
}
</script>
<style>
  @import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/grid/pdf/default-cs6" %}

By default, material theme is applied to exported PDF document.

#### Adding header and footer in Vue Grid component

PDF export provides an option to customize mapping of grid to exported PDF document.

To get start quickly with PDF export Options, you can check on this video:

You can customize text, page number, line, page size and changing orientation in header and footer.

#### Write a text in header and footer

You can add text either in Header or Footer of exported PDF document using [pdfExportProperties](#).

```
`ts
let pdfExportProperties = {
  header: {
    fromTop: 0,
    height: 130,
    contents: [
      {
        type: 'Text',
        value: "Northwind Traders",
        position: { x: 0, y: 50 },
        style: { textBrushColor: '#000000', fontSize: 13 }
      },
    ]
  }
}
```

#### *Draw a line in header and footer*

you can add line either in Header or Footer of the exported PDF document.

Supported line styles:

- Dash
- Dot
- DashDot
- DashDotDot
- Solid

```
`ts
let pdfExportProperties = {
  header: {
    fromTop: 0,
    height: 130,
    contents: [
      {
        type: 'Line',
        style: { penColor: '#000080', penSize: 2, dashStyle: 'Solid' },
        points: { x1: 0, y1: 4, x2: 685, y2: 4 }
      }
    ]
  }
}
```



```
}
]
}
}
`ts
```

#### *Add page number in header and footer*

you can add page number either in Header or Footer of exported PDF document.

Supported page number types:

- LowerLatin - a, b, c,
- UpperLatin - A, B, C,
- LowerRoman - i, ii, iii,
- UpperRoman - I, II, III,
- Number - 1,2,3.

```
`ts
let pdfExportProperties = {
header: {
fromTop: 0,
height: 130,
contents: [
{
type: 'PageNumber',
pageNumberType: 'Number',
format: 'Page {$current} of {$total}', //optional
position: { x: 0, y: 25 },
style: { textBrushColor: '#ffff80', fontSize: 15, hAlign: 'Center' }
}
]
}
}
`ts
```

#### *Insert an image in header and footer*

Image (Base64 string with .jpeg format) can be added in the exported document in header/footer using the [pdfExportProperties](#).

```
`ts
let pdfExportProperties = {
```

```

header: {
  fromTop: 0,
  height: 130,
  contents: [
    {
      type: 'Image',
      src: image,
      position: { x: 435, y: 10 },
      size: { height: 100, width: 250 },
    }
  ]
}
,

```

The below code illustrates the pdf export customization.

#### **APP.VUE**

```

<template>
  <div id="app">
    <ejs-grid ref='grid' id='Grid' :dataSource='data'
      :toolbar='toolbarOptions' height='272px' :allowPaging='true'
      :allowPdfExport='true' :toolbarClick='toolbarClick'>
      <e-columns>
        <e-column field='OrderID' headerText='Order ID'
          textAlign='Right' width=120></e-column>
        <e-column field='CustomerID' headerText='Customer ID'
          width=150></e-column>
        <e-column field='ShipCity' headerText='Ship City'
          width=150></e-column>
        <e-column field='ShipName' headerText='Ship Name'
          width=150></e-column>
      </e-columns>
    </ejs-grid>
  </div>
</template>
<script>
import Vue from "vue";
import { GridPlugin, Toolbar, PdfExport } from "@syncfusion/ej2-vue-grids";
import { data } from './datasource.js';
import { image } from './image.js';
Vue.use(GridPlugin);
export default {
  data() {
    return {
      data: data,
      toolbarOptions: ['PdfExport']
    }
  }
}

```

```

    };
  },
  methods: {
    toolbarClick(args) {
      if (args.item.id === 'Grid_pdfexport') { // 'Grid_pdfexport' -> Grid
        component id + _ + toolbar item name
        let pdfExportProperties = {
          header: {
            fromTop: 0,
            height: 130,
            contents: [
              {
                type: 'Image',
                src: image,
                position: { x: 40, y: 10 },
                size: { height: 100, width: 250 }
              }
            ]
          },
          footer: {
            fromBottom: 160,
            height: 150,
            contents: [
              {
                type: 'PageNumber',
                pageNumberType: 'Arabic',
                format: 'Page { $current } of { $total }',
                position: { x: 0, y: 25 },
                style: { textBrushColor: '#ffff80', fontSize: 15
            }
          }
        ]
      }
    };
    this.$refs.grid.pdfExport(pdfExportProperties);
  }
},
provide: {
  grid: [Toolbar, PdfExport]
}
}
</script>
<style>
  @import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/grid/pdf/default-cs3" %}

[Repeat column header on every page](#)

By default, column header will be placed on the first page of the pdf document but you can display column header on every page using **repeatHeader** property of **pdfGrid**.

In the below sample, we have enabled **repeatHeader** property in [pdfHeaderQueryCellInfo](#) event to show the header on every page.

**APP.VUE**

```

<template>
  <div id="app">
    <ejs-grid ref='grid' id='Grid' :dataSource='data'
      :toolbar='toolbarOptions' height='272px' :allowPdfExport='true'
      :pdfHeaderQueryCellInfo='pdfHeaderQueryCellInfo'
      :toolbarClick='toolbarClick'>
      <e-columns>
        <e-column field='OrderID' headerText='Order ID'
textAlign='Right' width=120></e-column>
        <e-column field='CustomerID' headerText='Customer ID'
width=150></e-column>
        <e-column field='Freight' headerText='Freight'
width=150></e-column>
        <e-column field='ShipName' headerText='Ship Name'
width=150></e-column>
      </e-columns>
    </ejs-grid>
  </div>
</template>
<script>
import Vue from "vue";
import { GridPlugin, Toolbar, PdfExport } from "@syncfusion/ej2-vue-grids";
import { purchaseData } from '../datasource.js';
Vue.use(GridPlugin);
export default {
  data() {
    return {
      data: purchaseData,
      toolbarOptions: ['PdfExport']
    };
  },
  methods: {
    toolbarClick(args) {
      if (args.item.id === 'Grid_pdfexport') {
        this.$refs.grid.pdfExport();
      }
    },
    pdfHeaderQueryCellInfo(args) {
      args.cell.row.pdfGrid.repeatHeader = true;
    }
  },
  provide: {
    grid: [Toolbar, PdfExport]
  }
}
</script>
<style>
  @import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/grid/pdf/default-cs4" %}

### Exporting hierarchy grid in Vue Grid component

The grid have an option to export the hierarchy grid to pdf document. By default, grid will exports the master grid with expanded child grids alone. you can change the exporting option by using the **PdfExportProperties.hierarchyExportMode** property. The available options are,

- | Mode | Behavior |
- |-----|-----|
- | Expanded | Exports the master grid with expanded child grids. |
- | All | Exports the master grid with all the child grids. |
- | None | Exports the master grid alone. |

### APP.VUE

```
<template>
  <div id="app">
    <ejs-grid ref='grid' id='Grid' :dataSource='parentData'
:childGrid='childGrid' :toolbar='["PdfExport"]' :toolbarClick='toolbarClick'
:allowPdfExport='true'>
      <e-columns>
        <e-column field='EmployeeID' headerText='Employee ID'
textAlign='Right' width=120></e-column>
        <e-column field='FirstName' headerText='FirstName'
width=150></e-column>
        <e-column field='LastName' headerText='Last Name'
width=150></e-column>
        <e-column field='City' headerText='City' width=150></e-
column>
      </e-columns>
    </ejs-grid>
  </div>
</template>
<script>
import Vue from "vue";
import { GridPlugin, DetailRow, Toolbar, PdfExport, PdfExportProperties }
from "@syncfusion/ej2-vue-grids";
import { data, employeeData } from './datasource.js';
import { extend } from '@syncfusion/ej2-base';
Vue.use(GridPlugin);
export default {
  data() {
    return {
      parentData: employeeData,
      childGrid: {
        dataSource: data,
        queryString: 'EmployeeID',
        columns: [
          { field: 'OrderID', headerText: 'Order ID', textAlign: 'Right',
width: 120 },
          { field: 'CustomerID', headerText: 'Customer ID', width: 150 },
          { field: 'ShipCity', headerText: 'Ship City', width: 150 },
          { field: 'ShipName', headerText: 'Ship Name', width: 150 }
        ]
      }
    }
  }
}
```

```

    },
    methods: {
      toolbarClick: function( args) {
        if (args['item'].id === 'Grid_pdfexport') {
          let exportProperties = {
            hierarchyExportMode: "Expanded"
          };
          this.$refs.grid.pdfExport(exportProperties);
        }
      },
    },
    provide: {
      grid: [DetailRow, Toolbar, PdfExport]
    }
  }
</script>
<style>
  @import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/grid/hierarchy-grid/default-cs9" %}

### Exporting grid with templates in Vue Grid control

The grid offers the option to export the column, detail, and caption templates to a PDF document. The template contains images, hyperlinks, and customized text.

#### Exporting with column template

The PDF export functionality allows you to export Grid columns that include images, hyperlinks, and custom text to a PDF document.

In the following sample, the hyperlinks and images are exported to PDF using [hyperlink](#) and [image](#) properties in the [pdfQueryCellInfo](#) event.

PDF Export supports base64 string to export the images.

### APP.VUE

```

<template>
  <ejs-grid id="ColumnTemplateGrid" ref="grid" :dataSource="data"
  :allowPdfExport="true"
  :toolbar="toolbar" :toolbarClick="toolbarClick"
  :pdfQueryCellInfo="pdfQueryCellInfo" height=315>
    <e-columns>
      <e-column headerText="Employee Image" textAlign="Center"
      :template="imageTemplate" width="150"></e-column>
      <e-column field="EmployeeID" headerText="Employee ID"
      width="125"></e-column>
      <e-column field="FirstName" headerText="Name" width="120"></e-
      column>
      <e-column headerText="Email ID" :template="mailTemplate"
      width="170"></e-column>
    </e-columns>
  </ejs-grid>
</template>
<script>
import Vue from 'vue';

```

```

import { GridPlugin, PdfExport, Toolbar } from '@syncfusion/ej2-vue-grids';
import { employeeData } from './datasource.js';
Vue.use(GridPlugin);
export default {
  data: () => {
    return {
      data: employeeData,
      toolbar: ['PdfExport'],
      imageTemplate: function () {
        return { template : Vue.component('imageTemplate',{
          template: `<div class="image">
            
            </div>`,
          data: function() {
            return {
              data: {}
            }
          },
        })},
      },
      mailTemplate: function () {
        return { template : Vue.component('mailTemplate',{
          template: `<div class="link">
            <a
:href="'mailto:'+data.EmailID">{{data.EmailID}}</a></div>
            </div>`,
          data: function() {
            return {
              data: {}
            }
          },
        })},
      },
    };
  },
  methods: {
    toolbarClick: function (args) {
      if (args.item.id === 'ColumnTemplateGrid_pdfexport') {
        this.$refs.grid.pdfExport();
      }
    },
    pdfQueryCellInfo: function(args) {
      if (args.column.headerText === 'Employee Image') {
        args.image = {
          base64: args.data.EmployeeImage,
          height: 50,
          width: 50,
        };
      }
      if (args.column.headerText === 'Email ID') {
        args.hyperLink = {
          target: 'mailto:' + args.data.EmailID,
          displayText: args.data.EmailID,
        };
      }
    }
  }
}





```

```

    },
    provide: {
      grid: [PdfExport, Toolbar],
    },
  }
</script>
<style>
  @import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
  .image img {
    height: 55px;
    width: 55px;
    border-radius: 50px;
    box-shadow: inset 0 0 1px #e0e0e0, inset 0 0 14px rgba(0,0,0,0.2);
  }
</style>

```

```
{% previewsample "page.domainurl/code-snippet/grid/pdf/column-template-export-cs1" %}
```

PDF Export		
Employee Image	Name	Email ID
	Nancy	nancy@domain.com
	Andrew	andrew@domain.com
	Janet	janet@domain.com
	Margaret	margaret@domain.com

#### Exporting with detail template

By default, the grid will export the parent grid with expanded detail rows alone. Change the exporting option by using the `PdfExportProperties.hierarchyExportMode` property. The available options are:

Mode	Behavior	
-----	-----	



| Expanded | Exports the parent grid with expanded detail rows. |

| All | Exports the parent grid with all the detail rows. |

| None | Exports the parent grid alone. |

The detail rows in the exported PDF can be customized or formatted using the [exportDetailTemplate](#) event. In this event, the detail rows of the PDF document are formatted in accordance with their parent row details.

In the following sample, the detail row content is formatted by specifying the [columnCount](#), [columnHeader](#), and [rows](#) properties using its [parentRow](#) details. This allows for the creation of detail rows in the PDF document. Additionally, custom styles can be applied to specific cells using the [style](#) property.

If [columnCount](#) is not provided, the columns in the detail row of the PDF grid will be generated based on the count of the [columnHeader](#)/[rows](#) first row's [cells](#).

When using [rowSpan](#), it is essential to provide the cell's [index](#) for proper functionality.

#### APP.VUE

```
<template>
  <ejs-grid id="DetailTemplateGrid" ref="grid" :dataSource="data"
  :detailTemplate="detailTemplate" :toolbar="toolbar"
  :toolbarClick="toolbarClick" :exportDetailTemplate="exportDetailTemplate"
  height=315 :allowPdfExport="true">
    <e-columns>
      <e-column field="Category" headerText="Category" width="140"
      textAlign="Right"></e-column>
      <e-column field="ProductID" headerText="Product ID" width="120"></e-
      column>
      <e-column field="Status" headerText="Status" width="120"></e-column>
    </e-columns>
  </ejs-grid>
</template>
<script>
import Vue from 'vue';
import { GridPlugin, DetailRow, PdfExport, Toolbar } from '@syncfusion/ej2-
vue-grids';
import { employeeData } from './datasource.js';
Vue.use(GridPlugin);
export default {
  data: () => {
    return {
      data: employeeData,
      toolbar: ['PdfExport'],
      detailTemplate: function () {
        return {
          template: Vue.component('detailTemplate', {
            template: `<table class="detailtable" width="100%">
              <colgroup>
                <col width="40%" />
                <col width="60%" />
              </colgroup>
              <thead>
                <tr>
```

```

        <th colspan="2" style="font-weight: 500;text-align:
center;background-color: #ADD8E6;">
        Product Details
    </th>
</tr>
</thead>
<tbody>
    <tr>
        <td rowspan="4" style="text-align: center;">
            
        </td>
        <td>
            <span style="font-weight: 500;color:
#0a76ff;">Offers: {{data.Offers}} </span>
        </td>
    </tr>
    <tr>
        <td>
            <span>Available: {{data.Available}} </span>
        </td>
    </tr>
    <tr>
        <td>
            <span class="link">
                Contact: <a
:href="'mailto:'+data.Contact">{{data.Contact}}</a>
            </span>
        </td>
    </tr>
    <tr>
        <td>
            <span style="font-weight: 500;color: #0a76ff;">
Ratings: {{data.Ratings}}</span>
        </td>
    </tr>
    <tr>
        <td style="text-align: center;">
            <span> {{data.productDesc}}</span>
        </td>
        <td>
            <span>{{data.ReturnPolicy}}</span>
        </td>
    </tr>
    <tr>
        <td style="text-align: center;">
            <span style="font-weight: 500;" >
{{data.Cost}}</span>
        </td>
        <td>
            <span>{{data.Cancellation}}</span>
        </td>
    </tr>
    <tr>
        <td style="text-align: center;">
            <span :class="data.Status" style="font-weight: 500;"
> {{data.Status}}</span>

```

```

                </td>
                <td>
                    <span style="font-weight: 500;color:
#0a76ff;">{{data.Delivery}}</span>
                </td>
            </tr>
        </tbody>
    </table>`,
    data: function () {
        return {
            data: {},
        }
    },
    }),
}
},
};
},
methods: {
    toolbarClick: function (args) {
        if (args.item.id === 'DetailTemplateGrid_pdfexport') {
            this.$refs.grid.pdfExport({ hierarchyExportMode: "Expanded" });
        }
    },
    exportDetailTemplate: function (args) {
        args.value = {
            columnCount: 2,
            columnHeader: [
                {
                    cells: [{
                        index: 0, colSpan: 2, value: 'Product Details',
                        style: { backgroundColor: '#ADD8E6', pdfTextAlign:
'Center', bold: true }
                    }]
                }
            ],
            rows: [
                {
                    cells: [
                        {
                            index: 0, rowspan: 4, image: { base64:
args.parentRow.data['ProductImg'], width: 80 }
                        },
                        {
                            index: 1, value: "Offers: " +
args.parentRow.data['Offers'],
                            style: { fontColor: '#0A76FF', fontSize: 15 }
                        },
                    ]
                },
                {
                    cells: [
                        {
                            index: 1, value: 'Available: ' +
args.parentRow.data['Available']
                        }
                    ]
                },
            ],
        },
    },
}

```

```

        {
            cells: [
                {
                    index: 1, value: 'Contact: ',
                    hyperlink: {
                        target: 'mailto:' +
args.parentRow.data['Contact'],
                        displayText: args.parentRow.data['Contact']
                    }
                }
            ]
        },
        {
            cells: [
                {
                    index: 1, value: 'Ratings: ' +
args.parentRow.data['Ratings'],
                    style: { fontColor: '#0A76FF', fontSize: 15 }
                }
            ]
        },
        {
            cells: [
                {
                    index: 0, value:
args.parentRow.data['productDesc'],
                    style: { pdfTextAlignment: 'Center' }
                },
                { index: 1, value:
args.parentRow.data['ReturnPolicy'] }
            ]
        },
        {
            cells: [
                {
                    index: 0, value: args.parentRow.data['Cost'],
                    style: { bold: true, pdfTextAlignment: 'Center' }
                }
            ]
        },
        { index: 1, value:
args.parentRow.data['Cancellation'] }
    ],
    {
        cells: [
            {
                index: 0, value: args.parentRow.data['Status'],
                style: {
                    fontColor: args.parentRow.data['Status'] ===
'Available' ? '#00FF00' : '#FF0000',
                    pdfTextAlignment: 'Center', fontSize: 15
                }
            },
            {
                index: 1, value:
args.parentRow.data['Delivery'],
                style: { fontColor: '#0A76FF', fontSize: 15 }
            }
        ]
    }
}

```

```

    }
    ],
  },
  provide: {
    grid: [DetailRow, PdfExport, Toolbar],
  },
}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
.detailtable td {
  font-size: 13px;
  padding: 4px;
  max-width: 0;
  overflow: hidden;
  text-overflow: ellipsis;
  white-space: nowrap;
}
.photo {
  width: 100px;
  height: 100px;
  border-radius: 50px;
  box-shadow: inset 0 0 1px #e0e0e0, inset 0 0 14px rgba(0,0,0,0.2);
}
.Unavailable {
  color: #FF0000;
}
.Available {
  color: #00FF00;
}
@media screen and (max-width: 800px) and (min-width: 320px) {
  .photo {
    width: 70px;
    height: 70px;
  }
}
</style>

```

```
{% previewsample "page.domainurl/code-snippet/grid/pdf/detail-template-export-cs1" %}
```

PDF Export		
Category	Product ID	Status
▶ Suits/Slim	EJ-SU-01	Available
▶ Suits/Classic	EJ-SU-02	Available
▶ Suits/Formal	EJ-SU-03	Available
▶ Phants/Slim	EJ-PH-01	Available
▶ Phants/Classic	EJ-PH-02	Available
▶ Shirts/Slim	EJ-SH-01	Available
▶ Shirts/Formal	EJ-SH-02	Available

#### Exporting with caption template

The PDF export feature enables exporting of Grid with a caption template to an PDF document.

In the following sample, the customized caption text is exported to PDF using [captionText](#) property in the [exportGroupCaption](#) event.

#### APP.VUE

```

<template>
  <ejs-grid id="CaptionTemplateGrid" ref="grid" :dataSource="data"
:allowGrouping="true" :groupSettings="groupOptions"
  :allowPdfExport="true" :toolbar="toolbar" :toolbarClick="toolbarClick"
:exportGroupCaption="exportGroupCaption" height=315>
    <e-columns>
      <e-column field="EmployeeID" headerText="Employee ID"
width="120"></e-column>
      <e-column field="FirstName" headerText="Name" width="120"></e-
column>
      <e-column field="City" headerText="City"></e-column>
      <e-column field="Title" headerText="Title" width="170"></e-
column>
    </e-columns>
  </ejs-grid>
</template>
<script>
import Vue from 'vue';

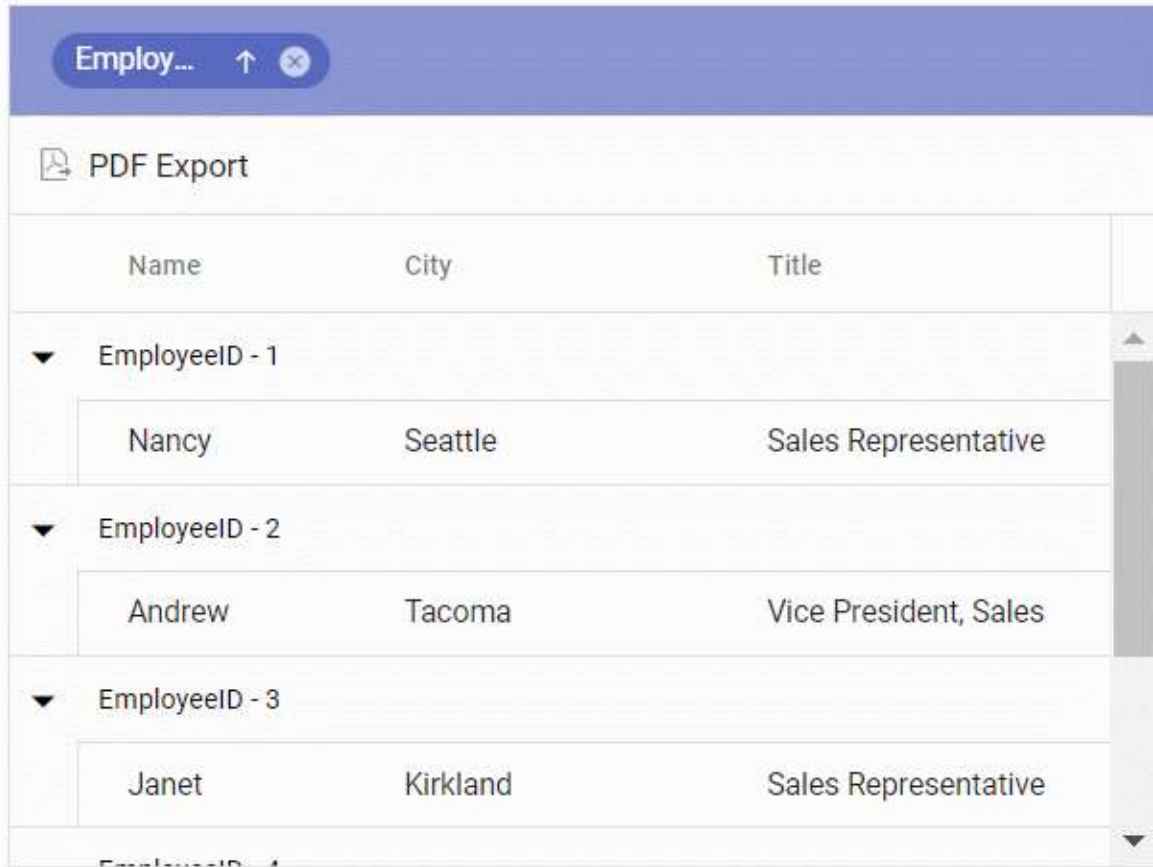
```

```

import { GridPlugin, Group, PdfExport, Toolbar } from '@syncfusion/ej2-vue-grids';
import { employeeData } from '../datasource.js';
Vue.use(GridPlugin);
export default {
  data: () => {
    return {
      data: employeeData,
      toolbar: ['PdfExport'],
      groupOptions: {
        columns: ['EmployeeID'],
        captionTemplate: '${field} - ${key}'
      }
    };
  },
  methods: {
    toolbarClick: function (args) {
      if (args.item.id === 'CaptionTemplateGrid_pdfexport') {
        this.$refs.grid.pdfExport();
      }
    },
    exportGroupCaption: function(args) {
      args.captionText = args.data.field + ' - ' + args.data.key;
    }
  },
  provide: {
    grid: [Group, PdfExport, Toolbar],
  },
}
</script>
<style>
  @import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/grid/pdf/caption-template-export-cs1" %}



The screenshot shows a web application interface. At the top, there is a blue header bar with a button labeled "Employ..." and two small icons (an upward arrow and a close 'x' icon). Below the header, there is a white bar with a PDF icon and the text "PDF Export". The main content area is a grid with three columns: "Name", "City", and "Title". The grid contains three rows of data, each preceded by a dropdown arrow and a label: "EmployeeID - 1", "EmployeeID - 2", and "EmployeeID - 3". The data rows are: Nancy (Seattle, Sales Representative), Andrew (Tacoma, Vice President, Sales), and Janet (Kirkland, Sales Representative). A vertical scrollbar is visible on the right side of the grid.

	Name	City	Title
▼ EmployeeID - 1	Nancy	Seattle	Sales Representative
▼ EmployeeID - 2	Andrew	Tacoma	Vice President, Sales
▼ EmployeeID - 3	Janet	Kirkland	Sales Representative

### Exporting grid in server in Vue Grid component

The Grid have an option to export the data to PDF in server side using Grid server export library.

#### Server dependencies

The Server side export functionality is shipped in the Syncfusion.EJ2.GridExport package, which is available in Essential Studio and [nuget.org](https://nuget.org). The following list of dependencies is required for Grid server side PDF exporting action.

- Syncfusion.EJ2
- Syncfusion.EJ2.GridExport

#### Server configuration

The following code snippet shows server configuration using ASP.NET Core Controller Action.

To Export the Grid in server side, You need to call the [serverPdfExport](#) method for passing the Grid properties to server exporting action.

```
`ts
public ActionResult PdfExport([FromForm] string gridModel)
{
    GridPdfExport exp = new GridPdfExport();
    Grid gridProperty = ConvertGridObject(gridModel);
```



```

return exp.PdfExport<OrdersDetails>(gridProperty, OrdersDetails.GetAllRecords());
}

private Grid ConvertGridObject(string gridProperty)
{
    Grid GridModel = (Grid)Newtonsoft.Json.JsonConvert.DeserializeObject(gridProperty, typeof(Grid));
    GridColumnModel cols =
    (GridColumnModel)Newtonsoft.Json.JsonConvert.DeserializeObject(gridProperty,
    typeof(GridColumnModel));
    GridModel.Columns = cols.columns;
    return GridModel;
}

public class GridColumnModel
{
    public List<GridColumn> columns { get; set; }
}
`
`

<template>
<div id="app">
<ejs-grid ref='grid' id='Grid' :dataSource='data' :toolbar='toolbarOptions' height='272px'
:toolbarClick='toolbarClick'>
<e-columns>
<e-column field='OrderID' headerText='Order ID' textAlign='Right' width=120></e-column>
<e-column field='CustomerID' headerText='Customer ID' width=150></e-column>
<e-column field='ShipCity' headerText='Ship City' width=150></e-column>
<e-column field='ShipName' headerText='Ship Name' width=150></e-column>
</e-columns>
</ejs-grid>
</div>
</template>
<script>
import Vue from "vue";
import { GridPlugin, Toolbar } from "@syncfusion/ej2-vue-grids";
import { data } from './datasource.js';

```

```

Vue.use(GridPlugin);
export default {
  data() {
    return {
      data: data,
      toolbarOptions: ['PdfExport']
    };
  },
  methods: {
    toolbarClick: function(args) {
      if (args.item.id === 'Gridpdfexport') { // 'Gridpdfexport' -> Grid component id + _ + toolbar item name
        this.$refs.grid.serverPdfExport('Home/PdfExport');
      }
    }
  },
  provide: {
    grid: [Toolbar]
  }
}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
</style>

```

Note: Refer to the GitHub sample for quick implementation and testing from [here](#).

#### *Export grid as memory stream*

The Grid offers an option to export the data as a memory stream instead of downloading it as a file in the browser. To obtain the memory stream of the exported grid, set the `AsMemoryStream` parameter to **true** in the [PdfExport](#) method.

The following code demonstrates how to get the memory stream of exported grid.

```

`ts
public object PdfExport(string gridModel)
{
  GridPdfExport exp = new GridPdfExport();

```

```

Grid gridProperty = ConvertGridObject(gridModel);
// pass third parameter as true to get the Memory Stream of exported grid data
return (MemoryStream)exp.PdfExport<OrdersDetails>(gridProperty, OrdersDetails.GetAllRecords(),
true);
}
`

```

#### *Merge grid's memory stream*

The [Essential PDF](#) library is used to merge multiple memory streams into a single stream. To learn more about the merge option, please refer to this [documentation](#).

You can merge a memory stream, a file stream, and a local file with the Grid's memory stream in the following ways:

#### *Merging with an existing memory stream*

If you already have a memory stream, you can directly use it to merge with the Grid's memory stream.

In the following code, the [Merge](#) method of the [PdfDocumentBase](#) class is used to merge the grid's memory stream with an existing memory stream.

```

`ts
using Syncfusion.Pdf;

public MemoryStream ms1; // defines existing memory stream
public object PdfExport(string gridModel)
{
    GridPdfExport exp = new GridPdfExport();
    Grid gridProperty = ConvertGridObject(gridModel);
    // get the memory stream of exported grid data
    MemoryStream ms2 = (MemoryStream)exp.PdfExport<OrdersDetails>(gridProperty,
OrdersDetails.GetAllRecords(), true);
    //Creates a PDF document.
    PdfDocument finalDoc = new PdfDocument();
    //Creates a PDF stream for merging. ms1 and ms2 represents the existing stream and grid's stream.
    Stream[] streams = { ms1, ms2 };
    //Merges PDFDocument.
    PdfDocumentBase.Merge(finalDoc, streams);
    //Save the document into stream.
    MemoryStream ms3 = new MemoryStream();
    finalDoc.Save(ms3);
    ms3.Position = 0;
}
`

```

```
//Close the document.
finalDoc.Close(true);
//Dispose the streams.
ms1.Dispose();
ms2.Dispose();
return ms3;
}
`ts
```

#### Merging with an existing file stream

If you already have a file stream, you can directly use it to merge with the Grid's memory stream. In the following code, the existing file stream is merged with the Grid's memory stream.

```
`ts
using Syncfusion.Pdf;

public FileStream fs1; // defines existing file stream
public ActionResult PdfExport(string gridModel)
{
    GridPdfExport exp = new GridPdfExport();
    Grid gridProperty = ConvertGridObject(gridModel);
    MemoryStream ms1 = (MemoryStream)exp.PdfExport<OrdersDetails>(gridProperty,
    OrdersDetails.GetAllRecords(), true);
    PdfDocument finalDoc = new PdfDocument();
    //fs1 and ms1 represents the existing stream and grid's stream.
    Stream[] streams = { fs1, ms1 };
    PdfDocumentBase.Merge(finalDoc, streams);
    MemoryStream ms3 = new MemoryStream();
    finalDoc.Save(ms3);
    ms3.Position = 0;
    return ms3;
}
`ts
```

#### Merging with a local file

To merge a local file with the Grid's memory stream, you need to convert it into a file stream before merging. In the following code, the existing local file is merged with the Grid's memory stream.

```
`ts
using Syncfusion.Pdf;
```

```
// get the file stream of local file
public FileStream fs1 = new FileStream("D:/PdfDoc.pdf", FileMode.Open, FileAccess.Read); //
PdfDoc.pdf is a local file which is located in local disk D.

public ActionResult PdfExport(string gridModel)
{
    GridPdfExport exp = new GridPdfExport();
    Grid gridProperty = ConvertGridObject(gridModel);
    MemoryStream ms1 = (MemoryStream)exp.PdfExport<OrdersDetails>(gridProperty,
    OrdersDetails.GetAllRecords(), true);
    PdfDocument finalDoc = new PdfDocument();
    //fs1 and ms1 represents the local file's stream and grid's stream.
    Stream[] streams = { fs1, ms1 };
    PdfDocumentBase.Merge(finalDoc, streams);
    MemoryStream ms3 = new MemoryStream();
    finalDoc.Save(ms3);
    ms3.Position = 0;
    return ms3;
}
,
```

#### Downloading the merged memory stream

You can download the merged memory stream by converting it into a `FileStreamResult`. In the following code, the merged memory stream is downloaded to the browser.

```
`ts
using Syncfusion.Pdf;

public ActionResult PdfExport(string gridModel)
{
    PdfDocument finalDoc = new PdfDocument();
    //ms1 and ms2 represents the streams needs to merge.
    Stream[] streams = { ms1, ms2 };
    PdfDocumentBase.Merge(finalDoc, streams);
    MemoryStream ms3 = new MemoryStream();
    finalDoc.Save(ms3);
    ms3.Position = 0;
    // Save the MemoryStream into FileStreamResult
```

```

FileStreamResult fileStreamResult = new FileStreamResult(ms3, "application/pdf");
fileStreamResult.FileNameDownload = "Export.pdf";
//Close the document.
finalDoc.Close(true);
//Dispose the streams.
ms1.Dispose();
ms2.Dispose();
// return the file
return fileStreamResult;
}

```

*Rotate a header text to a certain degree in the exported grid on the server side*

The Grid has support to customize the column header styles such as changing text orientation, the font color, and so on in the exported PDF file. To achieve this requirement, define the `BeginCellLayout` event of the `PdfExportProperties` with an event handler to perform the required action.

The `PdfHeaderQueryCellInfoEvent` will be triggered when creating a column header for the pdf document to be exported. Collect the column header details in this event and handle the custom in the `BeginCellLayout` event handler.

In the following demo, the `DrawString` method from the `Graphics` is used to rotate the header text of the column header inside the `BeginCellLayout` event handler.

A PDF exporting is not supported to rotate the column header on the client side.

```
`ts
```

```

public ActionResult PdfExport(string gridModel)
{
    GridPdfExport exp = new GridPdfExport();
    Grid gridProperty = ConvertGridObject(gridModel);
    gridProperty.ServerPdfHeaderQueryCellInfo = PdfHeaderQueryCellInfo;
    PdfGrid grid = new PdfGrid();
    PdfExportProperties pdfExportProperties = new PdfExportProperties();
    pdfExportProperties.IsRepeatHeader = true;
    pdfExportProperties.BeginCellLayout = new PdfGridBeginCellLayoutEventHandler(BeginCellEvent);
    gridProperty.ServerPdfQueryCellInfo = PdfQueryCellInfo;
    IEnumerable data = Utils.DataTableToJson(dt);
    var result = exp.PdfExport<dynamic>(gridProperty, data, pdfExportProperties);
    return result;
}

```

```

}

public void BeginCellEvent(object sender, PdfGridBeginCellLayoutEventArgs args)
{
    PdfGrid grid = (PdfGrid)sender;
    var brush = new PdfSolidBrush(new PdfColor(Color.DimGray));
    args.Graphics.Save();
    args.Graphics.TranslateTransform(args.Bounds.X + 50, args.Bounds.Height + 40); // give the value for
    bounds x and Y by the user
    args.Graphics.RotateTransform(-60); // give the rotate degree value by the user
    // Draw the text at particular bounds.
    args.Graphics.DrawString(headerValues[args.CellIndex], new PdfStandardFont(PdfFontFamily.Helvetica,
    10), brush, new PointF(0, 0));
    if (args.IsHeaderRow)
    {
        grid.Headers[0].Cells[args.CellIndex].Value = string.Empty;
    }
    args.Graphics.Restore();
}

private void PdfHeaderQueryCellInfo(object pdf)
{
    ServerPdfHeaderQueryCellInfoEventArgs name = (ServerPdfHeaderQueryCellInfoEventArgs)pdf;
    PdfGrid grid = new PdfGrid();
    headerValues.Add(name.Column.HeaderText);
    var longestString = headerValues.Where(s => s.Length == headerValues.Max(m => m.Length)).First();
    PdfFont font = new PdfStandardFont(PdfFontFamily.Helvetica, 6);
    SizeF size = font.MeasureString(longestString);
    name.Headers[0].Height = size.Width * 2;
}

```

### Limitations

- The export feature for detail and caption templates is not supported in server-side exporting.
- Multiple grids exporting feature is not supported with server side exporting.

## Excel Export

### Excel exporting in Vue Grid component

The excel export allows exporting Grid data to Excel document. You need to use the

[excelExport](#) method for exporting. To enable Excel export in the grid, set the [allowExcelExport](#) as true.

To use excel export, inject **ExcelExport** module in the **provide** section.

### APP.VUE

```
<template>
  <div id="app">
    <ejs-grid ref='grid' id='Grid' :dataSource='data'
:toolbar='toolbarOptions' height='272px' :allowExcelExport='true'
:toolbarClick='toolbarClick'>
      <e-columns>
        <e-column field='OrderID' headerText='Order ID'
textAlign='Right' width=120></e-column>
        <e-column field='CustomerID' headerText='Customer ID'
width=150></e-column>
        <e-column field='ShipCity' headerText='Ship City'
width=150></e-column>
        <e-column field='ShipName' headerText='Ship Name'
width=150></e-column>
      </e-columns>
    </ejs-grid>
  </div>
</template>
<script>
import Vue from "vue";
import { GridPlugin, Toolbar, ExcelExport } from "@syncfusion/ej2-vue-
grids";
import { data } from './datasource.js';
Vue.use(GridPlugin);
export default {
  data() {
    return {
      data: data,
      toolbarOptions: ['ExcelExport']
    };
  },
  methods: {
    toolbarClick: function(args) {
      if (args.item.id === 'Grid_excelexport') { // 'Grid_excelexport' ->
Grid component id + _ + toolbar item name
        this.$refs.grid.excelExport();
      }
    }
  },
  provide: {
    grid: [Toolbar, ExcelExport]
  }
}
</script>
<style>
  @import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
</style>
```



```
{% previewsample "page.domainurl/code-snippet/grid/excel/default-cs10" %}
```

### Show spinner while exporting

You can show/ hide spinner component while exporting the grid using `showSpinner/ hideSpinner` methods. You can use `toolbarClick` event to show spinner before exporting and hide a spinner in the `pdfExportComplete` or `excelExportComplete` event after the exporting.

In the `toolbarClick` event, based on the parameter `args.item.id` as `Gridpdfexport` or `Gridexcelexport` we can call the `showSpinner` method from grid instance.

In the `pdfExportComplete` or `excelExportComplete` event, We can call the `hideSpinner` method.

In the below demo, we have rendered the default spinner component when exporting the grid.

### APP.VUE

```
<template>
  <div id="app">
    <ejs-grid ref='grid' id='Grid' :dataSource='data'
    :toolbar='toolbarOptions' height='272px' :allowPdfExport='true'
    :allowExcelExport='true' :excelExportComplete='excelExportComplete'
    :pdfExportComplete='pdfExportComplete' :toolbarClick='toolbarClick'>
      <e-columns>
        <e-column field='OrderID' headerText='Order ID'
        textAlign='Right' width=120></e-column>
        <e-column field='CustomerID' headerText='Customer ID'
        :visible='false' width=150></e-column>
        <e-column field='ShipCity' headerText='Ship City'
        width=150></e-column>
        <e-column field='ShipName' headerText='Ship Name'
        width=150></e-column>
      </e-columns>
    </ejs-grid>
  </div>
</template>
<script>
import Vue from "vue";
import { GridPlugin, Toolbar, PdfExport, ExcelExport } from
"@syncfusion/ej2-vue-grids";
import { data } from './datasource.js';
Vue.use(GridPlugin);
export default {
  data() {
    return {
      data: data,
      toolbarOptions: ['PdfExport', 'ExcelExport']
    };
  },
  methods: {
    toolbarClick(args) {
      if (args.item.id === 'Grid_pdfexport') {
        this.$refs.grid.showSpinner();
        this.$refs.grid.pdfExport();
      }
      else if (args.item.id === 'Grid_excelexport') {
```

```

        this.$refs.grid.showSpinner();
        this.$refs.grid.excelExport();
    }
}
pdfExportComplete(args) {
    this.$refs.grid.hideSpinner();
}
excelExportComplete(args) {
    this.$refs.grid.hideSpinner();
}
},
provide: {
    grid: [Toolbar, PdfExport, ExcelExport]
}
}
</script>
<style>
    @import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/grid/pdf/default-cs1" %}

#### Custom data source

The excel export provides an option to define datasource dynamically before exporting. To export data dynamically, define the **dataSource** in [excelExportProperties](#).

#### APP.VUE

```

<template>
    <div id="app">
        <ejs-grid ref='grid' id='Grid' :dataSource='data'
        :toolbar='toolbarOptions' height='270px' :allowPaging='true'
        :allowExcelExport='true' :toolbarClick='toolbarClick'>
            <e-columns>
                <e-column field='OrderID' headerText='Order ID'
                textAlign='Right' width=120></e-column>
                <e-column field='CustomerID' headerText='Customer ID'
                width=150></e-column>
                <e-column field='ShipCity' headerText='Ship City'
                width=150></e-column>
                <e-column field='ShipName' headerText='Ship Name'
                width=150></e-column>
            </e-columns>
        </ejs-grid>
    </div>
</template>
<script>
import Vue from "vue";
import { GridPlugin, Toolbar, ExcelExport } from "@syncfusion/ej2-vue-grids";
import { data } from '../datasource.js';
Vue.use(GridPlugin);
export default {
    data() {
        return {
            data: data,

```

```

        toolbarOptions: ['ExcelExport']
    };
},
methods: {
    toolbarClick: function(args) {
        if (args.item.id === 'Grid_excelexport') { // 'Grid_excelexport' ->
Grid component id + _ + toolbar item name
            let excelExportProperties = {
                dataSource: data
            };
            this.$refs.grid.excelExport(excelExportProperties);
        }
    },
    provide: {
        grid: [Toolbar, ExcelExport]
    }
}
</script>
<style>
    @import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/grid/excel/default-cs11" %}

#### Passing additional parameters to the server when exporting

You can pass the additional parameter in the `query` property by invoking `addParams` method. In the `toolbarClick` event, you can define params as key and value pair so it will receive at the server side when exporting.

In the below example, we have passed `recordcount` as `12` using `addParams` method.

#### APP.VUE

```

<template>
    <div id="app">
        <ejs-grid ref='grid' id='Grid' :dataSource='data'
:toolbar='toolbarOptions' height='272px' :allowPdfExport='true'
:allowExcelExport='true' :excelExportComplete='excelExportComplete'
:pdfExportComplete='pdfExportComplete' :toolbarClick='toolbarClick'>
            <e-columns>
                <e-column field='OrderID' headerText='Order ID'
textAlign='Right' width=120></e-column>
                <e-column field='CustomerID' headerText='Customer ID'
:visible='false' width=150></e-column>
                <e-column field='ShipCity' headerText='Ship City'
width=150></e-column>
                <e-column field='ShipName' headerText='Ship Name'
width=150></e-column>
            </e-columns>
        </ejs-grid>
    </div>
</template>
<script>
import Vue from "vue";

```

```

import { GridPlugin, Toolbar, PdfExport, ExcelExport } from
"@syncfusion/ej2-vue-grids";
import { Query } from "@syncfusion/ej2-data";
import { data } from '../datasource.js';
Vue.use(GridPlugin);
export default {
  data() {
    return {
      data: data,
      toolbarOptions: ['PdfExport', 'ExcelExport'],
      queryClone: ""
    };
  },
  methods: {
    toolbarClick(args) {
      if (args.item.id === 'Grid_pdfexport') {
        debugger
        this.queryClone = this.$refs.grid.query;
        this.$refs.grid.query = new Query().addParams("recordcount",
"12")

        this.$refs.grid.pdfExport();
      }
      else if (args.item.id === 'Grid_excelexport') {
        this.queryClone = this.$refs.grid.query;
        this.$refs.grid.query = new Query().addParams("recordcount",
"12")

        this.$refs.grid.excelExport();
      }
    }
    pdfExportComplete(args) {
      this.$refs.grid.query = this.queryClone();
    }
    excelExportComplete(args) {
      this.$refs.grid.query = this.queryClone();
    }
  },
  provide: {
    grid: [Toolbar, PdfExport, ExcelExport]
  }
}
</script>
<style>
  @import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/grid/pdf/default-cs2" %}

[See Also](#)

- [Exporting Grid in Cordova application](#)

### Export multiple grids in Vue Grid component

The Excel export provides an option to export multiple grid data in the same or different sheets of an Excel file. Each grid is identified by its unique ID. You can specify which grids to export by listing their IDs in the `exportGrids` property.

#### Same sheet

Excel exporting provides support for exporting multiple grids on the same sheet. To export the grids in the same sheet, define `multipleExport.type` as `AppendToSheet` in `excelExportProperties`. It also has an option to provide blank rows between the grids. These blank rows count can be defined by using the `multipleExport.blankRows` property.

#### APP.VUE

```
<template>
  <div id="app">
    <p><b>First Grid:</b></p>
    <ejs-grid ref='grid1' id='FirstGrid' :dataSource='fData'
:toolbar='toolbarOptions' :allowExcelExport='true'
:exportGrids='exportGrids' :toolbarClick='toolbarClick'>
      <e-columns>
        <e-column field='OrderID' headerText='Order ID'
textAlign='Right' width=120></e-column>
        <e-column field='CustomerID' headerText='Customer ID'
width=150></e-column>
        <e-column field='ShipCity' headerText='Ship City'
width=150></e-column>
        <e-column field='ShipName' headerText='Ship Name'
width=150></e-column>
      </e-columns>
    </ejs-grid>
    <p><b>Second Grid:</b></p>
    <ejs-grid ref='grid2' id='SecondGrid' :dataSource='sData'
:allowExcelExport='true'>
      <e-columns>
        <e-column field='EmployeeID' headerText='Employee ID'
textAlign='Right' width=120></e-column>
        <e-column field='FirstName' headerText='FirstName'
width=150></e-column>
        <e-column field='LastName' headerText='Last Name'
width=150></e-column>
        <e-column field='City' headerText='City' width=150></e-
column>
      </e-columns>
    </ejs-grid>
  </div>
</template>
<script>
import Vue from "vue";
import { GridPlugin, Toolbar, ExcelExport } from "@syncfusion/ej2-vue-
grids";
import { data, employeeData } from './datasource.js';
Vue.use(GridPlugin);
export default {
  data() {
    return {
      fData: data.slice(0, 5),
```

```

        sData: employeeData.slice(0, 5),
        toolbarOptions: ['ExcelExport'],
        exportGrids: ['FirstGrid', 'SecondGrid'],
    };
},
methods: {
    toolbarClick: function (args) {
        if (args.item.id === 'FirstGrid_excelexport') { //
            'Grid_excelexport' -> Grid component id + _ + toolbar item name
            let appendExcelExportProperties = {
                multipleExport: { type: 'AppendToSheet', blankRows: 2 }
            };
            this.$refs.grid1.excelExport(appendExcelExportProperties, true);
        }
    },
    provide: {
        grid: [Toolbar, ExcelExport]
    }
}
</script>
<style>
    @import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/grid/excel/multiple-cs1" %}

By default, [multipleExport.blankRows](#) value is 5.

#### New sheet

Excel export functionality enables the exporting of multiple grids onto separate sheets (each grid in new sheet of excel) within the Excel file. To achieve this, you can specify [multipleExport.type](#) as [Link to the Video](#) in `exportProperties`.

#### APP.VUE

```

<template>
    <div id="app">
        <p><b>First Grid:</b></p>
        <ejs-grid ref='grid1' id='FirstGrid' :dataSource='fData'
        :toolbar='toolbarOptions' :allowExcelExport='true'
        :exportGrids='exportGrids' :toolbarClick='toolbarClick'>
            <e-columns>
                <e-column field='OrderID' headerText='Order ID'
                textAlign='Right' width=120></e-column>
                <e-column field='CustomerID' headerText='Customer ID'
                width=150></e-column>
                <e-column field='ShipCity' headerText='Ship City'
                width=150></e-column>
                <e-column field='ShipName' headerText='Ship Name'
                width=150></e-column>
            </e-columns>
        </ejs-grid>
        <p><b>Second Grid:</b></p>
        <ejs-grid ref='grid2' id='SecondGrid' :dataSource='sData'
        :allowExcelExport='true'>

```

```

        <e-columns>
            <e-column field='EmployeeID' headerText='Employee ID'
textAlign='Right' width=120></e-column>
            <e-column field='FirstName' headerText='FirstName'
width=150></e-column>
            <e-column field='LastName' headerText='Last Name'
width=150></e-column>
            <e-column field='City' headerText='City' width=150></e-
column>
        </e-columns>
    </ejs-grid>
</div>
</template>
<script>
import Vue from "vue";
import { GridPlugin, Toolbar, ExcelExport } from "@syncfusion/ej2-vue-
grids";
import { data, employeeData } from './datasource.js';
Vue.use(GridPlugin);
export default {
    data() {
        return {
            fData: data.slice(0, 5),
            sData: employeeData.slice(0, 5),
            toolbarOptions: ['ExcelExport'],
            exportGrids: ['FirstGrid', 'SecondGrid'],
        };
    },
    methods: {
        toolbarClick : function (args) {
            if (args.item.id === 'FirstGrid_excelelexport') { //
'Grid_excelelexport' -> Grid component id + _ + toolbar item name
                let appendExcelExportProperties = {
                    multipleExport: { type: 'NewSheet' }
                };
                this.$refs.grid1.excelExport(appendExcelExportProperties, true);
            }
        },
    },
    provide: {
        grid: [Toolbar, ExcelExport]
    }
}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/grid/excel/multiple-cs2" %}

### Excel export options in Vue Grid component

The excel export provides an option to customize mapping of the grid to excel document.

To get start quickly with Excel export Options, you can check on this video:

[Export current page](#)

The excel export provides an option to export the current page into excel. To export current page, define [exportType](#) to **CurrentPage**.

**APP.VUE**

```
<template>
  <div id="app">
    <ejs-grid ref='grid' id='Grid' :dataSource='data'
:toolbar='toolbarOptions' height='220px' :allowPaging='true'
:allowExcelExport='true' :toolbarClick='toolbarClick'>
      <e-columns>
        <e-column field='OrderID' headerText='Order ID'
textAlign='Right' width=120></e-column>
        <e-column field='CustomerID' headerText='Customer ID'
width=150></e-column>
        <e-column field='ShipCity' headerText='Ship City'
width=150></e-column>
        <e-column field='ShipName' headerText='Ship Name'
width=150></e-column>
      </e-columns>
    </ejs-grid>
  </div>
</template>
<script>
import Vue from "vue";
import { GridPlugin, Toolbar, ExcelExport, Page } from "@syncfusion/ej2-vue-
grids";
import { data } from './datasource.js';
Vue.use(GridPlugin);
export default {
  data() {
    return {
      data: data,
      toolbarOptions: ['ExcelExport']
    };
  },
  methods: {
    toolbarClick: function(args) {
      if (args.item.id === 'Grid_excelexport') { // 'Grid_excelexport' ->
Grid component id + _ + toolbar item name
        let excelExportProperties = {
          exportType: 'CurrentPage'
        };
        this.$refs.grid.excelExport(excelExportProperties);
      }
    },
  },
  provide: {
    grid: [Toolbar, ExcelExport, Page]
  }
}
</script>
<style>
  @import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
</style>
```



```
{% previewsample "page.domainurl/code-snippet/grid/excel/default-cs4" %}
```

### *Export the selected records only*

You can export the selected records data by passing it to `exportProperties.dataSource` property in the `toolbarClick` event.

In the below exporting demo, We can get the selected records using `getSelectedRecords` method and pass the selected data to `PdfExport` or `excelExport` property.

### **APP.VUE**

```
<template>
  <div id="app">
    <ejs-grid ref='grid' :dataSource='data' :toolbar='toolbarOptions'
      :allowPaging='true' :allowFiltering='true' :allowPdfExport='true'
      :allowExcelExport='true' :pageSettings='pageSettings'
      :toolbarClick='toolbarClick' :selectionSettings='selectionOption'>
      <e-columns>
        <e-column field='OrderID' headerText='Order ID'
          textAlign='Right' width=100></e-column>
        <e-column field='CustomerID' headerText='Customer ID'
          width=150></e-column>
        <e-column field='ShipCity' headerText='Ship City'
          width=150></e-column>
      </e-columns>
    </ejs-grid>
  </div>
</template>
<script>
import Vue from "vue";
import { GridPlugin, Toolbar, PdfExport, Filter, Page, ExcelExport } from
"@syncfusion/ej2-vue-grids";
import { data } from './datasource.js';
import { DataManager } from "@syncfusion/ej2-data";
Vue.use(GridPlugin);
export default {
  data: () => {
    return {
      data: data,
      toolbarOptions: ['PdfExport', 'ExcelExport'],
      pageSettings: { pageSize: 5, pageCount: 5 },
      selectionOption: { type: 'Multiple' }
    };
  },
  methods: {
    toolbarClick(args: ClickEventArgs) {
      if (args['item'].id.indexOf("pdfexport") !== -1) {
        let selectedRecords =
this.$refs.grid.getSelectedRecords();
        let exportProperties = {
          dataSource: selectedRecords
        };
        this.$refs.grid.pdfExport(exportProperties);
      }
      else if (args['item'].id.indexOf("excelexport") !== -1) {
        let selectedRecords =
this.$refs.grid.getSelectedRecords();
```

```

        let exportProperties = {
            dataSource: selectedRecords
        };
        this.$refs.grid.excelExport(exportProperties);
    },
    },
    provide: {
        grid: [Toolbar, PdfExport, Filter, Page, ExcelExport]
    },
}
</script>
<style>
    @import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/grid/how-to/export-filtered-data-cs1" %}

### Export hidden columns

The excel export provides an option to export hidden columns of grid by defining [includeHiddenColumn](#) as **true**.

### APP.VUE

```

<template>
    <div id="app">
        <ejs-grid ref='grid' id='Grid' :dataSource='data'
        :toolbar='toolbarOptions' height='270px' :allowPaging='true'
        :allowExcelExport='true' :toolbarClick='toolbarClick'>
            <e-columns>
                <e-column field='OrderID' headerText='Order ID'
                textAlign='Right' width=120></e-column>
                <e-column field='CustomerID' headerText='Customer ID'
                width=150></e-column>
                <e-column field='ShipCity' headerText='Ship City' width=150
                :visible='false'></e-column>
                <e-column field='ShipName' headerText='Ship Name'
                width=150></e-column>
            </e-columns>
        </ejs-grid>
    </div>
</template>
<script>
import Vue from "vue";
import { GridPlugin, Toolbar, ExcelExport } from "@syncfusion/ej2-vue-
grids";
import { data } from './datasource.js';
Vue.use(GridPlugin);
export default {
    data() {
        return {
            data: data,
            toolbarOptions: ['ExcelExport']
        };
    },
    methods: {

```

```

        toolbarClick: function(args) {
            if (args.item.id === 'Grid_excelexport') { // 'Grid_excelexport' ->
Grid component id + _ + toolbar item name
                let excelExportProperties = {
                    includeHiddenColumn: true
                };
                this.$refs.grid.excelExport(excelExportProperties);
            }
        },
        provide: {
            grid: [Toolbar, ExcelExport]
        }
    }
</script>
<style>
    @import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/grid/excel/default-cs5" %}

#### Show or hide columns

You can show a hidden column or hide a visible column while printing the grid using [toolbarClick](#) and [pdfExportComplete](#) events.

In the [toolbarClick](#) event, based on **args.item.id** as **Grid\_pdfexport**. We can show or hide columns by setting [column.visible](#) property to **true** or **false** respectively.

In the pdfExportComplete event, We have reversed the state back to the previous state.

In the below example, we have **CustomerID** as a hidden column in the grid. While exporting, we have changed **CustomerID** to visible column and **ShipCity** as hidden column.

#### APP.VUE

```

<template>
    <div id="app">
        <ejs-grid ref='grid' id='Grid' :dataSource='data'
:toolbar='toolbarOptions' height='272px' :allowExcelExport='true'
:excelExportComplete='excelExportComplete' :toolbarClick='toolbarClick'>
            <e-columns>
                <e-column field='OrderID' headerText='Order ID'
textAlign='Right' width=120></e-column>
                <e-column field='CustomerID' headerText='Customer ID'
:visible='false' width=150></e-column>
                <e-column field='ShipCity' headerText='Ship City'
width=150></e-column>
                <e-column field='ShipName' headerText='Ship Name'
width=150></e-column>
            </e-columns>
        </ejs-grid>
    </div>
</template>
<script>
import Vue from "vue";
import { GridPlugin, Toolbar, ExcelExport } from "@syncfusion/ej2-vue-
grids";

```

```

import { data } from './datasource.js';
Vue.use(GridPlugin);
export default {
  data() {
    return {
      data: data,
      toolbarOptions: ['ExcelExport']
    };
  },
  methods: {
    toolbarClick: function(args) {
      if (args.item.id === 'Grid_excelexport') {
        this.$refs.grid.getColumns()[1].visible = true;
        this.$refs.grid.getColumns()[3].visible = false;
        this.$refs.grid.excelExport();
      }
    },
    excelExportComplete(args) {
      this.$refs.grid.getColumns()[1].visible = true;
      this.$refs.grid.getColumns()[3].visible = true;
    }
  },
  provide: {
    grid: [Toolbar, ExcelExport]
  }
}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/grid/excel/default-cs6" %}

### Export with filter options

The excel export provides an option to export with filter option in excel by defining `enableFilter` as `true`. It requires the [allowFiltering](#) to be true.

### APP.VUE

```

<template>
  <div id="app">
    <ejs-grid ref='grid' id='Grid' :dataSource='data'
:allowFiltering='true' :toolbar='toolbarOptions' height='270px'
:allowPaging='true' :allowExcelExport='true' :toolbarClick='toolbarClick'>
      <e-columns>
        <e-column field='OrderID' headerText='Order ID'
textAlign='Right' width=120></e-column>
        <e-column field='CustomerID' headerText='Customer ID'
width=150></e-column>
        <e-column field='ShipCity' headerText='Ship City' width=150
:visible='false'></e-column>
        <e-column field='ShipName' headerText='Ship Name'
width=150></e-column>
      </e-columns>
    </ejs-grid>
  </div>

```

```

</template>
<script>
import Vue from "vue";
import { GridPlugin, Toolbar, ExcelExport, Filter } from "@syncfusion/ej2-vue-grids";
import { data } from './datasource.js';
Vue.use(GridPlugin);
export default {
  data() {
    return {
      data: data,
      toolbarOptions: ['ExcelExport']
    };
  },
  methods: {
    toolbarClick: function(args) {
      if (args.item.id === 'Grid_excelexport') { // 'Grid_excelexport' ->
        Grid component id + _ + toolbar item name
        let excelExportProperties = {
          enableFilter: true
        };
        this.$refs.grid.excelExport(excelExportProperties);
      }
    }
  },
  provide: {
    grid: [Toolbar, ExcelExport, Filter]
  }
}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/grid/excel/default-cs7" %}

### Exporting grouped records

The excel export provides outline option for grouped records which hides the detailed data for better viewing. In grid, we have provided the outline option for the exported document when the data's are grouped.

### APP.VUE

```

<template>
  <div id="app">
    <ejs-grid ref='grid' id='Grid' :dataSource='data'
      :allowGrouping='true' :groupSettings='groupOptions'
      :toolbar='toolbarOptions' height='220px' :allowPaging='true'
      :allowExcelExport='true' :toolbarClick='toolbarClick'>
      <e-columns>
        <e-column field='OrderID' headerText='Order ID'
          textAlign='Right' width=120></e-column>
        <e-column field='CustomerID' headerText='Customer ID'
          width=150></e-column>
        <e-column field='ShipCity' headerText='Ship City'
          width=150></e-column>
      </e-columns>
    </div>
  </template>

```

```

        <e-column field='ShipName' headerText='Ship Name'
width=150></e-column>
    </e-columns>
</ejs-grid>
</div>
</template>
<script>
import Vue from "vue";
import { GridPlugin, Toolbar, ExcelExport, Page, Group } from
"@syncfusion/ej2-vue-grids";
import { data } from './datasource.js';
Vue.use(GridPlugin);
export default {
  data() {
    return {
      data: data,
      toolbarOptions: ['ExcelExport'],
      groupOptions: { columns: ['CustomerID', 'ShipCity'] }
    };
  },
  methods: {
    toolbarClick: function(args) {
      if (args.item.id === 'Grid_excelexport') { // 'Grid_excelexport' ->
Grid component id + _ + toolbar item name
        this.$refs.grid.excelExport();
      }
    }
  },
  provide: {
    grid: [Toolbar, ExcelExport, Page, Group]
  }
}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/grid/excel/default-cs8" %}

#### Define file name

You can assign the file name for the exported document by defining **fileName** property in [excelExportProperties](#).

#### APP.VUE

```

<template>
  <div id="app">
    <ejs-grid ref='grid' id='Grid' :dataSource='data'
:toolbar='toolbarOptions' height='270px' :allowPaging='true'
:allowExcelExport='true' :toolbarClick='toolbarClick'>
      <e-columns>
        <e-column field='OrderID' headerText='Order ID'
textAlign='Right' width=120></e-column>
        <e-column field='CustomerID' headerText='Customer ID'
width=150></e-column>
      </e-columns>
    </ejs-grid>
  </div>
</template>

```

```

        <e-column field='ShipCity' headerText='Ship City'
width=150></e-column>
        <e-column field='ShipName' headerText='Ship Name'
width=150></e-column>
    </e-columns>
</ejs-grid>
</div>
</template>
<script>
import Vue from "vue";
import { GridPlugin, Toolbar, ExcelExport } from "@syncfusion/ej2-vue-
grids";
import { data } from './datasource.js';
Vue.use(GridPlugin);
export default {
  data() {
    return {
      data: data,
      toolbarOptions: ['ExcelExport']
    };
  },
  methods: {
    toolbarClick: function(args) {
      if (args.item.id === 'Grid_excelexport') { // 'Grid_excelexport' ->
Grid component id + _ + toolbar item name
        let excelExportProperties = {
          fileName:"new.xlsx"
        };
        this.$refs.grid.excelExport(excelExportProperties);
      }
    },
  },
  provide: {
    grid: [Toolbar, ExcelExport]
  }
}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/grid/excel/default-cs9" %}

### [Export the master detail grid](#)

It is possible to export the master-detail grid on the same Excel sheet using the `ExcelExportProperties` class in the grid.

To export the master-detail grid on the same sheet in the following sample, you need to set the `multipleExport.type` to `AppendToSheet` in the `exportProperties`. A promise object is created by exporting the master grid first, and then the detail grid is exported after the master grid has been successfully exported.

### **APP.VUE**

```

<template>
  <div id="app">

```

```

    <ejs-grid ref="MasterGrid" :dataSource="data" id="MasterGrid"
    :selectedRowIndex="1" :toolbar="toolbar" :rowSelected="rowSelected"
    :toolbarClick="toolbarClick" allowExcelExport="true">
        <e-columns>
            <e-column field="ContactName" headerText="Customer Name"
width="150"></e-column>
            <e-column field="CompanyName" headerText="Company Name"
width="150"></e-column>
            <e-column field="Address" headerText="Address" width="150"></e-
column>
            <e-column field="Country" headerText="Country" width="130"></e-
column>
        </e-columns>
    </ejs-grid>
    <div class="e-statustext">Showing orders of Customer: <b
id="key"></b></div>
    <ejs-grid ref="grid" :allowSelection="false" allowExcelExport="true">
        <e-columns>
            <e-column field="OrderID" headerText="Order ID" width="100"
textAlign="Right"></e-column>
            <e-column field="Freight" headerText="Freight" format="C2"
width="100" type="number"></e-column>
            <e-column field="ShipName" headerText="Ship Name"
width="200"></e-column>
            <e-column field="ShipCountry" headerText="Ship Country"
width="150"></e-column>
            <e-column field="ShipAddress" headerText="Ship Address"
width="200"></e-column>
        </e-columns>
    </ejs-grid>
</div>
</template>
<script>
import Vue from "vue";
import { GridPlugin, Toolbar, ExcelExport } from "@syncfusion/ej2-vue-
grids";
import { customerData, data } from "../datasource.js";
Vue.use(GridPlugin);
export default {
    data() {
        var names = ["AROUT", "BERGS", "BLONP", "CHOPS", "ERNSH"];
        return {
            toolbar: ["ExcelExport"],
            data: customerData.filter(function (e) {
                return names.indexOf(e.CustomerID) !== -1;
            })
        };
    },
    methods: {
        rowSelected: function (args) {
            let selectedRecord = args.data;
            this.$refs.grid.ej2Instances.dataSource = data.filter((record)
=> record.CustomerName === selectedRecord.ContactName).slice(0, 5);
            document.getElementById("key").innerHTML =
selectedRecord.ContactName;
        },
        toolbarClick: function (args) {

```



```

        const appendExcelExportProperties = {
            multipleExport: { type: "AppendToSheet", blankRows: 2 },
        };
        const firstGridExport =
this.$refs.MasterGrid.ej2Instances.excelExport(
            appendExcelExportProperties, true);
        firstGridExport.then((fData) => {

this.$refs.grid.ej2Instances.excelExport(appendExcelExportProperties, false,
fData);

        });
    },
    provide: {
        grid: [Toolbar, ExcelExport],
    }
};
</script>
<style>
    @import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/grid/print/mastergrid-cs1" %}

#### Export grid as blob

The Grid offers an option to export the data as a Blob instead of downloading it as a file in the browser. To export the grid as a Blob, set the `isBlob` parameter to **true** in the [excelExport](#) method. The grid returns the promise of a blob in the [excelExportComplete](#) event.

The following example demonstrates how to obtain the blob data of the exported grid by executing the promise in the [excelExportComplete](#) event.

#### APP.VUE

```

<template>
    <div id="app">
        <ejs-grid ref='grid' id='Grid' :dataSource='data'
:toolbar='toolbarOptions' height='220px' :allowPaging='true'
:allowExcelExport='true' :toolbarClick='toolbarClick'
:excelExportComplete="excelExportComplete">
            <e-columns>
                <e-column field='OrderID' headerText='Order ID'
textAlign='Right' width=120></e-column>
                <e-column field='CustomerID' headerText='Customer ID'
width=150></e-column>
                <e-column field='ShipCity' headerText='Ship City'
width=150></e-column>
                <e-column field='ShipName' headerText='Ship Name'
width=150></e-column>
            </e-columns>
        </ejs-grid>
    </div>
</template>
<script>
import Vue from "vue";

```

```

import { GridPlugin, Toolbar, ExcelExport, Page } from "@syncfusion/ej2-vue-grids";
import { data } from './datasource.js';
Vue.use(GridPlugin);
export default {
  data() {
    return {
      data: data,
      toolbarOptions: ['ExcelExport', 'CsvExport']
    };
  },
  methods: {
    toolbarClick(args) {
      if (args.item.id === 'Grid_excelexport') {
        // pass fourth parameter as true to get the blob data of exported
        grid
        this.$refs.grid.excelExport(null, null, null, true);
      }
      if (args.item.id === 'Grid_csvexport') {
        // pass fourth parameter as true to get the blob data of exported
        grid
        this.$refs.grid.csvExport(null, null, null, true);
      }
    },
    excelExportComplete(args) {
      // execute the promise to get the blob data
      args.promise.then((e) => {
        console.log(e.blobData);
      });
    },
  },
  provide: {
    grid: [Toolbar, ExcelExport, Page]
  }
}
</script>
<style>
  @import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/grid/excel/export-grid-as-blob-cs1" %}

Excel cell style customization in Vue Grid component

#### Conditional cell formatting

Grid cells in the exported Excel can be customized or formatted using [excelQueryCellInfo](#) event. In this event, we can format the grid cells of exported PDF document based on the column cell value.

In the below sample, we have set the background color for **Freight** column in the exported excel by **args.cell** and **backColor** property.

#### APP.VUE

```

<template>
  <div id="app">
    <ejs-grid ref='grid' id='Grid' :dataSource='data'
      :toolbar='toolbarOptions' height='272px' :allowExcelExport='true'

```

```

:queryCellInfo='queryCellInfo' :excelQueryCellInfo='excelQueryCellInfo'
:toolbarClick='toolbarClick'>
    <e-columns>
        <e-column field='OrderID' headerText='Order ID'
textAlign='Right' width=120></e-column>
        <e-column field='CustomerID' headerText='Customer ID'
width=150></e-column>
        <e-column field='Freight' headerText='Freight'
width=150></e-column>
        <e-column field='ShipName' headerText='Ship Name'
width=150></e-column>
    </e-columns>
</ejs-grid>
</div>
</template>
<script>
import Vue from "vue";
import { GridPlugin, Toolbar, ExcelExport } from "@syncfusion/ej2-vue-
grids";
import { data } from './datasource.js';
Vue.use(GridPlugin);
export default {
    data() {
        return {
            data: data,
            toolbarOptions: ['ExcelExport']
        };
    },
    methods: {
        toolbarClick: function(args) {
            if (args.item.id === 'Grid_excelexport') {
                this.$refs.grid.excelExport();
            }
        },
        excelQueryCellInfo(args) {
            if(args.column.field == 'Freight')
                if(args.value < 30) {
                    args.style = {backColor: '#99ffcc'};
                }
                else if(args.value < 60) {
                    args.style = {backColor: '#ffffb3'};
                }
                else {
                    args.style = {backColor: '#ff704d'};
                }
        },
        queryCellInfo(args) {
            if(args.column.field == 'Freight'){
                if(args.data['Freight'] < 30) {
                    args.cell.bgColor = '#99ffcc';
                }
                else if(args.data['Freight'] < 60) {
                    args.cell.bgColor = '#ffffb3';
                }
                else {
                    args.cell.bgColor = '#ff704d';
                }
            }
        }
    }
}

```

```

    }
  },
  provide: {
    grid: [Toolbar, ExcelExport]
  }
}
</script>
<style>
  @import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/grid/excel/default-cs2" %}

### Theme

The excel export provides an option to include theme for exported excel document.

To apply theme in exported Excel, define the **theme** in [exportProperties](#).

### APP.VUE

```

<template>
  <div id="app">
    <ejs-grid ref='grid' id='Grid' :dataSource='data'
      :toolbar='toolbarOptions' height='270px' :allowPaging='true'
      :allowExcelExport='true' :toolbarClick='toolbarClick'>
      <e-columns>
        <e-column field='OrderID' headerText='Order ID'
          textAlign='Right' width=120></e-column>
        <e-column field='CustomerID' headerText='Customer ID'
          width=150></e-column>
        <e-column field='ShipCity' headerText='Ship City'
          width=150></e-column>
        <e-column field='ShipName' headerText='Ship Name'
          width=150></e-column>
      </e-columns>
    </ejs-grid>
  </div>
</template>
<script>
import Vue from "vue";
import { GridPlugin, Toolbar, ExcelExport } from "@syncfusion/ej2-vue-grids";
import { data } from './datasource.js';
Vue.use(GridPlugin);
export default {
  data() {
    return {
      data: data,
      toolbarOptions: ['ExcelExport']
    };
  },
  methods: {
    toolbarClick: function(args) {
      if (args.item.id === 'Grid_excelexport') { // 'Grid_excelexport' ->
        Grid component id + _ + toolbar item name
        let excelExportProperties = {

```

```

        theme:
        {
            header: { fontName: 'Segoe UI', fontColor: '#666666'
        },
            record: { fontName: 'Segoe UI', fontColor: '#666666'
        },
            caption: { fontName: 'Segoe UI', fontColor:
        '#666666' }
        }
    };
    this.$refs.grid.excelExport(excelExportProperties);
}
},
provide: {
    grid: [Toolbar, ExcelExport]
}
}
</script>
<style>
    @import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/grid/excel/default-cs3" %}

By default, material theme is applied to exported excel document.

*Rotate a header text to a certain degree in the exported grid*

The DataGrid has support to customize the column header styles such as changing text orientation, the font color, and so on in the exported Excel file. To achieve this requirement, use the [excelHeaderQueryCellInfo](#) event of the Grid.

The `excelHeaderQueryCellInfo` will be triggered when creating a column header for the excel document to be exported. Customize the column header in this event.

In the following demo, using the `rotation` property of the style argument in the `excelHeaderQueryCellInfo` event, you can rotate the header text of the column header in the excel exported document.

#### APP.VUE

```

<template>
    <div id="app">
        <ejs-dropdownlist ref='dropdown' id='dropdownlist'
        :dataSource='degree' placeholder='Select a degree'></ejs-dropdownlist>
        <ejs-grid ref='grid' id='Grid' :dataSource='data'
        :created='setHeaderHeight' height='240px' :toolbar='toolbarOptions'
        :allowExcelExport='true' :excelQueryCellInfo='excelQueryCellInfo'
        :toolbarClick='toolbarClick'
        :excelHeaderQueryCellInfo='excelHeaderQueryCellInfo'>
            <e-columns>
                <e-column field='OrderID' headerText='Order ID'
                textAlign='Right' width=100></e-column>
                <e-column field='CustomerID' headerText='Customer ID'
                width=120 :customAttributes='customAttributes'></e-column>
            </e-columns>
        </ejs-grid>
    </div>
</template>

```

```

        <e-column field='Freight' headerText='Freight'
textAlign='Center' format='C2' width=80></e-column>
        <e-column field='ShipCity' headerText='Ship City'
width=100></e-column>
    </e-columns>
</ejs-grid>
</div>
</template>
<script>
import Vue from "vue";
import { GridPlugin, Toolbar, ExcelExport } from "@syncfusion/ej2-vue-
grids";
import { DropDownListPlugin } from "@syncfusion/ej2-vue-dropdowns";
import { data } from './datasource.js';
Vue.use(GridPlugin);
Vue.use(DropDownListPlugin);
export default {
    data: () => {
        return {
            data: data,
            customAttributes : {class : 'orientationcss'},
            toolbarOptions: ['ExcelExport'],
            degree : [90, 180, 45, 135]
        };
    },
    methods: {
        toolbarClick: function(args) {
            if (args.item.id === 'Grid_excelexport') {
                this.$refs.grid.excelExport();
            }
        },
        setHeaderHeight: function(args) {
            let textWidth = document.querySelector(".orientationcss >
div").scrollWidth; //Obtain the width of the headerText content.
            let headerCell = document.querySelectorAll(".e-headercell");
            for (let i = 0; i < headerCell.length; i++) {
                (headerCell.item(i)).style.height = textWidth + 'px'; //Assign the
obtained textWidth as the height of the headerCell.
            }
        },
        excelQueryCellInfo(args) {
            if (args.column.field === 'Freight') {
                if (args.value < 30) {
                    args.style = { backColor: '#99ffcc' };
                }
                else if (args.value < 60) {
                    args.style = { backColor: '#ffffb3' };
                }
                else {
                    args.style = { backColor: '#ff704d' };
                }
            }
        },
        excelHeaderQueryCellInfo(args) {
            let textWidth = document.querySelector(".orientationcss >
div").scrollWidth;
            if (args.gridCell.column.field === 'Freight') {

```

```

        args.style = { backgroundColor: '#99ffcc', valign: 'Bottom' };
    }
    else {
        args.style = { valign: 'Center', rotation:
this.$refs.dropdown.value };
    }
    args.cell.cellHeight = textWidth;
},
},
provide: {
    grid: [Toolbar, ExcelExport]
}
}
</script>
<style>
    @import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
    @import "../node_modules/@syncfusion/ej2-vue-
dropdowns/styles/material.css";
    .orientationcss .e-headercelldiv {
        transform: rotate(90deg);
        padding-top: 5px;
    }
</style>

```

{% previewsample "page.domainurl/code-snippet/grid/how-to/default-cs9" %}

### Adding header and footer in Vue Grid component

The excel export provides an option to include header and footer content for exported excel document.

### APP.VUE

```

<template>
    <div id="app">
        <ejs-grid ref='grid' id='Grid' :dataSource='data'
:toolbar='toolbarOptions' height='270px' :allowPaging='true'
:allowExcelExport='true' :toolbarClick='toolbarClick'>
            <e-columns>
                <e-column field='OrderID' headerText='Order ID'
textAlign='Right' width=120></e-column>
                <e-column field='CustomerID' headerText='Customer ID'
width=150></e-column>
                <e-column field='ShipCity' headerText='Ship City'
width=150></e-column>
                <e-column field='ShipName' headerText='Ship Name'
width=150></e-column>
            </e-columns>
        </ejs-grid>
    </div>
</template>
<script>
import Vue from "vue";
import { GridPlugin, Toolbar, ExcelExport } from "@syncfusion/ej2-vue-
grids";
import { data } from './datasource.js';
Vue.use(GridPlugin);
export default {

```

```

data() {
  return {
    data: data,
    toolbarOptions: ['ExcelExport']
  };
},
methods: {
  toolbarClick: function(args) {
    if (args.item.id === 'Grid_excelexport') { // 'Grid_excelexport' ->
Grid component id + _ + toolbar item name
      let excelExportProperties = {
        header: {
          headerRows: 7,
          rows: [
            { cells: [{ colSpan: 4, value: "Northwind Traders",
style: { fontColor: '#C67878', fontSize: 20, hAlign: 'Center', bold: true, }
}} ],
            { cells: [{ colSpan: 4, value: "2501 Aerial Center
Parkway", style: { fontColor: '#C67878', fontSize: 15, hAlign: 'Center',
bold: true, } } ] },
            { cells: [{ colSpan: 4, value: "Suite 200
Morrisville, NC 27560 USA", style: { fontColor: '#C67878', fontSize: 15,
hAlign: 'Center', bold: true, } } ] },
            { cells: [{ colSpan: 4, value: "Tel +1 888.936.8638
Fax +1 919.573.0306", style: { fontColor: '#C67878', fontSize: 15, hAlign:
'Center', bold: true, } } ] },
            { cells: [{ colSpan: 4, hyperlink: { target:
'https://www.northwind.com/', displayText: 'www.northwind.com' }, style: {
hAlign: 'Center' } } ] },
            { cells: [{ colSpan: 4, hyperlink: { target:
'mailto:support@northwind.com' }, style: { hAlign: 'Center' } } ] },
          ]
        },
        footer: {
          footerRows: 4,
          rows: [
            { cells: [{ colSpan: 4, value: "Thank you for your
business!", style: { hAlign: 'Center', bold: true } } ] },
            { cells: [{ colSpan: 4, value: "!Visit Again!",
style: { hAlign: 'Center', bold: true } } ] }
          ]
        }
      };
      this.$refs.grid.excelExport(excelExportProperties);
    }
  },
  provide: {
    grid: [Toolbar, ExcelExport]
  }
}
</script>
<style>
  @import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
</style>

```



```
{% previewsample "page.domainurl/code-snippet/grid/excel/default-cs1" %}
```

### Exporting hierarchy grid in Vue Grid component

The grid have an option to export the hierarchy grid to excel document. By default, grid will exports the visible child grids in expanded state. you can change the exporting option by using the **ExcelExportProperties.hierarchyExportMode** property. The available options are,

Mode	Behavior
Expanded	Exports the visible child grids in expanded state and remaining child grid in collapsed state when args.isChild property is set to true in <a href="#">beforeExcelExport</a> event.
All	Exports the all the child grids in expanded state.
None	Exports all child grids in collapsed state when args.isChild property is set to true in <a href="#">beforeExcelExport</a> event.

### APP.VUE

```
<template>
  <div id="app">
    <ejs-grid ref='grid' id='Grid' :dataSource='parentData'
      :childGrid='childGrid' :toolbar='["ExcelExport"]'
      :toolbarClick='toolbarClick' :beforeExcelExport='beforeExcelExport'
      :allowExcelExport='true'>
      <e-columns>
        <e-column field='EmployeeID' headerText='Employee ID'
          textAlign='Right' width=120></e-column>
        <e-column field='FirstName' headerText='First Name'
          width=150></e-column>
        <e-column field='LastName' headerText='Last Name'
          width=150></e-column>
        <e-column field='City' headerText='City' width=150></e-
column>
      </e-columns>
    </ejs-grid>
  </div>
</template>
<script>
import Vue from "vue";
import { GridPlugin, DetailRow, Toolbar, ExcelExport, ExcelExportProperties
} from "@syncfusion/ej2-vue-grids";
import { data, employeeData } from './datasource.js';
import { extend } from '@syncfusion/ej2-base';
Vue.use(GridPlugin);
export default {
  data() {
    return {
      parentData: employeeData,
      childGrid: {
        dataSource: data,
        queryString: 'EmployeeID',
        columns: [
          { field: 'OrderID', headerText: 'Order ID', textAlign: 'Right',
width: 120 },
          { field: 'CustomerID', headerText: 'Customer ID', width: 150 },
```

```

        { field: 'ShipCity', headerText: 'Ship City', width: 150 },
        { field: 'ShipName', headerText: 'Ship Name', width: 150 }
    ]
}
},
methods: {
    toolbarClick: function( args) {
        if (args['item'].id === 'Grid_excelexport') {
            let exportProperties = {
                hierarchyExportMode: "Expanded"
            };
            this.$refs.grid.excelExport(exportProperties);
        }
    },
    beforeExcelExport: function(args) {
        args.isChild = true;
    }
},
provide: {
    grid: [DetailRow, Toolbar, ExcelExport]
}
}
</script>
<style>
    @import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/grid/hierarchy-grid/default-cs1" %}

### Limitations

- Microsoft Excel permits up to seven nested levels in outlines. So that in the grid we can able to provide only up to seven nested levels and if it exceeds more than seven levels then the document will be exported without outline option.

Please refer the [Microsoft Limitation](#)

### Exporting grid with templates in Vue Grid control

The grid offers the option to export the column, detail, and caption templates to an Excel document. The template contains images, hyperlinks, and customized text.

#### Exporting with column template

The Excel export functionality allows you to export Grid columns that include images, hyperlinks, and custom text to an Excel document.

In the following sample, the hyperlinks and images are exported to Excel using [hyperlink](#) and [image](#) properties in the [excelQueryCellInfo](https://ej2.syncfusion.com/vue/documentation/api/grid/#excelquerycellinfo) event.

Excel Export supports base64 string to export the images.

### APP.VUE

```

<template>
  <ejs-grid id="ColumnTemplateGrid" ref="grid" :dataSource="data"
:allowExcelExport="true"
  :toolbar="toolbar" :toolbarClick="toolbarClick"
:excelQueryCellInfo="excelQueryCellInfo" height=315>
    <e-columns>
      <e-column headerText="Employee Image" textAlign="Center"
:template="imageTemplate" width="150"></e-column>
      <e-column field="EmployeeID" headerText="Employee ID"
width="125"></e-column>
      <e-column field="FirstName" headerText="Name" width="120"></e-
column>
      <e-column headerText="Email ID" :template="mailTemplate"
width="170"></e-column>
    </e-columns>
  </ejs-grid>
</template>
<script>
import Vue from 'vue';
import { GridPlugin, ExcelExport, Toolbar } from '@syncfusion/ej2-vue-
grids';
import { employeeData } from './datasource.js';
Vue.use(GridPlugin);
export default {
  data: () => {
    return {
      data: employeeData,
      toolbar: ['ExcelExport'],
      imageTemplate: function () {
        return { template : Vue.component('imageTemplate',{
          template: `<div class="image">
            
            </div>`,
          data: function() {
            return {
              data: {}
            }
          },
        })},
      },
      mailTemplate: function () {
        return { template : Vue.component('mailTemplate',{
          template: `<div class="link">
            <a
:href="'mailto:'+data.EmailID">{{data.EmailID}}</a></div>
            </div>`,
          data: function() {
            return {
              data: {}
            }
          },
        })},
      },
    };
  },
  methods: {





```

```

        toolbarClick: function (args) {
            if (args.item.id === 'ColumnTemplateGrid_excelexport') {
                this.$refs.grid.excelExport();
            }
        },
        excelQueryCellInfo: function (args) {
            if (args.column.headerText === 'Employee Image') {
                args.image = {
                    base64: args.data.EmployeeImage,
                    height: 70,
                    width: 70,
                };
            }
            if (args.column.headerText === 'Email ID') {
                args.hyperLink = {
                    target: 'mailto:' + args.data.EmailID,
                    displayText: args.data.EmailID,
                };
            }
        }
    },
    provide: {
        grid: [ExcelExport, Toolbar],
    },
}
</script>
<style>
    @import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
    .image img {
        height: 55px;
        width: 55px;
        border-radius: 50px;
        box-shadow: inset 0 0 1px #e0e0e0, inset 0 0 14px rgba(0,0,0,0.2);
    }
</style>

```

{% previewsample "page.domainurl/code-snippet/grid/excel/column-template-export-cs1" %}

Excel Export		
Employee Image	Name	Email ID
	Nancy	nancy@domain.com
	Andrew	andrew@domain.com
	Janet	janet@domain.com
	Margaret	margaret@domain.com

#### Exporting with detail template

By default, the grid will export the parent grid with expanded detail rows alone. Change the exporting option by using the `ExcelExportProperties.hierarchyExportMode` property. The available options are:

- | Mode | Behavior |
- |-----|-----|
- | Expanded | Exports the parent grid with expanded detail rows. |
- | All | Exports the parent grid with all the detail rows. |
- | None | Exports the parent grid alone. |

The detail rows in the exported Excel can be customized or formatted using the [exportDetailTemplate](#) event. In this event, the detail rows of the Excel document are formatted in accordance with their parent row details.

In the following sample, the detail row content is formatted by specifying the [columnHeader](#) and [rows](#) properties using its [parentRow](#) details. This allows for the creation of detail rows in the Excel document. Additionally, custom styles can be applied to specific cells using the [style](#) property.

When using [rowSpan](#), it is essential to provide the cell's [index](#) for proper functionality.

#### APP.VUE

```
<template>
```

```

<ejs-grid id="DetailTemplateGrid" ref="grid" :dataSource="data"
:detailTemplate="detailTemplate" :toolbar="toolbar"
:toolbarClick="toolbarClick" :exportDetailTemplate="exportDetailTemplate"
height=315 :allowExcelExport="true">
  <e-columns>
    <e-column field="Category" headerText="Category" width="140"
textAlign="Right"></e-column>
    <e-column field="ProductID" headerText="Product ID" width="140"></e-
column>
    <e-column field="Status" headerText="Status" width="200"></e-column>
  </e-columns>
</ejs-grid>
</template>
<script>
import Vue from 'vue';
import { GridPlugin, DetailRow, ExcelExport, Toolbar } from
'@syncfusion/ej2-vue-grids';
import { employeeData } from './datasource.js';
Vue.use(GridPlugin);
export default {
  data: () => {
    return {
      data: employeeData,
      toolbar: ['ExcelExport'],
      detailTemplate: function () {
        return {
          template: Vue.component('detailTemplate', {
            template: `<table class="detailtable" width="100%">
<colgroup>
  <col width="40%" />
  <col width="60%" />
</colgroup>
<thead>
  <tr>
    <th colspan="2" style="font-weight: 500;text-align:
center;background-color: #ADD8E6;">
      Product Details
    </th>
  </tr>
</thead>
<tbody>
  <tr>
    <td rowspan="4" style="text-align: center;">
      
    </td>
    <td>
      <span style="font-weight: 500;color:
#0a76ff;">Offers: {{data.Offers}} </span>
    </td>
  </tr>
  <tr>
    <td>
      <span>Available: {{data.Available}} </span>
    </td>
  </tr>
  <tr>
    <td>
      <span>Available: {{data.Available}} </span>
    </td>
  </tr>
  <tr>
    <td>
      <span>Available: {{data.Available}} </span>
    </td>
  </tr>
</tbody>
</table>

```

```

        <td>
            <span class="link">
                Contact: <a
:href="'mailto:' + data.Contact">{{data.Contact}}</a>
            </span>
        </td>
    </tr>
</tr>

```

```

    },
    exportDetailTemplate: function(args) {
        args.value = {
            columnHeader: [
                {
                    cells: [{
                        index: 0, colSpan: 2, value: 'Product Details',
                        style: { backgroundColor: '#ADD8E6', excelHAlign: 'Center',
bold: true }
                    }]
                }
            ],
            rows: [
                {
                    cells: [
                        {
                            index: 0, rowspan: 4, image: {
                                base64: args.parentRow.data['ProductImg'],
                                height: 80, width: 100
                            }
                        },
                        {
                            index: 1, value: "Offers: " +
args.parentRow.data['Offers'],
                            style: { bold: true, fontColor: '#0a76ff' }
                        },
                    ]
                },
                {
                    cells: [
                        {
                            index: 1, value: 'Available: ' +
args.parentRow.data['Available']
                        }
                    ]
                },
                {
                    cells: [
                        {
                            index: 1, value: 'Contact: ',
                            hyperLink: {
                                target: 'mailto: ' +
args.parentRow.data['Contact'],
                                displayText: args.parentRow.data['Contact']
                            }
                        }
                    ]
                },
                {
                    cells: [
                        {
                            index: 1, value: 'Ratings: ' +
args.parentRow.data['Ratings'],
                            style: { bold: true, fontColor: '#0a76ff' }
                        }
                    ]
                },
                {
                    cells: [

```



```

        {
            index: 0, value: args.parentRow.data['productDesc'],
            style: { excelHAlign: 'Center' }
        },
        { index: 1, value: args.parentRow.data['ReturnPolicy'] }
    ]
},
{
    cells: [
        {
            index: 0, value: args.parentRow.data['Cost'],
            style: { excelHAlign: 'Center', bold: true }
        },
        { index: 1, value: args.parentRow.data['Cancellation'] }
    ]
},
{
    cells: [
        {
            index: 0, value: args.parentRow.data['Status'],
            style: {
                bold: true, fontColor:
args.parentRow.data['Status'] === 'Available' ? '#00FF00' : '#FF0000',
                excelHAlign: 'Center'
            }
        },
        {
            index: 1, value: args.parentRow.data['Delivery'],
            style: { bold: true, fontColor: '#0a76ff' }
        }
    ]
}
    ],
};
}
},
provide: {
    grid: [DetailRow, ExcelExport, Toolbar],
},
}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
.detailtable td {
    font-size: 13px;
    padding: 4px;
    max-width: 0;
    overflow: hidden;
    text-overflow: ellipsis;
    white-space: nowrap;
}
.photo {
    width: 100px;
    height: 100px;
    border-radius: 50px;
    box-shadow: inset 0 0 1px #e0e0e0, inset 0 0 14px rgba(0,0,0,0.2);
}

```

```

.Unavailable {
    color: #FF0000;
}
.Available {
    color: #00FF00;
}
@media screen and (max-width: 800px) and (min-width: 320px) {
    .photo {
        width: 70px;
        height: 70px;
    }
}
</style>

```

{% previewsample "page.domainurl/code-snippet/grid/excel/detail-template-export-cs1" %}

Excel Export			
	Category	Product ID	Status
▶	Suits/Slim	EJ-SU-01	Available
▶	Suits/Classic	EJ-SU-02	Available
▶	Suits/Formal	EJ-SU-03	Available
▶	Phants/Slim	EJ-PH-01	Available
▶	Phants/Classic	EJ-PH-02	Available
▶	Shirts/Slim	EJ-SH-01	Available
▶	Shirts/Formal	EJ-SH-02	Available

#### Exporting with caption template

The Excel export feature enables exporting of Grid with a caption template to an Excel document.

In the following sample, the customized caption text is exported to Excel using [captionText](#) property in the [exportGroupCaption](#) event.

#### APP.VUE

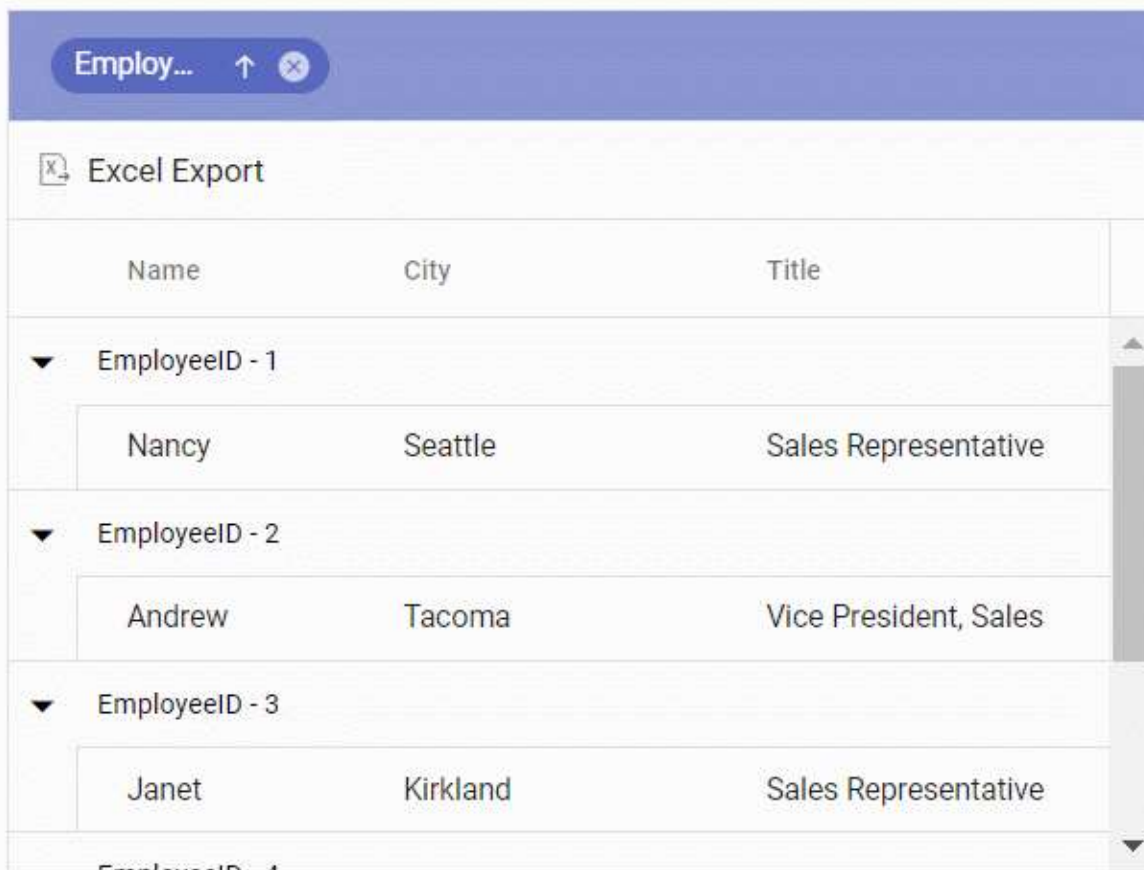
```
<template>
```

```

<ejs-grid id="CaptionTemplateGrid" ref="grid" :dataSource="data"
:allowGrouping="true" :groupSettings="groupOptions"
:allowExcelExport="true" :toolbar="toolbar" :toolbarClick="toolbarClick"
:exportGroupCaption="exportGroupCaption" height=315>
  <e-columns>
    <e-column field="EmployeeID" headerText="Employee ID"
width="120"></e-column>
    <e-column field="FirstName" headerText="Name" width="120"></e-
column>
    <e-column field="City" headerText="City"></e-column>
    <e-column field="Title" headerText="Title" width="170"></e-
column>
  </e-columns>
</ejs-grid>
</template>
<script>
import Vue from 'vue';
import { GridPlugin, Group, ExcelExport, Toolbar } from '@syncfusion/ej2-
vue-grids';
import { employeeData } from './datasource.js';
Vue.use(GridPlugin);
export default {
  data: () => {
    return {
      data: employeeData,
      toolbar: ['ExcelExport'],
      groupOptions: {
        columns: ['EmployeeID'],
        captionTemplate: '${field} - ${key}'
      }
    };
  },
  methods: {
    toolbarClick: function (args) {
      if (args.item.id === 'CaptionTemplateGrid_excelexport') {
        this.$refs.grid.excelExport();
      }
    },
    exportGroupCaption: function(args) {
      args.captionText = args.data.field + ' - ' + args.data.key;
    }
  },
  provide: {
    grid: [Group, ExcelExport, Toolbar],
  },
}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/grid/excel/caption-template-export-cs1" %}



Name	City	Title
EmployeeID - 1		
Nancy	Seattle	Sales Representative
EmployeeID - 2		
Andrew	Tacoma	Vice President, Sales
EmployeeID - 3		
Janet	Kirkland	Sales Representative

### Exporting grid in server in Vue Grid component

The Grid have an option to export the data to Excel in server side using Grid server export library.

#### Server dependencies

The Server side export functionality is shipped in the Syncfusion.EJ2.GridExport package, which is available in Essential Studio and [nuget.org](https://www.nuget.org/packages/Syncfusion.EJ2.GridExport). The following list of dependencies is required for Grid server side Excel exporting action.

- Syncfusion.EJ2
- Syncfusion.EJ2.GridExport

#### Server configuration

The following code snippet shows server configuration using ASP.NET Core Controller Action.

To Export the Grid in server side, You need to call the [serverExcelExport](#) method for passing the Grid properties to server exporting action.

```
`ts
public ActionResult ExcelExport([FromForm] string gridModel)
{
    GridExcelExport exp = new GridExcelExport();
    Grid gridProperty = ConvertGridObject(gridModel);
```

```

return exp.ExcelExport<OrdersDetails>(gridProperty, OrdersDetails.GetAllRecords());
}

private Grid ConvertGridObject(string gridProperty)
{
    Grid GridModel = (Grid)Newtonsoft.Json.JsonConvert.DeserializeObject(gridProperty, typeof(Grid));
    GridColumnModel cols =
    (GridColumnModel)Newtonsoft.Json.JsonConvert.DeserializeObject(gridProperty,
    typeof(GridColumnModel));
    GridModel.Columns = cols.columns;
    return GridModel;
}
`
`

<template>
<div id="app">
<ejs-grid ref='grid' id='Grid' :dataSource='data' :toolbar='toolbarOptions' height='272px'
:toolbarClick='toolbarClick'>
<e-columns>
<e-column field='OrderID' headerText='Order ID' textAlign='Right' width=120></e-column>
<e-column field='CustomerID' headerText='Customer ID' width=150></e-column>
<e-column field='ShipCity' headerText='Ship City' width=150></e-column>
<e-column field='ShipName' headerText='Ship Name' width=150></e-column>
</e-columns>
</ejs-grid>
</div>
</template>
<script>
import Vue from "vue";
import { GridPlugin, Toolbar } from "@syncfusion/ej2-vue-grids";
import { data } from './datasource.js';
Vue.use(GridPlugin);
export default {
data() {
return {

```

```

data: data,
toolbarOptions: ['ExcelExport']
};
},
methods: {
toolbarClick: function(args) {
if (args.item.id === 'Gridexcelexport') { // 'Gridexcelexport' -> Grid component id + _ + toolbar item
name
this.$refs.grid.serverExcelExport('Home/ExcelExport');
}
}
},
provide: {
grid: [Toolbar]
}
}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
</style>
`

```

Note: Refer to the GitHub sample for quick implementation and testing from [here](#).

#### *CSV export in server side*

You can export the Grid to CSV format by using the [serverCsvExport](#) method which will pass the Grid properties to server.

In the below demo, we have invoked the above method inside the [toolbarClick](#) event. In server side, we have deserialized the Grid properties and passed to the `CsvExport` method which will export the properties to CSV format.

```

`ts
public ActionResult CsvGridExport([FromForm] string gridModel)
{
GridExcelExport exp = new GridExcelExport();
Grid gridProperty = ConvertGridObject(gridModel);
return exp.CsvExport<OrdersDetails>(gridProperty, OrdersDetails.GetAllRecords());
}

```

```

private Grid ConvertGridObject(string gridProperty)
{
    Grid GridModel = (Grid)Newtonsoft.Json.JsonConvert.DeserializeObject(gridProperty, typeof(Grid));
    GridColumnModel cols =
    (GridColumnModel)Newtonsoft.Json.JsonConvert.DeserializeObject(gridProperty,
    typeof(GridColumnModel));
    GridModel.Columns = cols.columns;
    return GridModel;
}
、
、

<template>
<div id="app">
<ejs-grid ref='grid' id='Grid' :dataSource='data' :toolbar='toolbarOptions' height='272px'
:toolbarClick='toolbarClick'>
<e-columns>
<e-column field='OrderID' headerText='Order ID' textAlign='Right' width=120></e-column>
<e-column field='CustomerID' headerText='Customer ID' width=150></e-column>
<e-column field='ShipCity' headerText='Ship City' width=150></e-column>
<e-column field='ShipName' headerText='Ship Name' width=150></e-column>
</e-columns>
</ejs-grid>
</div>
</template>
<script>
import Vue from "vue";
import { GridPlugin, Toolbar } from "@syncfusion/ej2-vue-grids";
import { data } from './datasource.js';
Vue.use(GridPlugin);
export default {
data() {
return {
data: data,
toolbarOptions: ['CsvExport']

```

```

};
},
methods: {
  toolbarClick: function(args) {
    if (args.item.id === 'Gridcsvexport') { // 'Gridcsvexport' -> Grid component id + _ + toolbar item name
      this.$refs.grid.serverCsvExport('Home/CsvGridExport');
    }
  }
},
provide: {
  grid: [Toolbar]
}
}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
</style>
`

```

#### *Export grid as memory stream*

The Grid offers an option to export the data as a memory stream instead of downloading it as a file in the browser. To obtain the memory stream of the exported grid, set the `AsMemoryStream` parameter to **true** in the [ExcelExport](#) and [CsvExport](#) methods.

The following code demonstrates how to get the memory stream of exported grid.

```

`ts
public object ExcelExport(string gridModel)
{
  GridExcelExport exp = new GridExcelExport();
  Grid gridProperty = ConvertGridObject(gridModel);
  // pass third parameter as true to get the Memory Stream of exported grid data
  return (MemoryStream)exp.ExcelExport<OrdersDetails>(gridProperty, OrderRepository.GetAllRecords(),
  true);
}
public object CsvExport(string gridModel)
{

```



```

GridExcelExport exp = new GridExcelExport();
Grid gridProperty = ConvertGridObject(gridModel);
return (MemoryStream)exp.CsvExport<OrdersDetails>(gridProperty, OrderRepository.GetAllRecords(),
true);
}

```

#### *Merge grid's memory stream*

The [Essential XlsIO](#) library is used to merge multiple memory streams into a single stream. To learn more about the merge option, please refer to this [documentation](#).

You can merge a memory stream, a file stream, and a local file with the Grid's memory stream in the following ways:

#### *Merging with an existing memory stream*

If you already have a memory stream, you can directly use it to merge with the Grid's memory stream.

In the following code, `ExcelEngine` and `AddCopy` method of `Worksheets` are used to merge the grid's memory stream with an existing memory stream.

```

`ts
using Syncfusion.XlsIO;

public MemoryStream ms1; // defines existing memory stream
public object ExcelExport(string gridModel)
{
    GridExcelExport exp = new GridExcelExport();
    Grid gridProperty = ConvertGridObject(gridModel);
    // get the memory stream of exported grid data
    MemoryStream ms2 = (MemoryStream)exp.ExcelExport<OrdersDetails>(gridProperty,
    OrderRepository.GetAllRecords(), true);

    //New instance of ExcelEngine is created equivalent to launching Microsoft Excel with no workbooks
    open
    ExcelEngine excelEngine = new ExcelEngine();
    //Instantiate the Excel application object
    IApplication application = excelEngine.Excel;
    //Assigns default application version
    application.DefaultVersion = ExcelVersion.Xlsx;
    //open an workbook of existing memory stream and grid's memory stream through Open method of
    IWorkbooks
    IWorkbook sourceWorkbook = application.Workbooks.Open(ms1);
    IWorkbook destinationWorkbook = application.Workbooks.Open(ms2);
}

```

```
//Copy all the worksheet from the Source workbook to the destination workbook
for (int i = 0; i < sourceWorkbook.Worksheets.Count; i++)
{
    destinationWorkbook.Worksheets.AddCopy(sourceWorkbook.Worksheets[i]);
}
destinationWorkbook.ActiveSheetIndex = 1;
//Saving the workbook as stream
MemoryStream ms3 = new MemoryStream();
destinationWorkbook.SaveAs(ms3);
ms3.Position = 0;
//Dispose the instance of ExcelEngine
excelEngine.Dispose();
//Dispose the streams.
ms1.Dispose();
ms2.Dispose();
return ms3;
}
`
```

#### Merging with an existing file stream

If you already have a file stream, you can directly use it to merge with the Grid's memory stream. In the following code, the existing file stream is merged with the Grid's memory stream.

```
`ts
using Syncfusion.XlsIO;

public FileStream fs1; // defines existing file stream
public object ExcelExport(string gridModel)
{
    GridExcelExport exp = new GridExcelExport();
    Grid gridProperty = ConvertGridObject(gridModel);
    MemoryStream ms1 = (MemoryStream)exp.ExcelExport<OrdersDetails>(gridProperty,
    OrderRepository.GetAllRecords(), true);
    ExcelEngine excelEngine = new ExcelEngine();
    IApplication application = excelEngine.Excel;
    application.DefaultVersion = ExcelVersion.Xlsx;
    //fs1 and ms1 represents the existing stream and grid's stream.
```

```

IWorkbook sourceWorkbook = application.Workbooks.Open(fs1);
IWorkbook destinationWorkbook = application.Workbooks.Open(ms1);
for (int i = 0; i < sourceWorkbook.Worksheets.Count; i++)
{
    destinationWorkbook.Worksheets.AddCopy(sourceWorkbook.Worksheets[i]);
}
destinationWorkbook.ActiveSheetIndex = 1;
//Saving the workbook as stream
MemoryStream ms3 = new MemoryStream();
destinationWorkbook.SaveAs(ms3);
ms3.Position = 0;
return ms3;
}
`

```

#### Merging with a local file

To merge a local file with the Grid's memory stream, you need to convert it into a file stream before merging. In the following code, the existing local file is merged with the Grid's memory stream.

```

`ts
using Syncfusion.XlsIO;

// get the file stream of local file
public FileStream fs1 = new FileStream("D:/ExcelDoc.xlsx", FileMode.Open, FileAccess.Read); //
ExcelDoc.xlsx is a local file which is located in local disk D.

public object ExcelExport(string gridModel)
{
    GridExcelExport exp = new GridExcelExport();
    Grid gridProperty = ConvertGridObject(gridModel);
    MemoryStream ms1 = (MemoryStream)exp.ExcelExport<OrdersDetails>(gridProperty,
    OrderRepository.GetAllRecords(), true);
    ExcelEngine excelEngine = new ExcelEngine();
    IApplication application = excelEngine.Excel;
    application.DefaultVersion = ExcelVersion.Xlsx;
    //fs1 and ms1 represents the local file's stream and grid's stream.
    IWorkbook sourceWorkbook = application.Workbooks.Open(fs1);
    IWorkbook destinationWorkbook = application.Workbooks.Open(ms1);

```

```

for (int i = 0; i < sourceWorkbook.Worksheets.Count; i++)
{
    destinationWorkbook.Worksheets.AddCopy(sourceWorkbook.Worksheets[i]);
}
destinationWorkbook.ActiveSheetIndex = 1;
MemoryStream ms3 = new MemoryStream();
destinationWorkbook.SaveAs(ms3);
ms3.Position = 0;
return ms3;
}

```

#### Downloading the merged memory stream

You can download the merged memory stream by converting it into a `FileStreamResult`. In the following code, the merged memory stream is downloaded to the browser.

```

`ts
using Syncfusion.XlsIO;

public ActionResult ExcelExport(string gridModel)
{
    ExcelEngine excelEngine = new ExcelEngine();
    IApplication application = excelEngine.Excel;
    application.DefaultVersion = ExcelVersion.Xlsx;
    //open an workbook of streams through Open method of IWorkbooks
    IWorkbook sourceWorkbook = application.Workbooks.Open(ms1);
    IWorkbook destinationWorkbook = application.Workbooks.Open(ms2);
    for (int i = 0; i < sourceWorkbook.Worksheets.Count; i++)
    {
        destinationWorkbook.Worksheets.AddCopy(sourceWorkbook.Worksheets[i]);
    }
    destinationWorkbook.ActiveSheetIndex = 1;
    MemoryStream ms3 = new MemoryStream();
    destinationWorkbook.SaveAs(ms3);
    ms3.Position = 0;
    // Save the MemoryStream into FileStreamResult
}

```

```

FileStreamResult fileStreamResult = new FileStreamResult(ms3, "Application/vnd.openxmlformats-officedocument.spreadsheetml.sheet");
fileStreamResult.FileName = "Export.xlsx";
//Dispose the instance of ExcelEngine
excelEngine.Dispose();
//Dispose the streams.
ms1.Dispose();
ms2.Dispose();
// return the file
return fileStreamResult;
}

```

*Rotate a header text to a certain degree in the exported grid on the server side*

The DataGrid has support to customize the column header styles such as changing text orientation, the font color, and so on in the exported Excel file. To achieve this requirement, use the `ServerExcelHeaderQueryCellInfo` event of the Grid.

The `ServerExcelHeaderQueryCellInfo` will be triggered when creating a column header for the excel document to be exported in the server side. Customize the column header in this event.

In the following demo, using the `HeaderCellRotate` method of the `GridExcelExport` class in the `ServerExcelHeaderQueryCellInfo` event, you can rotate the header text of the column header in the excel exported document.

```

`ts
public ActionResult ExcelExport(string gridModel)
{
    GridExcelExport exp = new GridExcelExport();
    Grid gridProperty = ConvertGridObject(gridModel);
    gridProperty.ServerExcelHeaderQueryCellInfo = ExcelHeaderQueryCellInfo;
    IEnumerable data = Utils.DataTableToJson(dt);
    var result = exp.ExcelExport<dynamic>(gridProperty, data);
    return result;
}

private void ExcelHeaderQueryCellInfo(object excel)
{
    ServerExcelHeaderQueryCellInfoEventArgs name = (ServerExcelHeaderQueryCellInfoEventArgs)excel;
    headerValues.Add(name.Column.HeaderText);
}

```

```

var longestString = headerValues.Where(s => s.Length == headerValues.Max(m => m.Length)).First();
GridExcelExport exp = new GridExcelExport();
var size = exp.ExcelTextSize(name.Style.Font.FontName, (float)name.Style.Font.Size, longestString);
name.Cell.RowHeight = size.Width;
exp.HeaderCellRotate(name, 45); // Give the rotate degree value by the user.
name.Style.Borders.LineStyle = Syncfusion.XlsIO.ExcelLineStyle.None;
}
`

```

### Limitations

- The export feature for detail and caption templates is not supported in server-side exporting.
- Multiple grids exporting feature is not supported with server side exporting.

## Global local in Vue Grid component

### Localization

The **Localization** library allows you to localize default text content of the Grid. The grid component has static text on some features (like group drop area text, pager information text, etc.) that can be changed to other cultures (Arabic, Deutsch, French, etc.) by defining the [locale](#) value and translation object.

The following list of properties and its values are used in the grid.

Locale key words | Text

EmptyRecord | No records to display

True | true

False | false

InvalidFilterMessage | Invalid Filter Data

GroupDropArea | Drag a column header here to group its column

UnGroup | Click here to ungroup

GroupDisable | Grouping is disabled for this column

FilterbarTitle | \s filter bar cell

EmptyDataSourceError | DataSource must not be empty at initial load since columns are generated from dataSource in AutoGenerate Column Grid

Add | Add

Edit | Edit

Cancel | Cancel

Update | Update

Delete | Delete

Print | Print

Pdfexport | PDF Export

Excelexport | Excel Export

Wordexport | Word Export

Csvexport | CSV Export

Search | Search

Columnchooser | Columns

Save | Save

Item | item

Items | items

EditOperationAlert | No records selected for edit operation

DeleteOperationAlert | No records selected for delete operation

SaveButton | Save

OKButton | OK

CancelButton | Cancel

EditFormTitle | Details of

AddFormTitle | Add New Record

BatchSaveConfirm | Are you sure you want to save changes?

BatchSaveLostChanges | Unsaved changes will be lost. Are you sure you want to continue?

ConfirmDelete | Are you sure you want to Delete Record?

CancelEdit | Are you sure you want to Cancel the changes?

ChooseColumns | Choose Column

SearchColumns | search columns

Matches | No Matches Found

FilterButton | Filter

ClearButton | Clear

StartsWith | Starts With

EndsWith | Ends With

Contains | Contains

Equal | Equal

NotEqual | Not Equal

LessThan | Less Than

LessThanOrEqual | Less Than Or Equal

GreaterThan | Greater Than

GreaterThanOrEqual | Greater Than Or Equal

ChooseDate | Choose a Date

EnterValue | Enter the value

Copy | Copy

Group | Group by this column

Ungroup | Ungroup by this column

autoFitAll | AutoFit all columns

autoFit | AutoFit this column

Export | Export

FirstPage | First Page

LastPage | Last Page

PreviousPage | Previous Page

NextPage | Next Page

SortAscending | Sort Ascending

SortDescending | Sort Descending

EditRecord | Edit Record

DeleteRecord | Delete Record

FilterMenu | Filter

SelectAll | Select All

Blanks | Blanks

FilterTrue | True

FilterFalse | False

NoResult | No Matches Found

ClearFilter | Clear Filter

NumberFilter | Number Filters

TextFilter | Text Filters

DateFilter | Date Filters

MatchCase | Match Case

Between | Between

CustomFilter | Custom Filter

CustomFilterPlaceholder | Enter the value

CustomFilterDatePlaceholder | Choose a date

AND | AND



OR | OR

ShowRowsWhere | Show rows where:

currentPageInfo | {0} of {1} pages

totalItemsInfo | ({0} items)

totalItemInfo | ({0} item)

firstPageTooltip | Go to first page

lastPageTooltip | Go to last page

nextPageTooltip | Go to next page

previousPageTooltip | Go to previous page

nextPagerTooltip | Go to next pager items

previousPagerTooltip | Go to previous pager items

pagerDropDown | Items per page

pagerAllDropDown | Items

All | All

### [Loading translations](#)

To load translation object in an application use `load` function of `L10n` class.

The below example demonstrates the Grid in `Deutsch` culture.

### **APP.VUE**

```
<template>
  <div id="app">
    <ejs-grid :dataSource='data' locale='de-DE' :allowGrouping='true'
    :allowPaging='true' :pageSettings='pageOptions' height='220px'>
      <e-columns>
        <e-column field='OrderID' headerText='Order ID'
        textAlign='Right' width=120></e-column>
        <e-column field='CustomerID' headerText='Customer ID'
        width=150></e-column>
        <e-column field='ShipCity' headerText='Ship City'
        width=150></e-column>
        <e-column field='ShipName' headerText='Ship Name'
        width=150></e-column>
      </e-columns>
    </ejs-grid>
  </div>
</template>
<script>
import Vue from "vue";
import { L10n, setCulture } from '@syncfusion/ej2-base';
import { GridPlugin, Page, Group } from "@syncfusion/ej2-vue-grids";
import { data } from './datasource.js';
setCulture('de-DE');
L10n.load({
  'de-DE': {
    'grid': {
```

```

        'EmptyRecord': 'Keine Aufzeichnungen angezeigt',
        'GroupDropArea': 'Ziehen Sie einen Spaltenkopf hier, um die
Gruppe ihre Spalte',
        'UnGroup': 'Klicken Sie hier, um die Gruppierung aufheben',
        'EmptyDataSourceError': 'DataSource darf bei der Erstaustauslastung
nicht leer sein, da Spalten aus der dataSource im AutoGenerate
Spaltenraster',
        'Item': 'Artikel',
        'Items': 'Artikel'
    },
    'pager': {
        'currentPageInfo': '{0} von {1} Seiten',
        'totalItemsInfo': '({0} Beiträge)',
        'firstPageTooltip': 'Zur ersten Seite',
        'lastPageTooltip': 'Zur letzten Seite',
        'nextPageTooltip': 'Zur nächsten Seite',
        'previousPageTooltip': 'Zurück zur letzten Seit',
        'nextPagerTooltip': 'Gehen Sie zu den nächsten Pager-Elementen',
        'previousPagerTooltip': 'Gehen Sie zu vorherigen Pager-
Elementen'
    }
    });
Vue.use(GridPlugin);
export default {
    data() {
        return {
            data: data,
            pageOptions: { pageSize: 7 }
        };
    },
    provide: {
        grid: [Page, Group]
    }
}
</script>
<style>
    @import
"https://ej2.syncfusion.com/vue/documentation/node_modules/@syncfusion/ej2-
vue-grids/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/grid/databind/default-cs5" %}

### Internationalization

The **Internationalization** library is used to globalize number, date, and time values in grid component using format strings in the [columns.format](#).

### APP.VUE

```

<template>
    <div id="app">
        <ejs-grid :dataSource='data' locale='de-DE' height='315px'>
            <e-columns>
                <e-column field='OrderID' headerText='Order ID'
textAlign='Right' width=120></e-column>

```

```

        <e-column field='CustomerID' headerText='Customer ID'
width=150></e-column>
        <e-column field='Freight' headerText='Freight'
:format='formatOptions' textAlign='Right' width=150></e-column>
        <e-column field='OrderDate' headerText='Order Date'
:format='dateFormatOptions' textAlign='Right' width=150></e-column>
        <e-column field='ShipName' headerText='Ship Name'
width=150></e-column>
    </e-columns>
</ejs-grid>
</div>
</template>
<script>
import Vue from "vue";
import { L10n, loadCldr, setCulture, setCurrencyCode } from
"@syncfusion/ej2-base";
import * as currencies from './currencies.json';
import * as cagregorian from './ca-gregorian.json';
import * as numbers from './numbers.json';
import * as timeZoneNames from './timeZoneNames.json';
import * as numberingSystems from './numberingSystems.json';
import { GridPlugin } from "@syncfusion/ej2-vue-grids";
import { data } from './datasource.js';
setCulture('de-DE');
setCurrencyCode('EUR');
L10n.load({
    'de-DE': {
        'grid': {
            'EmptyRecord': 'Keine Aufzeichnungen angezeigt',
            'GroupDropArea': 'Ziehen Sie einen Spaltenkopf hier, um die
Gruppe ihre Spalte',
            'UnGroup': 'Klicken Sie hier, um die Gruppierung aufheben',
            'EmptyDataSourceError': 'DataSource darf bei der Erstaustlastung
nicht leer sein, da Spalten aus der dataSource im AutoGenerate
Spaltenraster',
            'Item': 'Artikel',
            'Items': 'Artikel'
        },
        'pager': {
            'currentPageInfo': '{0} von {1} Seiten',
            'totalItemsInfo': '({0} Beiträge)',
            'firstPageTooltip': 'Zur ersten Seite',
            'lastPageTooltip': 'Zur letzten Seite',
            'nextPageTooltip': 'Zur nächsten Seite',
            'previousPageTooltip': 'Zurück zur letzten Seit',
            'nextPagerTooltip': 'Gehen Sie zu den nächsten Pager-Elementen',
            'previousPagerTooltip': 'Gehen Sie zu vorherigen Pager-
Elementen'
        }
    }
});
Vue.use(GridPlugin);
export default {
    data() {
        return {
            data: data,

```

```

        formatOptions: { format: 'C2' , useGrouping: false,
        minimumSignificantDigits:1, maximumSignificantDigits:3, currency: 'EUR' },
        dateFormatOptions: {type: 'date', format: 'dd.MM.yy'}
    };
}
}
</script>
<style>
    @import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/grid/databind/default-cs6" %}

\* In the above sample, **Freight** column is formatted by **NumberFormatOptions**.

\* By default, [locale](#) value is **en-US**. If you want to change **en-US** culture, then set the [locale](#).

### Right to Left - RTL

RTL provides an option to switch the text direction and layout of Grid component from right to left. It improves the user experiences and accessibility for users who use right-to-left languages (Arabic, Farsi, Urdu, etc). To enable RTL in the Grid, set the [enableRtl](#) to true.

### APP.VUE

```

<template>
    <div id="app">
        <ejs-grid :dataSource='data' :enableRtl='true' locale='ar-AE'
        :allowPaging='true' :pageSettings='pageOptions'>
            <e-columns>
                <e-column field='OrderID' headerText='Order ID'
                textAlign='Right' width=120></e-column>
                <e-column field='CustomerID' headerText='Customer ID'
                width=150></e-column>
                <e-column field='ShipCity' headerText='Ship City'
                width=150></e-column>
                <e-column field='ShipName' headerText='Ship Name'
                width=150></e-column>
            </e-columns>
        </ejs-grid>
    </div>
</template>
<script>
import Vue from "vue";
import { L10n, setCulture } from '@syncfusion/ej2-base';
import { GridPlugin, Page } from "@syncfusion/ej2-vue-grids";
import { data } from './datasource.js';
setCulture('ar-AE');
L10n.load({
    'ar-AE': {
        'grid': {
            'EmptyRecord': 'لا سجلات لعرضها',
            'EmptyDataSourceError': 'يجب أن يكون مصدر البيانات فارغة في التحميل الأولي منذ يتم إنشاء الأعمدة من مصدر البيانات في أوتوجينيراتد عمود الشبكة'
        },
        'pager': {

```

```

        'currentPageInfo': '{0} صفحة 1 {من}',
        'totalItemsInfo': '({0} العناصر)',
        'firstPageTooltip': 'انتقل إلى الصفحة الأولى',
        'lastPageTooltip': 'انتقل إلى الصفحة الأخيرة',
        'nextPageTooltip': 'انتقل إلى الصفحة التالية',
        'previousPageTooltip': 'انتقل إلى الصفحة السابقة',
        'nextPagerTooltip': 'انتقل إلى عناصر بيكر التالية',
        'previousPagerTooltip': 'للذهاب إلى عناصر بيكر السابقة'
    }
  }
});
Vue.use(GridPlugin);
export default {
  data() {
    return {
      data: data,
      pageOptions: { pageSize: 7 }
    };
  },
  provide: {
    grid: [Page]
  }
}
</script>
<style>
  @import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/grid/databind/default-cs7" %}

See Also

- [How to apply CHF currency format in a column in Vue Grid](#)
- [Filter the value with locale and custom format the Vue Grid](#)
- [How to customize edit dialog button in Vue Grid](#)

## Accessibility in Vue Grid component

The Grid component followed the accessibility guidelines and standards, including [ADA](#), [Section 508](#), [WCAG 2.2](#) standards, and [WCAG roles](#) that are commonly used to evaluate accessibility.

### WAI-ARIA attributes

The Grid component followed the [WAI-ARIA](#) patterns to meet the accessibility. The following ARIA attributes are used in the Grid component:

Attributes	Purpose
---	---
<code>role=grid</code>	To represent the element containing the grid component.
<code>role=row</code>	To represent the element containing the cells of the row in the grid.
<code>role=rowgroup</code>	To represent the group of rows in the grid.

| **role=columnheader** | To represent the cell in a row contains header information for a column in the grid. |

| **role=gridcell** | To represent a cell in the grid component. |

| **role=button** | To represent the element that acts as a button in the grid. |

| **role=search** | To represent the element that acts as a search region in the grid. |

| **role=presentation** | To represent the element to be not available for accessibility concerns. |

| **role=navigation** | To represent the element containing pager elements to navigate from one page to another. |

| **aria-colindex** | Defines the column index of the column with respect to the total number of columns within the grid. |

| **aria-rowindex** | Defines row index of the row with respect to the total number of rows within the grid. |

| **aria-rowspan** | Defines the number of rows spanned by a cell within the grid. |

| **aria-colspan** | Defines the number of columns spanned by a cell within the grid. |

| **aria-rowcount** | Defines the total number of rows in the grid. |

| **aria-colcount** | Defines the total number of columns in the grid. |

| **aria-selected** | Indicates the current "selected" state of the rows and cells in the grid. |

| **aria-expanded** | Indicate if the expand icon in the hierarchy grid or grouped grid or detail grid is expanded or collapsed |

| **aria-sort** | Indicates whether the data in the grid are sorted in ascending or descending order. |

| **aria-busy** | Indicates an element is being modified and that assistive technologies may want to wait until the changes are complete before informing the user about the update. |

| **aria-owns** | Identifies an element in order to define a visual, functional, or contextual relationship between a parent and its child elements. |

| **aria-hidden** | Hides the element from accessibility concerns. |

| **aria-labelledby** | Provides an accessible name for the checkbox labels in excel filter, checkbox filter and column chooser dialog. |

| **aria-describedby** | Provides an description about the features enabled in the header when the grid header cell is focused. |

### Keyboard interaction

The Grid component followed the [keyboard interaction](#) guideline, making it easy for people who use assistive technologies (AT) and those who completely rely on keyboard navigation. The following keyboard shortcuts are supported by the Grid component.

<b>Pager</b>

**Windows | MAC | To do this**

**Home | Fn + Left Arrow | Moves the focus to the first cell of the focused row.**

**End | Fn + Right Arrow | Moves the focus to the last cell of the focused row.**

**Ctrl + Home | Command + Fn + Left Arrow | Moves the focus to the first Cell of the first row in the grid.**

**Ctrl + End | Command + Fn + Right Arrow | Moves the focus to the last Cell of the last row in the grid.**

**Up Arrow | Up Arrow | Moves the cell focus upward from the focused cell.**

**Down Arrow | Down Arrow | Moves the cell focus downward from the focused cell.**

**Right Arrow | Right Arrow | Moves the cell focus right side from the focused cell.**

**Left Arrow | Left Arrow | Moves the cell focus left side from the focused cell.**

**Alt + J | Alt + J | Moves the focus to the entire grid.**

**Alt + W | Alt + W | Move the focus to the grid content element.**

**<b>Selection</b>**

**Windows | MAC | To do this**

**Ctrl + Up Arrow | Command + Up Arrow | Collapses all the visible groups.**

**Ctrl + Down Arrow | Command + Down Arrow | Expands all the visible groups.**

**Ctrl + Space | Ctrl + Space | Performs grouping when focused on a header element.**

**Enter | Enter | If the current cell is an expand/collapse cell then expands/collapses the current group/detailrow/childgrid.**

**<b>Print</b>**

**Windows | MAC | To do this**

**Ctrl + C | Command + C | Copies selected rows or cells data into the clipboard.**

**Ctrl + Shift + H | Ctrl + Shift + H | Copies selected rows or cells data with header into clipboard**

**<b>Editing</b>**

**Windows | MAC | To do this**

**Alt + Down arrow | Alt + Down arrow | Opens the filter menu(excel, menu and checkbox filter) when its header element is in focused state.**

**<b>Column Menu</b>**

**Windows | MAC | To do this**

**Ctrl + left arrow or right arrow | Command + left arrow or right arrow | Reorders the focused header column to the left or right side.**

**<b>Sorting</b>**

**Windows | MAC | To do this**

**Enter | Enter | Performs sorting(ascending/descending) on a column when its header element is in focused state.**

**Ctrl + Enter | Command + Enter | Performs multi-sorting on a column when its header element is in focused state.**

**Shift + Enter | Shift + Enter | Clears sorting for the focused header column.**

<br>

**\* The Command and Control keys on Mac devices can be interchanged. When this switch occurs, use the Command key in place of the Control key and the Control key in place of the Command key for the above listed key interactions with Mac devices.**

**\* For example, after switching the keys to group the columns when the header element is focused use Command + Space and for expanding the visible groups use Ctrl + Down Arrow.**

### Ensuring accessibility

The Grid component's accessibility levels are ensured through an [accessibility-checker](#) and [axe-core](#) software tools during automated testing.

The accessibility compliance of the Grid component is shown in the following sample. Open the [sample](#) in a new window to evaluate the accessibility of the Grid component with accessibility tools.

{% previewsample "https://ej2.syncfusion.com/accessibility/grid.html" %}

[See also](#)

- [Accessibility in Syncfusion Vue components](#)

### Clipboard in Vue Grid component

The clipboard provides an option to copy selected rows or cells data into the clipboard.

The following list of keyboard shortcuts is supported in the Grid to copy selected rows or cells data into clipboard.

Interaction keys | Description

**Ctrl + C | Copy selected rows or cells data into clipboard.**

**Ctrl + Shift + H | Copy selected rows or cells data with header into clipboard.**

### APP.VUE

```
<template>
  <div id="app">
    <ejs-grid :dataSource='data' height='315px'
      :selectionSettings='selectionOptions'>
      <e-columns>
        <e-column field='OrderID' headerText='Order ID'
          textAlign='Right' width=120></e-column>
        <e-column field='CustomerID' headerText='Customer ID'
          width=150></e-column>
        <e-column field='ShipCity' headerText='Ship City'
          width=150></e-column>
      </e-columns>
    </ejs-grid>
  </div>
</template>
```



```

        <e-column field='ShipName' headerText='Ship Name'
width=150></e-column>
    </e-columns>
</ejs-grid>
</div>
</template>
<script>
import Vue from "vue";
import { GridPlugin } from "@syncfusion/ej2-vue-grids";
import { data } from './datasource.js';
Vue.use(GridPlugin);
export default {
    data() {
        return {
            data: data,
            selectionOptions: { type: 'Multiple' }
        };
    }
}
</script>
<style>
    @import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/grid/databind/default-cs1" %}

### Copy to clipboard by external buttons

To copy selected rows or cells data into clipboard with help of external buttons, you need to invoke the [copy](#) method.

### APP.VUE

```

<template>
    <div id="app">
        <ejs-button id='copy' @click.native='copy'>Copy</ejs-button>
        <ejs-button id='copyHeader'
@click.native='copyHeader'>CopyHeader</ejs-button>
        <ejs-grid ref='grid' :dataSource='data' height='280px'
:selectionSettings='selectionOptions'>
            <e-columns>
                <e-column field='OrderID' headerText='Order ID'
textAlign='Right' width=120></e-column>
                <e-column field='CustomerID' headerText='Customer ID'
width=150></e-column>
                <e-column field='ShipCity' headerText='Ship City'
width=150></e-column>
                <e-column field='ShipName' headerText='Ship Name'
width=150></e-column>
            </e-columns>
        </ejs-grid>
    </div>
</template>
<script>
import Vue from "vue";
import { GridPlugin } from "@syncfusion/ej2-vue-grids";
import { ButtonPlugin } from "@syncfusion/ej2-vue-buttons";

```

```

import { data } from './datasource.js';
Vue.use(GridPlugin);
Vue.use(ButtonPlugin);
export default {
  data() {
    return {
      data: data,
      selectionOptions: { type: 'Multiple' }
    };
  },
  methods: {
    copy: function() {
      this.$refs.grid.copy();
    },
    copyHeader: function() {
      this.$refs.grid.copy(true);
    }
  }
}
</script>
<style>
  @import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/grid/print/default-cs1" %}

### AutoFill

AutoFill Feature allows you to copy the data of selected cells and paste it to another cells by just dragging the autofill icon of the selected cells up to required cells. This feature is enabled by defining [enableAutoFill](#) property as true.

### APP.VUE

```

<template>
  <div id="app">
    <ejs-grid :dataSource='data' :enableAutoFill= "true"
      :selectionSettings='selectionOptions' :editSettings='editSettings'
      :toolbar='toolbar' height='273px'>
      <e-columns>
        <e-column field='OrderID' headerText='Order ID'
          textAlign='Right' :isPrimaryKey='true' :visible='false' width=120></e-
          column>
        <e-column field='CustomerID' headerText='Customer ID'
          width=150></e-column>
        <e-column field='ShipCity' headerText='Ship City'
          width=150></e-column>
        <e-column field='ShipCountry' headerText='ShipCountry'
          width=150></e-column>
        <e-column field='ShipName' headerText='Ship Name'
          width=150></e-column>
      </e-columns>
    </ejs-grid>
  </div>
</template>
<script>
import Vue from "vue";

```

```

import { GridPlugin, Toolbar, Edit } from "@syncfusion/ej2-vue-grids";
import { data } from './datasource.js';
Vue.use(GridPlugin);
export default {
  data() {
    return {
      data: data,
      editSettings: { allowEditing: true, allowAdding: true, allowDeleting:
true, mode: 'Batch' },
      toolbar: ['Add', 'Update', 'Cancel'],
      selectionOptions: { type: 'Multiple', mode: 'Cell', cellSelectionMode:
'Box' }
    };
  },
  provide: {
    grid: [Edit, Toolbar]
  }
}
</script>
<style>
@import
"https://ej2.syncfusion.com/vue/documentation/node_modules/@syncfusion/ej2-
vue-grids/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/grid/databind/default-cs2" %}

\* If [enableAutoFill](#) is set to true, then the autofill icon will be displayed on cell selection to copy cells.

\* It requires the selection [mode](#) to be **Cell**, [cellSelectionMode](#) to be **Box** and also Batch Editing should be enabled.

#### Limitations of AutoFill

- Since the string values are not parsed to number and date type, so when the selected string type cells are dragged to number type cells then it will display as **NaN**. For date type cells, when the selected string type cells are dragged to date type cells then it will display as an **empty cell**.
- Linear series and the sequential data generations are not supported in this autofill feature.

#### Paste

You can able to copy the content of a cell or a group of cells by selecting the cells and pressing Ctrl + C shortcut key and paste it to another set of cells by selecting the cells and pressing Ctrl + V shortcut key.

#### APP.VUE

```

<template>
  <div id="app">
    <ejs-grid :dataSource='data' :selectionSettings='selectionOptions'
:editSettings='editSettings' :toolbar='toolbar' height='273px'>
      <e-columns>
        <e-column field='OrderID' headerText='Order ID'
textAlign='Right' :isPrimaryKey='true' :visible='false' width=120></e-
column>

```

```

        <e-column field='CustomerID' headerText='Customer ID'
width=150></e-column>
        <e-column field='ShipCity' headerText='Ship City'
width=150></e-column>
        <e-column field='ShipCountry' headerText='ShipCountry'
width=150></e-column>
        <e-column field='ShipName' headerText='Ship Name'
width=150></e-column>
    </e-columns>
</ejs-grid>
</div>
</template>
<script>
import Vue from "vue";
import { GridPlugin, Toolbar, Edit } from "@syncfusion/ej2-vue-grids";
import { data } from './datasource.js';
Vue.use(GridPlugin);
export default {
  data() {
    return {
      data: data,
      editSettings: { allowEditing: true, allowAdding: true, allowDeleting:
true, mode: 'Batch' },
      toolbar: ['Add', 'Update', 'Cancel'],
      selectionOptions: { type: 'Multiple', mode: 'Cell', cellSelectionMode:
'Box' }
    };
  },
  provide: {
    grid: [Edit, Toolbar]
  }
}
</script>
<style>
@import
"https://ej2.syncfusion.com/vue/documentation/node_modules/@syncfusion/ej2-
vue-grids/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/grid/databind/default-cs3" %}

To perform paste functionality, it requires the selection [mode](#) to be **Cell**, [cellSelectionMode](#) to be **Box** and also Batch Editing should be enabled.

#### *Limitations of Paste Functionality*

- Since the string values are not parsed to number and date type, so when the copied string type cells are pasted to number type cells then it will display as **NaN**. For date type cells, when the copied string format cells are pasted to date type cells then it will display as an **empty cell**.

### Context menu in Vue Grid component

The Grid has options to show the context menu when right clicked on it. To enable this feature, you need to define either default or custom item in the [contextMenuItems](#).

To use the context menu, inject the **ContextMenu** module in the **provide** section.

The default items are in the following table.

Items | Description

**AutoFit** | Auto fit the current column.

**AutoFitAll** | Auto fit all columns.

**Edit** | Edit the current record.

**Delete** | Delete the current record.

**Save** | Save the edited record.

**Cancel** | Cancel the edited state.

**Copy** | Copy the selected records.

**PdfExport** | Export the grid data as Pdf document.

**ExcelExport** | Export the grid data as Excel document.

**CsvExport** | Export the grid data as CSV document.

**Group** | Group the current column.

**Ungroup** | Ungroup the current column.

**SortAscending** | Sort the current column in ascending order.

**SortDescending** | Sort the current column in descending order.

**FirstPage** | Go to the first page.

**PrevPage** | Go to the previous page.

**LastPage** | Go to the last page.

**NextPage** | Go to the next page.

#### APP.VUE

```
<template>
  <div id="app">
    <ejs-grid :dataSource='data' id="gridcomp" :allowPaging='true'
:allowExcelExport='true' :allowPdfExport='true' height='215px'
:allowSorting='true'
      :contextMenuItems="contextMenuItems" :editSettings='editing'
:allowGrouping='true'>
      <e-columns>
        <e-column field='OrderID' headerText='Order ID' width='120'
textAlign="Right" isPrimaryKey='true'></e-column>
        <e-column field='CustomerID' headerText='Customer Name'></e-
column>
        <e-column field='Freight' headerText='Freight' format='C2'
textAlign="Right" editType='numericedit'></e-column>
        <e-column field='ShipCity' headerText='Ship City'
width='150'></e-column>
      </e-columns>
    </div>
  </template>
```

```

    </ejs-grid>
  </div>
</template>
<script>
import Vue from "vue";
import { GridPlugin, ContextMenu, Page, Resize, Sort, Group, Edit,
PdfExport, ExcelExport } from "@syncfusion/ej2-vue-grids";
import { data } from './datasource.js';
Vue.use(GridPlugin);
export default {
  data() {
    return {
      data: data,
      contextMenuItems: ['AutoFit', 'AutoFitAll', 'SortAscending',
'SortDescending',
'Copy', 'Edit', 'Delete', 'Save', 'Cancel',
'PdfExport', 'ExcelExport', 'CsvExport', 'FirstPage',
'PrevPage',
'LastPage', 'NextPage', 'Group', 'Ungroup'],
      editing: {allowEditing: true, allowDeleting: true}
    };
  },
  provide : {
    grid: [ContextMenu, Page, Resize, Sort, Group, Edit, PdfExport,
ExcelExport]
  }
}
</script>
<style>
@import
"https://ej2.syncfusion.com/vue/documentation/node_modules/@syncfusion/ej2-
vue-grids/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/grid/contextMenu/default-cs1" %}

### Custom context menu items

The custom context menu items can be added by defining the [contextMenuItems](#) as a collection of [contextMenuItemModel](#). Actions for this customized items can be defined in the [contextMenuClick](#) event.

### APP.VUE

```

<template>
  <div id="app">
    <ejs-grid ref='grid' id='gridcomp' :dataSource='data'
:allowSelection='true' :allowPaging='true' height='265px'
:contextMenuItems='contextMenuItems' :contextMenuClick='contextMenuClick'>
      <e-columns>
        <e-column field='EmployeeID' :isPrimaryKey='true'
headerText='Employee ID' textAlign='Right' width=120></e-column>
        <e-column field='FirstName' headerText='First Name'
width=150></e-column>
        <e-column field='LastName' headerText='Last Name'
width=150></e-column>
      </e-columns>
    </ejs-grid>
  </div>
</template>

```

```

        <e-column field='City' headerText='City' width=150></e-
column>
    </e-columns>
</ejs-grid>
</div>
</template>
<script>
import Vue from "vue";
import { GridPlugin, ContextMenu, Page } from "@syncfusion/ej2-vue-grids";
import { employeeData } from './datasource.js';
Vue.use(GridPlugin);
export default {
    data() {
        return {
            data: employeeData,
            contextMenuItems: [{text: 'Copy with headers', target: '.e-content',
id: 'copywithheader'}]
        };
    },
    methods: {
        contextMenuClick: function(args) {
            if(args.item.id === 'copywithheader') {
                this.$refs.grid.copy(true);
            }
        }
    },
    provide: {
        grid: [ContextMenu, Page]
    }
}
</script>
<style>
@import
"https://ej2.syncfusion.com/vue/documentation/node_modules/@syncfusion/ej2-
vue-grids/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/grid/contextMenu/default-cs2" %}

### Show context menu on left click

By default, the context menu items will be shown in the Grid using the right mouse click action. Show the context menu items during the left mouse click action using the [created](#) and context menu's `beforeOpen` events of the Grid.

Using the `onclick` eventlistener of Grid , you can get the clicked position values and send them to the `open` method of the context menu in the `onclick` event of the Grid. Also, we have prevented the default right click action to open the context menu items using the [created](#) event of the Grid.

This is demonstrated in the following sample.

### APP.VUE

```

<template>
    <div id="app">
        <ejs-grid ref='grid' :dataSource='data' :allowPaging='true'
            height='265px'

```

```

        :contextMenuItems='contextMenuItems'
        :editSettings = 'editOptions'
        :created = 'created' v-on:click.native="clicked">
            <e-columns>
                <e-column field='EmployeeID' :isPrimaryKey='true'
headerText='Employee ID'
                textAlign='Right' width=120></e-column>
                <e-column field='FirstName' headerText='FirstName'
width=150></e-column>
                <e-column field='LastName' headerText='Last Name'
width=150></e-column>
                <e-column field='City' headerText='City' width=150></e-
column>
            </e-columns>
        </ejs-grid>
    </div>
</template>
<script>
import Vue from "vue";
import { GridPlugin, ContextMenu, Page, Edit } from "@syncfusion/ej2-vue-
grids";
import { employeeData } from './datasource.js';
Vue.use(GridPlugin);
export default {
    data() {
        return {
            values: "",
            data: employeeData,
            contextMenuItems: ['Copy', 'Edit', 'Delete', 'Save', 'Cancel'],
            editOptions: {
                allowDeleting: true,
                allowEditing: true,
                allowAdding: true,
            },
        };
    },
    methods: {
        created: function(args) {
            this.$refs.grid.ej2Instances.contextMenuModule.contextMenu.beforeOpen
= (
                args
            ) => {
                if (args.event && args.event.which === 3) args.cancel = true;
                args.event = this.values;

            this.$refs.grid.ej2Instances.contextMenuModule.contextMenuBeforeOpen(
                args
            );
        };
    },
    clicked: function(event) {
        this.values = event;
        this.$refs.grid.ej2Instances.contextMenuModule.contextMenu.open(
            this.values.pageY + pageYOffset,
            this.values.pageX + pageXOffset
        );
    }
}

```



```

    },
    provide: {
      grid: [ContextMenu, Page, Edit]
    }
  }
</script>
<style>
  @import
  "https://ej2.syncfusion.com/vue/documentation/node_modules/@syncfusion/ej2-
  vue-grids/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/grid/contextMenu/default-cs3" %}

You can hide or show an item in context menu for specific area inside of grid by defining the [target](#) property.

### Enable or disable context menu items

It is possible to enable or disable the default and custom context menu items in the Grid component. This is achieved by using the [enableItems](#) method of the ContextMenu. To enable or disable menu items, set the **enable** parameter in the **enableItems** method to true, and vice versa.

In the following sample, the Copy item is enabled or disabled based on some condition (as per the needs of the application) in the [rowSelected](#) event of the Grid.

### APP.VUE

```

<template>
  <div id="app">
    <ejs-grid ref='grid' :dataSource='data' id="gridcomp"
    :allowPaging='true' height='215px'
    :contextMenuItems="contextMenuItems" :editSettings='editSettings'
    :rowSelected='rowSelected'>
      <e-columns>
        <e-column field='OrderID' headerText='Order ID' width='120'
        textAlign="Right" isPrimaryKey='true'></e-column>
        <e-column field='CustomerID' headerText='Customer Name'></e-
        column>
        <e-column field='Freight' headerText='Freight' format='C2'
        textAlign="Right" editType='numericedit'></e-column>
        <e-column field='ShipCity' headerText='Ship City'
        width='150'></e-column>
      </e-columns>
    </ejs-grid>
  </div>
</template>
<script>
import Vue from "vue";
import { GridPlugin, ContextMenu, Page, Edit } from "@syncfusion/ej2-vue-
grids";
import { data } from './datasource.js';
Vue.use(GridPlugin);
export default {
  data() {
    return {
      data: data,

```

```

        contextMenuItems: ['Copy', 'Edit', 'Delete'],
        editSettings: { allowEditing: true, allowAdding: true, allowDeleting:
true }
    };
    },
    methods: {
        rowSelected: function (args) {
            if (args.data.OrderID % 2 === 0) {

this.$refs.grid.ej2Instances.contextMenuModule.contextMenu.enableItems(['Cop
y'], false);
            } else {

this.$refs.grid.ej2Instances.contextMenuModule.contextMenu.enableItems(['Cop
y'], true);
            }
        },
        provide : {
            grid: [ContextMenu, Page, Edit]
        }
    }
</script>
<style>
    @import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/grid/contextMenu/default-cs4" %}

### Loading animation in Vue Grid component

The grid has an option to show a loading indicator in-between the time of fetching the data and binding it to the grid during initial rendering or refreshing or after performing any grid action like sorting, filtering, grouping, and more. The grid supports two indicator types, which is achieved by setting the `loadingIndicator.indicatorType` property to `Spinner` or `Shimmer`. The default value of the indicator type is "Spinner."

In the following sample, the Shimmer indicator is displayed while the grid is loading and refreshing when using the remote data.

#### APP.VUE

```

<template>
    <div id="app">
        <ejs-grid :dataSource="data" :allowSorting='true'
:allowFiltering='true' :allowPaging="true" :pageSettings='pageSettings'
:loadingIndicator='loadingIndicator'>
            <e-columns>
                <e-column field='OrderID' headerText='Order ID'
textAlign='Right' width=90></e-column>
                <e-column field='CustomerID' headerText='Customer ID'
width=120></e-column>
                <e-column field='Freight' headerText='Freight' textAlign='Right'
format='C2' width=90></e-column>
                <e-column field='OrderDate' headerText='Order Date'
textAlign='Right' format='yMd' type='date' width=120></e-column>
            </e-columns>
        </ejs-grid>
    </div>
</template>

```

```

        </ejs-grid>
    </div>
</template>
<script>
import Vue from "vue";
import { GridPlugin, Page, Sort, Filter } from "@syncfusion/ej2-vue-grids";
import { DataManager, ODataAdaptor } from "@syncfusion/ej2-data";
Vue.use(GridPlugin);
export default {
  data() {
    return {
      data: new DataManager({
        url: 'https://services.syncfusion.com/js/production/api/Orders/',
        adaptor: new ODataAdaptor()
      }),
      pageSettings: { pageCount: 3 },
      loadingIndicator: { indicatorType: 'Shimmer' },
    };
  },
  provide: {
    grid: [Page, Sort, Filter]
  }
}
</script>
<style>
  @import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/grid/databind/remote-cs7" %}

## Style And Appearance

### Style and appearance in Vue Grid component

To modify the Grid appearance, you need to override the default CSS of grid. Please find the CSS structure that can be used to modify the Grid appearance. Also, you have an option to create your own custom theme for all the JavaScript controls using our [Theme Studio](#).

#### Customizing the Grid root element

Use the below CSS to customize the Grid root element.

```

,

.e-grid {
font-family: cursive;
}
,

```

You can modify the grid styling appearance by overriding the default CSS style of the Grid.

In the following sample, the font family of grid content is changed to **cursive**, and the background color of rows, selected rows, alternate rows, and row hovering color is modified using the below CSS styles.

#### APP.VUE

```

<template>

```

```

    <div id="app">
      <ejs-grid ref='grid' :dataSource='data' :allowPaging="true"
:pageSettings='pageSettings' :selectionSettings='selectionOptions'
height='272px'>
        <e-columns>
          <e-column field='OrderID' headerText='Order ID'
type='number' textAlign='Right' :isPrimaryKey='true' width=100></e-column>
          <e-column field='CustomerID' headerText='Customer ID'
type='string' width=120></e-column>
          <e-column field='Freight' headerText='Freight' type='number'
format='C2' textAlign='Right' width=100></e-column>
          <e-column field='ShipName' headerText='Ship Name'
type='string' width=180></e-column>
        </e-columns>
      </ejs-grid>
    </div>
  </template>
  <script>
import Vue from "vue";
import { GridPlugin, Page } from "@syncfusion/ej2-vue-grids";
import { data } from './datasource.js';
Vue.use(GridPlugin);
export default {
  data() {
    return {
      data: data,
      pageSettings: { pageSize: 8 },
      selectionOptions: { type: 'Multiple' }
    };
  },
  provide: {
    grid: [Page]
  }
}
  </script>
  <style>
    @import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
    .e-grid {
      font-family: cursive;
    }
    .e-grid .e-row:hover .e-rowcell {
      background-color: rgb(204, 229, 255) !important;
    }
    .e-grid .e-rowcell.e-selectionbackground {
      background-color: rgb(230, 230, 250);
    }
    .e-grid .e-row.e-altrow {
      background-color: rgb(150, 212, 212);
    }
    .e-grid .e-row {
      background-color: rgb(180, 180, 180);
    }
  </style>

```

{% previewsample "page.domainurl/code-snippet/grid/pdf/multiple-cs2" %}

## Header in Vue Grid component

### *Customizing the Grid header*

Use the below CSS to customize the Grid header root element. Using this, you can override the thin line between header and content of the grid.

```
,  
  
.e-grid .e-gridheader {  
border:2px solid green  
}  
,
```

### *Customizing the Grid header cell*

You can customize the Grid header cell elements using this CSS.

```
,  
  
.e-grid .e-headercell {  
color: darkblue;  
}  
,
```

### *Customizing the Grid header cell div element*

Use the below CSS to customize the Grid header cell div element. Using this CSS, you can customize the header text content.

```
,  
  
.e-grid .e-headercelldiv {  
font-size: 15px;  
}  
,
```

## Paging in Vue Grid component

### *Customizing the Grid pager root element*

Using this CSS, you can customize the Grid pager root element.

```
,  
  
.e-grid .e-gridpager {  
font-family: cursive;  
background-color: #deecf9;  
}  
,
```

### *Customizing the Grid pager container element*

Use the below CSS to customize the Grid pager container element.

```
,
```

```
.e-grid .e-pagercontainer {  
background-color: #deecf9;  
}  
、
```

#### *Customizing the Grid pager navigation elements*

Customize the Grid pager navigation elements, using the below selector.

```
、  
  
.e-grid .e-gridpager .e-prevpagedisabled,  
.e-grid .e-gridpager .e-prevpage,  
.e-grid .e-gridpager .e-nextpage,  
.e-grid .e-gridpager .e-nextpagedisabled,  
.e-grid .e-gridpager .e-lastpagedisabled,  
.e-grid .e-gridpager .e-lastpage,  
.e-grid .e-gridpager .e-firstpage,  
.e-grid .e-gridpager .e-firstpagedisabled {  
background-color: #deecf9;  
}  
、
```

#### *Customizing the Grid pager page numeric link elements*

Use the below CSS to customize the Grid pager current page numeric link elements.

```
、  
  
.e-grid .e-gridpager .e-numericitem {  
border-radius: initial;  
}  
、
```

#### *Customizing the Grid pager current page numeric element*

Using this CSS, you can customize the Grid pager current page numeric item.

```
、  
  
.e-grid .e-gridpager .e-currentitem {  
background-color: #0078d7;  
}  
、
```

### Sorting in Vue Grid component

#### *Customizing the Grid sorting icon*

Use the below CSS to customize the Grid sorting icon which present in the Grid header. You can use the available Syncfusion [icons](#) based on your theme.

```
,  
  
.e-grid .e-icon-ascending::before {  
content: '\e306';  
}  
  
.e-grid .e-icon-ascending::before {  
content: '\e304';  
}  
,
```

#### *Customizing the Grid multi sorting icon*

Use the below CSS to customize the Grid multi sorting icon which present in the Grid header. You can use the available Syncfusion [icons](#) based on your theme.

```
,  
  
.e-grid .e-sortnumber {  
background-color: #deecf9;  
font-family: cursive;  
}  
,
```

### Filtering in Vue Grid component

#### *Customizing the filterbar cell element*

Use the below CSS to customize the Grid Filterbar cell element present in the Grid header.

```
,  
  
.e-grid .e-filterbarcell {  
background-color: #deecf9;  
}  
,
```

#### *Customizing the filterbar input element*

Customize the Grid Filterbar input element using this CSS.

```
,  
  
.e-grid .e-filterbarcell .e-input-group input.e-input{  
font-family: cursive;  
}
```

,

#### *Customizing the filterbar input focus*

Use the below CSS to customize the Grid Filterbar highlight color of focused filterbar input element.

,

```
.e-grid .e-filterbarcell .e-input-group.e-input-focus::after,  
.e-grid .e-filterbarcell .e-input-group.e-input-focus::before{  
background-color: #0078d7;  
}
```

,

#### *Customizing the filterbar input clear icon*

Use the below CSS to customize the Grid Filterbar input clear icon.

,

```
.e-grid .e-filterbarcell .e-input-group .e-clear-icon::before {  
content: '\e825';  
}
```

,

#### *Customizing the Grid filtering icon*

Below CSS customizes the Grid filter icon which present in the Grid header. You can also use custom icons. Here we have used bootstrap icon for filter icon element.

,

```
.e-grid .e-icon-filter::before{  
content: '\002a';  
}
```

,

#### *Customizing the filter dialog content*

Use the below CSS to customize the Grid filter dialog content element.

,

```
.e-grid .e-filter-popup .e-dlg-content {  
background-color: #deecf9;  
}
```

,

#### *Customizing the filter dialog footer*

Grid filter dialog footer element can be customized by using the below CSS.

,

```
.e-grid .e-filter-popup .e-footer-content {
```



```
background-color: #deecf9;
}
```

#### *Customizing the filter dialog input element*

Use the below CSS to customize the Grid filter dialog input element.

```
.e-grid .e-filter-popup .e-input-group input.e-input{
font-family: cursive;
}
```

#### *Customizing the filter dialog button element*

Use the below CSS to customize the Grid filter dialog button element.

```
.e-grid .e-filter-popup .e-btn{
font-family: cursive;
}
```

#### *Customizing the excel filter dialog number filters element*

Grid Excel filter dialog number filters element can be customized using the below CSS.

```
.e-grid .e-filter-popup .e-contextmenu-wrapper ul{
background-color: #deecf9;
}
```

### Grouping in Vue Grid component

#### *Customizing the group header*

Use the below CSS to customize the group header element.

```
.e-grid .e-groupdroparea {
background-color: #deecf9;
}
```

#### *Customizing the group expand or collapse icons*

Use the below CSS to customize the Grid group expand/collapse icon. You can use the available Syncfusion [icons](#) based on your theme.

```
、  
  
.e-grid .e-icon-gdownarrow::before{  
content:'\e916'  
}  
  
.e-grid .e-icon-grightarrow::before{  
content:'\e913'  
}  
、
```

#### *Customizing the group caption row*

Below CSS customizes the Grid group caption row element.

```
、  
  
.e-grid .e-groupcaption {  
background-color: #deecf9;  
}  
  
.e-grid .e-recordplusexpand,  
.e-grid .e-recordpluscollapse {  
background-color: #deecf9;  
}  
、
```

#### *Customizing the indent cell*

Use the below CSS to customize the Grid grouping indent cell element.

```
、  
  
.e-grid .e-indentcell {  
background-color: #deecf9;  
}  
、
```

#### *Tool bar in Vue Grid component*

##### *Customizing the toolbar root element*

Use the below CSS to customize the Grid toolbar root element.

```
、  
  
.e-grid .e-toolbar-items {  
background-color: #deecf9;  
}  
、
```

*Customizing the toolbar button element*

Use the below CSS to customize the Grid toolbar button element.

`

```
.e-grid .e-toolbar .e-btn {  
background-color: #deecf9;  
}
```

`

*Editing in Vue Grid component**Customizing the edited and added row element*

Use the below CSS to customize the Grid edited and added row table elements

`

```
.e-grid .e-editedrow table,  
.e-grid .e-addedrow table {  
background-color: #deecf9;  
}
```

`

*Customizing the edited row input element*

Grid edited row input elements can be customized using the below CSS.

`

```
.e-grid .e-gridform .e-rowcell .e-input-group .e-input.e-field {  
font-family: cursive;  
}
```

`

*Customizing the edit dialog header element*

Use the below CSS to customize the Grid edit dialog header element

`

```
.e-edit-dialog .e-dlg-header-content {  
background-color: #deecf9;  
}
```

`

*Customizing the edited row input element in dialog edit mode*

Below CSS customizes the Grid edited row input element in Dialog edit mode

`

```
.e-gridform .e-rowcell .e-float-input .e-field {  
font-family: cursive;
```

```
}
,
```

























### *Customizing the command column buttons*

Use the below CSS to customize the command column buttons such as edit, delete, update, and cancel.

```
,
```

```
.e-grid .e-edit::before, .e-grid .e-delete::before {
color: #ff8787;
}

.e-grid .e-update::before, .e-grid .e-cancel-icon::before {
color: #0078d7;
}
,
```

Order ID	Customer ID	Freight	Ship Country	Manage Records
10248	VINET	\$2.38	France	 
10249	TOMSP	\$11.61	Germany	 
10250	HANAR	\$65.83	Brazil	 
10251	VICTE	\$41.34	France	 
10252	SUPRD	\$51.30	Belgium	 
10253	HANAR	\$58.17	Brazil	 
10254	CHOPS	\$22.98	Switzerland	 
10255	RICSU	\$148.33	Switzerland	 
10256	WELLI	\$13.97	Brazil	 
10257	HILAA	\$81.91	Venezuela	 
10258	ERNSH	\$140.51	Austria	 
10259	CENTC	\$3.25	Mexico	 
<div> <span>&lt;</span> <span>&lt;</span> <span>1</span> <span>2</span> <span>3</span> <span>4</span> <span>5</span> <span>6</span> <span>7</span> <span>8</span> <span>9</span> <span>10</span> <span>...</span> <span>&gt;</span> <span>&gt;</span> </div> <div>1 of 17 pages (200 items)</div>				

### Aggregate in Vue Grid component

#### *Customizing the aggregate root element*

Use the below CSS to customize the Grid aggregate root elements

```
,
```

```
.e-grid .e-gridfooter {
background-color: #deecf9;
}
,
```

,

#### *Customizing the aggregate cell elements*

Using this CSS, you can customize the Grid summary row cell elements.

,

```
.e-grid .e-summaryrow .e-summarycell {  
background-color: #deecf9;  
}
```

,

#### *Selection in Vue Grid component*

##### *Customizing the row selection background*

Row selection background can be customized using below CSS.

,

```
.e-grid td.e-selectionbackground {  
background-color: #deecf9;  
}
```

,

##### *Customizing the cell selection background*

Cell selection background can be customized using below CSS.

,

```
.e-grid td.e-cellselectionbackground {  
background-color: #deecf9;  
}
```

,

##### *Customizing the column selection background*

Column selection background can be customized using below CSS.

,

```
.e-grid .e-columnselection {  
background-color: #deecf9;  
}
```

,

#### *How To*

Register the grid using vue component in Vue Grid component

Import the `GridComponent` from the `@syncfuion/ej2-vue-grids` package, register the same using the `Vue.component()` with name of the grid selector and the `GridComponent` as its arguments.

Refer to the code example given below.

```
`ts
import { GridComponent } from '@syncfusion/ej2-vue-grids';
Vue.component('ejs-grid', GridComponent);
`
```

Using `Vue.component` will register the grid component alone. Child directives such as `Columns` and `Aggregates` need to be registered separately.

Refer to the code example given below to register column directives

```
`ts
import { ColumnsDirective, ColumnDirective } from '@syncfusion/ej2-vue-grids';
Vue.component('e-columns', ColumnsDirective);
Vue.component('e-column', ColumnDirective);
`
```

Refer to the code example given below to register aggregates directives

```
`ts
import { AggregatesDirective, AggregateDirective, AggregateColumnsDirective,
AggregateColumnDirective } from '@syncfusion/ej2-vue-grids';
Vue.component('e-aggregates', AggregatesDirective);
Vue.component('e-aggregate', AggregateDirective);
Vue.component('e-columns', AggregateColumnsDirective);
Vue.component('e-column', AggregateColumnDirective);
`
```

#### [Enable/disable grid and its actions in Vue Grid component](#)

You can enable/disable the Grid and its actions by applying/removing corresponding CSS styles.

To enable/disable the grid and its actions, follow the given steps:

**Step 1:** Create CSS class with custom style to override the default style of Grid.

```
`
.disablegrid {
pointer-events: none;
opacity: 0.4;
}
.wrapper {
cursor: not-allowed;
}
`
```

**Step 2:** Add/Remove the CSS class to the Grid in the click event handler of Button.

```
`ts
btnClick(args) {
  if (this.$refs.Grid.$el.classList.contains('disablegrid')) {
    this.$refs.Grid.$el.classList.remove('disablegrid');
    document.getElementById("GridParent").classList.remove('wrapper');
  }
  else {
    this.$refs.Grid.$el.classList.add('disablegrid');
    document.getElementById("GridParent").classList.add('wrapper');
  }
}
```

In the below demo, the button click will enable/disable the Grid and its actions.

#### **APP.VUE**

```
<template>
  <div id="app">
    <div>
      <ejs-button iconCss="e-icons e-play-icon" cssClass="e-flat"
      :isPrimary="true" :isToggle="true" v-
      on:click.native="btnClick">Enable/Disable Grid</ejs-button>
      <div id="GridParent">
        <ejs-grid ref='Grid' :dataSource='data'
        :editSettings='editSettings' :toolbar='toolbar' height='273px'>
          <e-columns>
            <e-column field='OrderID' headerText='Order ID'
            textAlign='Right' :isPrimaryKey='true' width=100></e-column>
            <e-column field='CustomerID' headerText='Customer ID'
            width=120></e-column>
            <e-column field='Freight' headerText='Freight'
            textAlign= 'Right' editType='numericedit' width=120 format= 'C2'></e-
            column>
            <e-column field='ShipCountry' headerText='Ship
            Country' editType= 'dropdownedit' width=150></e-column>
          </e-columns>
        </ejs-grid>
      </div>
    </div>
  </div>
</template>
<script>
import Vue from "vue";
import { GridPlugin, Page, Toolbar, Edit } from "@syncfusion/ej2-vue-grids";
import { ButtonPlugin } from "@syncfusion/ej2-vue-buttons";
import { data } from './datasource.js';
Vue.use(GridPlugin);
```

```

Vue.use(ButtonPlugin);
export default {
  data() {
    return {
      data: data,
      editSettings: { allowAdding:true, allowDeleting:true, allowEditing:
true },
      toolbar: ['Add', 'Edit', 'Delete', 'Update', 'Cancel']
    };
  },
  methods: {
    btnClick(args) {
      if (this.$refs.Grid.$el.classList.contains('disablegrid')) {
        this.$refs.Grid.$el.classList.remove('disablegrid');
        document.getElementById("GridParent").classList.remove('wrapper');
      }
      else {
        this.$refs.Grid.$el.classList.add('disablegrid');
        document.getElementById("GridParent").classList.add('wrapper');
      }
    }
  },
  provide: {
    grid: [Page, Edit, Toolbar]
  }
}
</script>
<style>
  @import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
  .disablegrid {
    pointer-events: none;
    opacity: 0.4;
  }
  .wrapper {
    cursor: not-allowed;
  }
</style>

```

{% previewsample "page.domainurl/code-snippet/grid/edit/default-cs45" %}

### Hide sorting in excel filter in Vue Grid component

You can hide the sorting options on the excel filter dialog by setting display as none for the following classes.

```

、

.e-excel-ascending,
.e-excel-descending,
.e-separator.e-excel-separator {
display: none;
}
、

```



**APP.VUE**

```

<template>
  <div id="app">
    <ejs-grid ref='grid' :dataSource='data' allowFiltering='true'
    :filterSettings='filterOptions' height='273px' >
      <e-columns>
        <e-column field='OrderID' headerText='Order ID'
        textAlign='Right' width=100></e-column>
        <e-column field='CustomerID' headerText='Customer ID'
        width=120></e-column>
        <e-column field='ShipCity' headerText='Ship City'
        width=100></e-column>
        <e-column field='ShipName' headerText='Ship Name'
        width=100></e-column>
      </e-columns>
    </ejs-grid>
  </div>
</template>
<script>
import Vue from "vue";
import { GridPlugin, Filter } from "@syncfusion/ej2-vue-grids";
import { data } from './datasource.js';
Vue.use(GridPlugin);
export default {
  data: () => {
    return {
      data: data,
      filterOptions: {
        type: 'Excel'
      },
    };
  },
  provide: {
    grid: [Filter],
  },
}
</script>
<style>
  @import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
  .e-excel-ascending,
  .e-excel-descending,
  .e-separator.e-excel-separator {
    display: none;
  }
</style>

```

{% previewsample "page.domainurl/code-snippet/grid/how-to/default-cs21" %}

### Cascading drop down list with grid editing in Vue Grid component

You can achieve the Cascading DropDownList with grid Editing by using the Cell Edit Template feature.

In the below demo, Cascading DropDownList rendered for **ShipCountry** and **ShipState** column.

**APP.VUE**

```

<template>

```

```

    <div id="app">
      <ejs-grid ref='grid' :dataSource='data' :editSettings='editSettings'
:toolbar='toolbar' height='273px' >
        <e-columns>
          <e-column field='OrderID' headerText='Order ID'
:isPrimaryKey='true' textAlign='Right' width=100></e-column>
          <e-column field='CustomerID' headerText='Customer ID'
width=120></e-column>
          <e-column field='ShipCountry' headerText='ShipCountry'
editType='dropdownedit' :edit='countryParams' width=150></e-column>
          <e-column field='ShipCity' headerText='Ship City'
editType='dropdownedit' :edit='stateParams' width=150></e-column>
        </e-columns>
      </ejs-grid>
    </div>
  </template>
  <script>
import Vue from "vue";
import { createElement } from '@syncfusion/ej2-base';
import { GridPlugin, Toolbar, Edit } from "@syncfusion/ej2-vue-grids";
import { DropDownList } from "@syncfusion/ej2-dropdowns";
import { Query } from '@syncfusion/ej2-data';
import { cascadeData } from './datasource.js';
let country= [
    { countryName: 'United States', countryId: '1' },
    { countryName: 'Australia', countryId: '2' }
];
let state = [
    { stateName: 'New York', countryId: '1', stateId: '101' },
    { stateName: 'Virginia ', countryId: '1', stateId: '102' },
    { stateName: 'Washington', countryId: '1', stateId: '103' },
    { stateName: 'Queensland', countryId: '2', stateId: '104' },
    { stateName: 'Tasmania ', countryId: '2', stateId: '105' },
    { stateName: 'Victoria', countryId: '2', stateId: '106' }
];
let countryElem, stateElem, countryObj, stateObj;
Vue.use(GridPlugin);
export default {
  data: () => {
    return {
      data: cascadeData,
      toolbar: ['Add', 'Edit', 'Delete', 'Update', 'Cancel'],
      editSettings: { allowEditing: true, allowAdding: true,
allowDeleting: true },
      countryParams: {
        create: () => {
          countryElem = document.createElement('input');
          return countryElem;
        },
        read: () => {
          return countryObj.text;
        },
        destroy: () => {
          countryObj.destroy();
        },
        write: () => {
          countryObj = new DropDownList({

```

```

        dataSource: country,
        fields: { value: 'countryId', text: 'countryName' },
        change: () => {
            stateObj.enabled = true;
            let tempQuery = new Query().where('countryId', 'equal',
countryObj.value);
            stateObj.query = tempQuery;
            stateObj.text = null;
            stateObj.dataBind();
        },
        placeholder: 'Select a country',
        floatLabelType: 'Never'
    });
    countryObj.appendTo(countryElem);
}
},
stateParams: {
    create: () => {
        stateElem = document.createElement('input');
        return stateElem;
    },
    read: () => {
        return stateObj.text;
    },
    destroy: () => {
        stateObj.destroy();
    },
    write: () => {
        stateObj = new DropDownList({
            dataSource: state,
            fields: { value: 'stateId', text: 'stateName' },
            enabled: false,
            placeholder: 'Select a state',
            floatLabelType: 'Never'
        });
        stateObj.appendTo(stateElem);
    }
},
};
},
provide: {
    grid: [Edit, Toolbar]
}
}
</script>
<style>
    @import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/grid/how-to/default-cs20" %}

### Perform grid actions by keyboard short cut keys in Vue Grid component

Using keyboard shortcuts, Grid performs navigation and actions.

In addition, You can also perform grid actions with custom keyboard shortcuts. This operation has to be achieved outside of the grid with the help of `keydown` event.

The following example demonstrates on **Adding** a new row when **Enter** key is pressed in the grid.

#### APP.VUE

```
<template>
  <div id="app">
    <ejs-grid ref='grid' :dataSource='data' :editSettings='editSettings'
:toolbar='toolbar' height='273px' :load = 'load'>
      <e-columns>
        <e-column field='OrderID' headerText='Order ID'
textAlign='Right' :isPrimaryKey='true' width=100></e-column>
        <e-column field='CustomerID' :visible = 'false'
headerText='Customer ID' width=120></e-column>
        <e-column field='Freight' headerText='Freight' textAlign=
'Right' editType= 'numericedit' width=120 format= 'C2'></e-column>
        <e-column field='ShipCountry' headerText='Ship Country'
editType= 'dropdownedit' width=150></e-column>
      </e-columns>
    </ejs-grid>
  </div>
</template>
<script>
import Vue from "vue";
import { GridPlugin, Page, Toolbar, Edit } from "@syncfusion/ej2-vue-grids";
import { data } from './datasource.js';
Vue.use(GridPlugin);
export default {
  data() {
    return {
      data: data,
      editSettings: { allowEditing: true, allowAdding: true, allowDeleting:
true, mode: 'Normal' },
      toolbar: ['Add', 'Edit', 'Delete', 'Update', 'Cancel']
    };
  },
  methods: {
    load(args) {
      this.$refs.grid.$el.addEventListener('keydown', this.keyDownHandler);
    }
    keyDownHandler(e: any) {
      if(e.keyCode) {
        let gridIns = this.$refs.grid.ej2Instances;
        gridIns.addRecord();
      }
    }
  },
  provide: {
    grid: [Page, Edit, Toolbar]
  }
}
</script>
<style>
  @import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
</style>
```

{% previewsample "page.domainurl/code-snippet/grid/edit/default-cs47" %}

## Exporting grid in cordova application in Vue Grid component

Cordova application does not support direct file download. So we have to use the Blob stream to export the Grid. You can use corresponding exporting methods and exportComplete events to get the Blob stream.

### APP.VUE

```
<template>
  <div id="app">
    <ejs-grid ref='grid' :dataSource='data' :toolbar='toolbarOptions'
height='272px' :allowExcelExport='true'
:excelExportComplete='excelExpComplete'
:pdfExportComplete='pdfExportComplete' :allowPdfExport='true'
:toolbarClick='toolbarClick' >
      <e-columns>
        <e-column field='OrderID' headerText='Order ID'
textAlign='Right' width=100></e-column>
        <e-column field='CustomerID' headerText='Customer ID'
width=150></e-column>
        <e-column field='ShipCity' headerText='Ship City'
width=150></e-column>
      </e-columns>
    </ejs-grid>
  </div>
</template>
<script>
import Vue from "vue";
import { GridPlugin, Toolbar, PdfExport, ExcelExport } from "@syncfusion/ej2-
vue-grids";
import { data } from './datasource.js';
Vue.use(GridPlugin);
export default {
  data: () => {
    return {
      data: data,
      toolbarOptions: ['Add', 'PdfExport', 'ExcelExport'],
    };
  },
  methods: {
    toolbarClick: function (args) {
      if (args['item'].id.indexOf("pdfexport") !== -1) {
        this.$refs.grid.pdfExport(null, null, null, true);
      }
      if (args['item'].id.indexOf("excelexport") !== -1) {
        this.$refs.grid.excelExport(null, null, null, true);
      }
    },
    excelExpComplete: function (args) {
      //This event will be triggered when excel exporting.
      args.promise.then((e) => {
        //In this `then` function, we can get blob data through the
arguments after promise resolved.
        this.exportBlob(e.blobData);
      });
    },
    pdfExportComplete: function (args) {
      //This event will be triggered when pdf exporting.
    }
  }
}
```

```

        args.promise.then((e) => {
            //In this `then` function, we can get blob data through the
            arguments after promise resolved.
            this.exportBlob(e.blobData);
        });
    },
    exportBlob: function (blob) {
        let a = document.createElement('a');
        document.body.appendChild(a);
        a.style.display = 'none';
        let url = window.URL.createObjectURL(blob);
        a.href = url;
        a.download = 'Export';
        a.click();
        window.URL.revokeObjectURL(url);
        document.body.removeChild(a);
    }
},
provide: {
    grid: [Toolbar, ExcelExport, PdfExport]
},
}
</script>
<style>
    @import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/grid/how-to/foreignKey-cs8" %}

### Customize pager drop down in Vue Grid component

To customize default values of pager dropdown, you need to define `pageSizes` as array of strings.

#### APP.VUE

```

<template>
    <div id="app">
        <ejs-grid :dataSource='data' :allowPaging='true'
        :pageSettings='pageSettings' height='280px' >
            <e-columns>
                <e-column field='OrderID' headerText='Order ID'
                :isPrimaryKey='true' textAlign='Right' width=100></e-column>
                <e-column field='CustomerID' headerText='Customer ID'
                width=120></e-column>
                <e-column field='Freight' headerText='Freight'
                textAlign='Center' format='C2' width=80></e-column>
                <e-column field='ShipCountry' headerText='Ship Country'
                width=120></e-column>
            </e-columns>
        </ejs-grid>
    </div>
</template>
<script>
import Vue from "vue";
import { GridPlugin, Toolbar, Page } from "@syncfusion/ej2-vue-grids";
import { data } from '../datasource.js';
Vue.use(GridPlugin);

```

```

export default {
  data: () => {
    return {
      data: data,
      pageSettings: {pageSizes: ['5', '10', 'All']}
    };
  },
  provide: {
    grid: [Page]
  }
}
</script>
<style>
  @import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/grid/how-to/pagerdropdown-cs2" %}

### Hide the expand/collapse icon in parent row in Vue Grid component

By default, the expand/collapse icon will be visible even if the child grid is empty.

You can use [rowDataBound](#) event to hide the icon when there are no records in child grid.

To hide the expand/collapse icon in parent row when no records in child grid, follow the given steps:

**Step 1:** Create CSS class with custom style to override the default style of Grid.

```

.e-row[aria-selected="true"] .e-customizedExpandcell {
background-color: #e0e0e0;
}
.e-grid.e-gridhover tr[role='row']:hover {
background-color: #eee;
}

```

**Step 2:** Add the CSS class to the Grid in the [rowDataBound](#) event handler of Grid.

```

`ts
rowDataBound(args){
let filter:string = args.data.EmployeeID;

let childrecord: any = new DataManager(this.$refs.Grid.childGrid.dataSource).executeLocal(new
Query().where("EmployeeID", "equal", parseInt(filter), true));

if(childrecord.length == 0) {
//here hide which parent row has no child records
args.row.querySelector('td').innerHTML=" ";
args.row.querySelector('td').className = "e-customizedExpandcell";
}
}

```

```

}
}
,

```

In the below demo, the expand/collapse icon in the row with **EmployeeID** as **1** is hidden as it does not have record in child Grid.

### APP.VUE

```

<template>
  <div id="app">
    <ejs-grid ref='Grid' :dataSource="data" height='315px'
:childGrid='childGrid' :rowDataBound='rowDataBound'>
      <e-columns>
        <e-column field='OrderID' headerText='Order ID'
textAlign='Right' width=90></e-column>
        <e-column field='EmployeeID' headerText='Customer ID'
width=120></e-column>
        <e-column field='Freight' headerText='Freight' textAlign='Right'
format='C2' width=90></e-column>
        <e-column field='OrderDate' headerText='Order Date'
textAlign='Right' format='yMd' type='date' width=120></e-column>
      </e-columns>
    </ejs-grid>
  </div>
</template>
<script>
import Vue from "vue";
import { GridPlugin, Page, DetailRow } from "@syncfusion/ej2-vue-grids";
import { DataManager, Query } from '@syncfusion/ej2-data';
import { employeeData } from './datasource.js';
let dataManger: Object = [{Order:100, ShipName:'Berlin', EmployeeID:2},
                           {Order:101, ShipName:'Capte', EmployeeID:3},
                           {Order:102, ShipName:'Marlon', EmployeeID:4},
                           {Order:103, ShipName:'Black pearl', EmployeeID:5},
                           {Order:104, ShipName:'Pearl', EmployeeID:6},
                           {Order:105, ShipName:'Noth bay', EmployeeID:7},
                           {Order:106, ShipName:'baratna', EmployeeID:8},
                           {Order:107, ShipName:'Charge', EmployeeID:9}];

Vue.use(GridPlugin);
export default {
  data() {
    return {
      data: employeeData,
      childGrid: {
        dataSource: dataManger,
        queryString: 'EmployeeID',
        columns: [
          { field: 'Order', headerText: 'Order ID', textAlign: 'Right',
width: 120 },
          { field: 'EmployeeID', headerText: 'Employee ID', width: 150 },
          { field: 'ShipName', headerText: 'Ship Name', width: 150 }
        ]
      }
    };
  },
},

```



```

methods: {
  rowDataBound(args) {
    let filter:string = args.data.EmployeeID;
    let childrecord: any = new
DataManager(this.$refs.Grid.childGrid.dataSource).executeLocal(new
Query().where("EmployeeID", "equal", parseInt(filter), true));
    if(childrecord.length == 0) {
      //here hide which parent row has no child records
      args.row.querySelector('td').innerHTML=" ";
      args.row.querySelector('td').className = "e-customizedExpandcell";
    }
  },
  provide: {
    grid: [DetailRow]
  }
}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
.e-row[aria-selected="true"] .e-customizedExpandcell {
  background-color: #e0e0e0;
}
.e-grid.e-gridhover tr[role='row']:hover {
  background-color: #eee;
}
</style>

```

{% previewsample "page.domainurl/code-snippet/grid/edit/default-cs46" %}

### Complex column as foreign key column in Vue Grid component

The following example shows how to set the complex column as foreign key column.

In the following example, `Employee.EmployeeID` is a complex column and also declared as a foreign column which shows `FirstName` column from foreign data.

#### APP.VUE

```

<template>
  <div id="app">
    <ejs-grid :dataSource='data' height='315'>
      <e-columns>
        <e-column field='OrderID' headerText='Order ID'
textAlign='Right' width=100></e-column>
        <e-column field='Employee.EmployeeID' headerText='Employee
Name' width=120 foreignKeyValue='FirstName' foreignKeyField='EmployeeID'
:dataSource='employeeData'></e-column>
        <e-column field='Freight' headerText='Freight'
textAlign='Right' width=80></e-column>
        <e-column field='ShipCity' headerText='Ship City' width=130
></e-column>
      </e-columns>
    </ejs-grid>
  </div>
</template>
<script>

```

```

import Vue from "vue";
import { GridPlugin, ForeignKey } from "@syncfusion/ej2-vue-grids";
import { data, employeeData } from './datasource.js';
Vue.use(GridPlugin);
export default {
  data() {
    return {
      data: data.slice(0, 5),
      employeeData: employeeData.slice(0, 5)
    };
  },
  provide: {
    grid: [ForeignKey]
  }
}
</script>
<style>
  @import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/grid/column/foreigncolumn-cs4" %}

\* For remote data, the sorting and grouping is done based on [column.foreignKeyField](#) instead of [column.foreignKeyValue](#).

\* If [column.foreignKeyField](#) is not defined, then the column uses [column.field](#).

List of array of objects in Vue Grid component

The following example shows how to set Complex field for datasource having Array Of Objects.

#### APP.VUE

```

<template>
  <div id="app">
    <ejs-grid :dataSource='data' :height='315'>
      <e-columns>
        <e-column field='EmployeeID' headerText='Employee ID'
textAlign='Right' width=120></e-column>
        <e-column field='Names.0.FirstName' headerText='First Name'
width=120></e-column>
        <e-column field='Names.0.LastName' headerText='Last Name'
width=120></e-column>
        <e-column field='Title' headerText='Title' width=150></e-
column>
      </e-columns>
    </ejs-grid>
  </div>
</template>
<script>
import Vue from "vue";
import { GridPlugin } from "@syncfusion/ej2-vue-grids";
import { data } from './datasource.js';
Vue.use(GridPlugin);
export default {
  data() {
    return {
      data: data.slice(0, 5)
    }
  }
}

```

```

    };
  }
}
</script>
<style>
  @import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/grid/column/complex-cs2" %}

### Add params for filtering in Vue Grid component

You can customize the default settings of the components which are used in Menu filter by using params of filter property in column definition.

In the below sample, OrderID and Freight Columns are numeric columns, while open the filter dialog then you can see that NumericTextBox with spin button is displayed to change/set the filter value. Now using the params option we hide the spin button in NumericTextBox for OrderID Column.

### APP.VUE

```

<template>
  <div id="app">
    <ejs-grid :dataSource='data' :allowFiltering='true'
    :filterSettings='filterOptions' height='273px'>
      <e-columns>
        <e-column field='OrderID' headerText='Order ID'
        textAlign='Right' :filter='filterParams' width=100></e-column>
        <e-column field='CustomerID' headerText='Customer ID'
        width=120></e-column>
        <e-column field='Freight' headerText='Freight' textAlign=
        'Right' width=120 format= 'C2'></e-column>
        <e-column field='ShipCountry' headerText='Ship Country'
        width=150></e-column>
      </e-columns>
    </ejs-grid>
  </div>
</template>
<script>
import Vue from "vue";
import { GridPlugin, Filter } from "@syncfusion/ej2-vue-grids";
import { data } from './datasource.js'
Vue.use(GridPlugin);
export default {
  data() {
    return {
      data: data,
      filterOptions: {
        type: 'Menu'
      },
      filterParams: { params: { showSpinButton: false } }
    };
  },
  provide: {
    grid: [Filter]
  }
}

```

```

</script>
<style>
  @import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/grid/filter/default-cs12" %}

### Select grid rows based on certain condition in Vue Grid component

You can select the specific row in the grid based on a certain condition by using the [selectRows](#) method in the [dataBound](#) event of Grid.

In the below demo, we have selected the grid rows only when **EmployeeID** column value greater than 3.

#### APP.VUE

```

<template>
  <div id="app">
    <ejs-grid ref='Grid' :dataSource='data' :allowPaging='true'
height='280px' :rowDataBound='rowDataBound'
:selectionSettings='selectionOptions' :dataBound='dataBound'>
      <e-columns>
        <e-column field='OrderID' headerText='Order ID'
:isPrimaryKey='true' textAlign='Right' width=100></e-column>
        <e-column field='CustomerID' headerText='Customer ID'
width=120></e-column>
        <e-column field='EmployeeID' headerText='Employee ID'
textAlign='Right' width=120></e-column>
        <e-column field='ShipCity' headerText='Ship City'
width=80></e-column>
        <e-column field='ShipCountry' headerText='Ship Country'
width=120></e-column>
      </e-columns>
    </ejs-grid>
  </div>
</template>
<script>
import Vue from "vue";
import { GridPlugin, Toolbar, Page } from "@syncfusion/ej2-vue-grids";
import { employeeData } from '../datasource.js';
Vue.use(GridPlugin);
export default {
  data: () => {
    return {
      data: employeeData,
      selectionOptions: { type: 'Multiple' },
      selIndex: []
    };
  },
  methods: {
    rowDataBound(args) {
      if (args.data['EmployeeID'] > 3) {
        this.selIndex.push(parseInt(args.row.getAttribute('aria-
rowindex')));
      }
    }
  }
}

```

```

    dataBound(args) {
        if (this.selIndex.length) {
            this.$refs.Grid.selectRows(this.selIndex);
            this.selIndex = [];
        }
    },
    provide: {
        grid: [Page]
    }
}
</script>
<style>
    @import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/grid/edit/default-cs48" %}

### Collapse grouped rows at initial render in Vue Grid component

You can collapse all the grouped rows at initial rendering by using `dataBound` event with `collapseAll` method of the grid.

In the below demo, all the grouped rows are collapsed at initial rendering.

#### APP.VUE

```

<template>
    <div id="app">
        <ejs-grid ref='grid' :dataSource='data' :allowGrouping='true'
        :groupSettings='groupOptions' :dataBound='dataBound' height='267px'>
            <e-columns>
                <e-column field='OrderID' headerText='Order ID'
                textAlign='Right' width=120></e-column>
                <e-column field='CustomerID' headerText='Customer ID'
                width=150></e-column>
                <e-column field='ShipCity' headerText='Ship City'
                width=150></e-column>
                <e-column field='ShipName' headerText='Ship Name'
                width=150></e-column>
            </e-columns>
        </ejs-grid>
    </div>
</template>
<script>
import Vue from "vue";
import { GridPlugin, Group } from "@syncfusion/ej2-vue-grids";
import { data } from './datasource.js';
Vue.use(GridPlugin);
var initial = true;
export default {
    data() {
        return {
            data: data,
            groupOptions: { columns: ['ShipCity'] }
        };
    },
}

```

```

methods: {
  dataBound: function() {
    if(initial === true) {
      this.$refs.grid.ej2Instances.groupModule.collapseAll();
      initial = false;
    }
  },
  provide: {
    grid: [Group]
  }
}
</script>
<style>
  @import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/grid/group/default-cs10" %}

### Grouped row page size in Vue Grid component

By default, we have displayed the no of records based on the `pageSize`. If you want to show grouped column rows based on the `pageSize` then we suggest you to use the below way.

In the below sample, we have overridden the default `generateQuery` to display the grouped rows instead of grid rows based on the `pageSize`.

### APP.VUE

```

<template>
  <div id="app">
    <ejs-grid :dataSource='data' :allowGrouping='true'
    :groupSettings='groupOptions' :allowPaging='true'
    :pageSettings='pageOptions' height='273px'>
      <e-columns>
        <e-column field='OrderID' headerText='Order ID'
        textAlign='Right' width=100></e-column>
        <e-column field='CustomerID' headerText='Customer ID'
        width=120></e-column>
        <e-column field='Freight' headerText='Freight' textAlign=
        'Right' width=120 format= 'C2'></e-column>
        <e-column field='ShipCountry' headerText='Ship Country'
        width=150></e-column>
      </e-columns>
    </ejs-grid>
  </div>
</template>
<script>
import Vue from "vue";
import { GridPlugin, Group, Page, Data } from "@syncfusion/ej2-vue-grids";
import { data } from './datasource.js'
Vue.use(GridPlugin);
let old = Data.prototype.generateQuery;
Data.prototype.generateQuery = function() {
  let query = old.call(this, true);
  this.pageQuery(query);
  return query;
}

```

```

};
export default {
  data() {
    return {
      data: data,
      pageOptions: { pageSize: 5 },
      groupOptions: { columns: ["ShipCountry"] }
    };
  },
  provide: {
    grid: [Group, Page]
  }
}
</script>
<style>
  @import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/grid/filter/default-cs14" %}

### Get row cell index in Vue Grid component

You can get the specific row and cell index of the grid by using `rowSelected` event of the grid. Here, we can get the row and cell index by using `aria-rowindex`(get row Index from `tr` element) and `aria-colindex`(column index from `td` element) attribute.

### APP.VUE

```

<template>
  <div id="app">
    <ejs-grid ref='grid' :dataSource='data' :rowSelected='rowSelected'
height='267px'>
      <e-columns>
        <e-column field='OrderID' headerText='Order ID'
textAlign='Right' width=120></e-column>
        <e-column field='CustomerID' headerText='Customer ID'
width=150></e-column>
        <e-column field='ShipCity' headerText='Ship City'
width=150></e-column>
        <e-column field='ShipName' headerText='Ship Name'
width=150></e-column>
      </e-columns>
    </ejs-grid>
  </div>
</template>
<script>
import Vue from "vue";
import { GridPlugin, Group } from "@syncfusion/ej2-vue-grids";
import { data } from './datasource.js';
Vue.use(GridPlugin);
export default {
  data() {
    return {
      data: data
    };
  },
  methods: {

```

```

    rowSelected: function(args) {
      alert("row index: "+args.row.getAttribute('aria-rowindex'));
      alert("column index: "+args.target.getAttribute('aria-colindex'));
    }
  }
}
</script>
<style>
  @import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/grid/group/default-cs11" %}

### Display null values at bottom in Vue Grid component

By default the null values are displayed at bottom of the Grid row while perform sorting in ascending order. As well as this values are displayed at top of the Grid row while perform sorting with descending order. But you can customize this default order to display the null values at always bottom row of the Grid by using [column.sortComparer](#) method.

In the below demo we have displayed the null date values at bottom of the Grid row while sorting the **OrderDate** column in both ways.

### APP.VUE

```

<template>
  <div id="app">
    <ejs-grid :dataSource='data' :allowSorting='true'
    :actionBegin='actionBegin'>
      <e-columns>
        <e-column field='OrderID' headerText='Order ID'
width=100></e-column>
        <e-column field='CustomerID' headerText='Customer ID'
width=120></e-column>
        <e-column field='OrderDate' headerText='Order Date'
format='yMd' :sortComparer='sortComparer' width=120></e-column>
        <e-column field='ShipCountry' headerText='Ship Country'
width=150></e-column>
      </e-columns>
    </ejs-grid>
  </div>
</template>
<script>
import Vue from "vue";
import { GridPlugin, Sort } from "@syncfusion/ej2-vue-grids";
import { data } from './datasource.js';
Vue.use(GridPlugin);
let action;
export default {
  data: () => {
    return {
      data: data
    };
  },
  methods: {
    sortComparer: function(reference, comparer) {
      let sortAsc = action === "Ascending" ? true : false;

```



```

        if (sortAsc && reference === null) {
            return 1;
        }
        else if (sortAsc && comparer === null) {
            return -1;
        }
        else if (!sortAsc && reference === null) {
            return -1;
        }
        else if (!sortAsc && comparer === null) {
            return 1;
        }
        else {
            return reference - comparer;
        }
    },
    actionBegin: function(args) {
        if (args.requestType == "sorting") {
            action = args.direction;
        }
    }
},
provide: {
    grid: [Sort]
}
}
</script>
<style>
    @import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/grid/how-to/null-date-value-cs1" %}

Enable editing in single click in Vue Grid component

#### Normal Editing

You can make a row editable on a single click with **Normal** mode of editing in Grid, by using the [startEdit](#) and [endEdit](#) methods.

Bind the **mouseup** event for Grid and in the event handler call the [startEdit](#) and [endEdit](#), based on the clicked target element.

#### APP.VUE

```

<template>
    <div id="app">
        <ejs-grid ref='grid' id='grid' :dataSource='data' :load='load'
        :editSettings='editSettings' :toolbar='toolbar' :allowPaging="true">
            <e-columns>
                <e-column field='OrderID' headerText='Order ID'
                textAlign='Right' width=100 :isPrimaryKey='true'></e-column>
                <e-column field='CustomerID' headerText='Customer ID'
                width=120></e-column>
                <e-column field='Freight' headerText='Freight'
                textAlign='Right' width=120 format='C2'></e-column>
                <e-column field='ShipCountry' headerText='Ship Country'
                width=150></e-column>
            </e-columns>
        </ejs-grid>
    </div>
</template>

```

```

        </ejs-grid>
    </div>
</template>
<script>
import Vue from "vue";
import { GridPlugin, Edit } from "@syncfusion/ej2-vue-grids";
import { data } from './datasource.js';
Vue.use(GridPlugin);
export default {
  data: function() {
    return {
      data: data,
      editSettings: { allowEditing: true, allowAdding: true, allowDeleting:
true, mode: 'Normal' },
      toolbar: ['Add', 'Edit', 'Delete', 'Update', 'Cancel']
    };
  },
  methods: {
    load: function() {
      this.$refs.grid.ej2Instances.element.addEventListener('mouseup',
function(e) {
        var instance = this.ej2_instances[0];
        if (e.target.classList.contains("e-rowcell")) {
          if (instance.isEdit)
            instance.endEdit();
          let index = parseInt(e.target.getAttribute("Index"));
          instance.selectRow(index);
          instance.startEdit();
        }
      });
    },
    provide: {
      grid: [Edit]
    }
  }
}
</script>
<style>
  @import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/grid/edit/single-click-cs1" %}

#### Open dropdown edit popup on single click

You can open the default dropdown edit popup with single click edit by focusing the dropdown element and calling the EJ2 dropdown list's [showPopup](#) method in the Grid's [actionComplete](#) event. In this demo we have used a global flag variable in the **mouseup** event to ensure if the dropdown column is the clicked edit target.

#### APP.VUE

```

<template>
  <div id="app">
    <ejs-grid ref='grid' id='grid' :dataSource='data' :load='load'
:actionComplete='onActionComplete' :editSettings='editSettings'
:toolbar='toolbar' :allowPaging="true">

```

```

        <e-columns>
            <e-column field='OrderID' headerText='Order ID'
textAlign='Right' width=100 :isPrimaryKey='true'></e-column>
            <e-column field='CustomerID' headerText='Customer ID'
width=120></e-column>
            <e-column field='Freight' headerText='Freight'
textAlign='Right' width=120 format='C2'></e-column>
            <e-column field='ShipCountry' headerText='Ship Country'
editType='dropdownedit' width=150></e-column>
        </e-columns>
    </ejs-grid>
</div>
</template>
<script>
import Vue from "vue";
import { GridPlugin, Edit } from "@syncfusion/ej2-vue-grids";
import { data } from './datasource.js';
Vue.use(GridPlugin);
export default {
    data: function() {
        return {
            data: data,
            editSettings: { allowEditing: true, allowAdding: true, allowDeleting:
true, mode: 'Normal' },
            toolbar: ['Add', 'Edit', 'Delete', 'Update', 'Cancel'],
            isDropdown: false
        };
    },
    methods: {
        load: function() {
            this.$refs.grid.ej2Instances.element.addEventListener('mouseup',
function(e) {
                if (e.target.classList.contains("e-rowcell")) {
                    if (this.$refs.grid.ej2Instances.isEdit)
                        this.$refs.grid.ej2Instances.endEdit();
                    let rowInfo =
this.$refs.grid.ej2Instances.getRowInfo(e.target);
                    if (rowInfo.column.field === "ShipCountry")
                        this.isDropdown = true;
                    this.$refs.grid.ej2Instances.selectRow(rowInfo.rowIndex);
                    this.$refs.grid.ej2Instances.startEdit();
                }.bind(this));
        },
        onActionComplete: function(args) {
            if (args.requestType === "beginEdit" && this.isDropdown) {
                this.isDropdown = false;
                let dropdownObj = args.form.querySelector('.e-
dropdownlist').ej2_instances[0];
                dropdownObj.element.focus();
                dropdownObj.showPopup();
            }
        }
    },
    provide: {
        grid: [Edit]
    }
}

```

```

</script>
<style>
  @import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/grid/edit/single-click-cs2" %}

### Render date picker in frozen columns in Vue Grid component

You can render the custom component in frozen columns using edit params.

In the following example, we have rendered the **Datepicker** component to the **OrderDate** column. Here the datepicker object is appended to the corresponding input element in the editing row.

#### APP.VUE

```

<template>
  <div id="app">
    <ejs-grid :frozenColumns="2" :dataSource="data"
    :editSettings="editSettings" :toolbar="toolbar"
    height="273px">
      <e-columns>
        <e-column field="OrderID" headerText="Order ID" :isPrimaryKey="true"
width="100"></e-column>
        <e-column field="CustomerID" headerText="Customer ID"
width="120"></e-column>
        <e-column field="ShipCity" headerText="Ship City" textAlign="Right"
width="120"></e-column>
        <e-column field="OrderDate" headerText="Order Date" type="date"
format="yMd"
:edit="dpParams" width="150"></e-column>
      </e-columns>
    </ejs-grid>
  </div>
</template>
<script>
import Vue from "vue";
import {
  GridPlugin,
  Page,
  Freeze,
  Toolbar,
  Edit
} from "@syncfusion/ej2-vue-grids";
import { DatePicker } from "@syncfusion/ej2-calendars";
import { data } from "../datasource.js";
Vue.use(GridPlugin);
let elem;
let datePickerObj;
export default {
  data() {
    return {
      data: data,
      editSettings: {
        allowEditing: true,
        allowAdding: true,
        allowDeleting: true

```

```

    },
    toolbar: ["Add", "Edit", "Delete", "Update", "Cancel"],
    dpParams: {
      create: function() {
        elem = document.createElement("input");
        return elem;
      },
      read: () => {
        return datePickerObj.value;
      },
      destroy: () => {
        datePickerObj.destroy();
      },
      write: args => {
        datePickerObj = new DatePicker({
          value: new Date(args.rowData[args.column.field]),
          floatLabelType: "Never"
        });
        datePickerObj.appendTo(args.element);
      }
    }
  };
},
provide: {
  grid: [Page, Edit, Toolbar, Freeze]
}
};
</script>
<style>
  @import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/grid/how-to/dropdown-cs4" %}

[Edit template from another vue in Vue Grid component](#)

You can achieve the dialog template editing using another vue page.

,

<template>

<div id="app">

<ejs-grid :dataSource="data" :editSettings="editSettings" :actionBegin="actionBegin"  
:actionComplete="actionComplete" :toolbar="toolbar" height="273px">

<e-columns>

<e-column field="OrderID" headerText="Order ID" textAlign="Right" :isPrimaryKey="true"  
width="100"></e-column>

<e-column field="CustomerID" headerText="Customer ID" width="120"></e-column>

<e-column field="ShipCountry" headerText="Ship Country" width="150"></e-column>

</e-columns>

</ejs-grid>

```
</div>
</template>
<script>
import Vue from "vue";
import { GridPlugin, Page, Toolbar, Edit } from "@syncfusion/ej2-vue-grids";
import { data } from './datasource.js';
import DialogTemplate from "./dialogtemp.vue";
import { DatePickerPlugin } from "@syncfusion/ej2-vue-calendars";
import { DropDownListPlugin } from "@syncfusion/ej2-vue-dropdowns";
import { NumericTextBox } from "@syncfusion/ej2-inputs";
import { NumericTextBoxPlugin } from "@syncfusion/ej2-vue-inputs";
import { DataUtil } from '@syncfusion/ej2-data';
Vue.use(GridPlugin);
Vue.use(DropDownListPlugin);
Vue.use(DatePickerPlugin);
Vue.use(NumericTextBoxPlugin)
export default {
  data() {
    return {
      data: data,
      editSettings: {
        allowEditing: true,
        allowAdding: true,
        allowDeleting: true,
        mode: "Dialog",
        template: function() {
          return { template: DialogTemplate };
        }
      },
      toolbar: ["Add", "Edit", "Delete", "Update", "Cancel"]
    };
  },
  provide: {
```

```
grid: [Page, Edit, Toolbar]
},
methods: {
  actionBegin(args) {
    if (args.requestType === "save") {
      // cast string to integer value.
      args.data["Freight"] = parseFloat(args.form.querySelector("#Freight").value);
    }
  },
  actionComplete(args) {
    // Set initail Focus
    if (args.requestType === "beginEdit") {
      args.form.elements.namedItem("OrderID").focus();
    }
  }
}
}
}
</script>
<style>
@import "https://cdn.syncfusion.com/ej2/material.css";
@import "https://stackpath.bootstrapcdn.com/bootstrap/4.1.2/css/bootstrap.min.css";
.form-group.col-md-6 {
width: 250px;
height: 54px;
}
.form-group.col-md-12 {
height: 72px;
}
ShipAddress {
resize: vertical;
}
</style>
`
```

Create new vue page with the name of `dialogtemp.vue` and paste the below code.

```
<template>
<div formGroup="orderForm">
<div class="form-row">
<div class="form-group col-md-6">
<div class="e-float-input e-control-wrapper">
<input ref="OrderID" id="OrderID" name="OrderID" v-model="data.OrderID" type="text"
:disabled="!data.isAdd">
<span class="e-float-line"></span>
<label class="e-float-text e-label-top" for="OrderID">Order ID</label>
</div>
</div>
<div class="form-group col-md-6">
<div class="e-float-input e-control-wrapper">
<input ref="CustomerID" id="CustomerID" name="CustomerID" v-model="data.CustomerID"
type="text">
<span class="e-float-line"></span>
<label class="e-float-text e-label-top" for="CustomerID">Customer Name</label>
</div>
</div>
</div>
<div class="form-row">
<div class="form-group col-md-6">
<ejs-numerictextbox id="Freight" placeholder="Freight" v-model="data.Freight"
floatLabelType="Always" ></ejs-numerictextbox>
</div>
<div class="form-group col-md-6">
<ejs-datepicker id="OrderDate" placeholder="Order Date" v-model="data.OrderDate"
floatLabelType="Always" ></ejs-datepicker>
</div>
</div>
<div class="form-row">
<div class="form-group col-md-6">
```



```

<ejs-dropdownlist id="ShipCountry" v-model="data.ShipCountry"
:dataSource="shipCountryDistinctData" :fields="{text: 'ShipCountry', value: 'ShipCountry' }"
placeholder="Ship Country" popupHeight="300px" floatLabelType="Always" ></ejs-dropdownlist>
</div>
<div class="form-group col-md-6">
<ejs-dropdownlist id="ShipCity" v-model="data.ShipCity" :dataSource="shipCityDistinctData"
:fields="{text: 'ShipCity', value: 'ShipCity' }" placeholder="Ship City" popupHeight="300px"
floatLabelType="Always"></ejs-dropdownlist>
</div>
</div>
<div class="form-row">
<div class="form-group col-md-12">
<div class="e-float-input e-control-wrapper">
<textarea id="ShipAddress" name="ShipAddress" type="text" v-model="data.ShipAddress"></textarea>
<span class="e-float-line"></span>
<label class="e-float-text e-label-top" for="ShipAddress">Ship Address</label>
</div>
</div>
</div>
</div>
</div>
</template>
<script>
import { data } from './datasource.js';
import { DataUtil } from "@syncfusion/ej2-data";
export default {
data() {
let shipCity = DataUtil.distinct(data, "ShipCity", true);
let shipCountry = DataUtil.distinct(data, "ShipCountry", true);
return {
data: {},
shipCityDistinctData: shipCity,
shipCountryDistinctData: shipCountry
};
},

```

```

mounted() {
// Set initail Focus
if (this.data.isAdd) {
this.$refs.OrderID.focus();
} else {
this.$refs.CustomerID.focus();
}
}
};
</script>
`

```

#### Access grid instance in Vue Grid component

You can access the instance of the Vue Grid in the following way.

```

`ts
dataBound: function (args) {
var gridObj = this.$refs.grid1.$el.ej2_instances[0]; // get the instance of the Grid.
Object.assign(gridObj.filterModule.filterOperators, {
startsWith: "contains", // change the default operator as contains for string type column.
});
}
`

```

In the following sample, the filter operation is performed by getting the instance of the Vue Grid component.

#### **APP.VUE**

```

<template>
  <div id="app">
    <ejs-grid ref="grid1" :dataSource="data" :allowPaging="true"
:allowFiltering="true" :filterSettings="filterOptions"
:dataBound="dataBound">
      <e-columns>
        <e-column field="InternalDocumentNo"
headerText="InternalDocumentNo" width="180" isPrimaryKey="true"
headerTextAlign="center">
          </e-column>
        <e-column field="DocumentNumber" headerText="DocumentNumber"
width="180" headerTextAlign="center" :filter="filterOptions">
          </e-column>
        <e-column field="CustomerCode" headerText="CustomerCode"
width="180" headerTextAlign="center">
          </e-column>
      </e-columns>
    </div>
  </template>

```

```

        <e-column field="WarehouseName" headerText="WarehouseName"
width="180" isPrimaryKey="true" headerTextAlign="center">
        </e-column>
    </e-columns>
</ejs-grid>
</div>
</template>
<script>
import Vue from "vue";
import { GridPlugin, Page, Filter } from "@syncfusion/ej2-vue-grids";
Vue.use(GridPlugin);
export default {
    data() {
        return {
            data: [
                {
                    InternalDocumentNo: 84374,
                    DocumentNumber: "SOV-7855",
                    CustomerCode: "0500733131",
                    WarehouseName: "Main Warehouse"
                },
                {
                    InternalDocumentNo: 84372,
                    DocumentNumber: "SOV-7853",
                    CustomerCode: "0500733132",
                    WarehouseName: "Main Warehouse"
                },
                {
                    InternalDocumentNo: 84365,
                    DocumentNumber: "SOV-7852",
                    CustomerCode: "0500733131",
                    WarehouseName: "Main Warehouse"
                },
                {
                    InternalDocumentNo: 84358,
                    DocumentNumber: "SOV-7850",
                    CustomerCode: "0500733131",
                    WarehouseName: "Main Warehouse",
                },
                {
                    InternalDocumentNo: 84357,
                    DocumentNumber: "SOV-7849",
                    CustomerCode: "0500733131",
                    WarehouseName: "Main Warehouse"
                }
            ],
            filterOptions: {
                operator: "contains",
            }
        };
    },
    methods: {
        dataBound: function (args) {
            var gridObj = this.$refs.grid1.$el.ej2_instances[0];
            Object.assign(gridObj.filterModule.filterOperators, {
                startsWith: "contains", // change the default operator as
contains for string type column
            });
        }
    }
};

```

```

    });
  },
  provide: {
    grid: [Page, Filter]
  }
};
</script>
<style>
@import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/grid/how-to/dropdown-cs3" %}

### Get parent column instance in Vue Grid component

Emit values from one component and listen to emitted values in other component by using the **eventHub**. The **eventHub** is a global event bus used to communicate between any components.

In the following example, we have defined a template column in the grid column definition which emits a value and listens the emitted value in the [created](#) event of the Grid component.

### APP.VUE

```

<template>
  <div id="app">
    <ejs-grid ref="grid" :dataSource="data" height="272px">
      <e-columns>
        <e-column
          field="OrderID" headerText="Order ID" textAlign="Right"
          width="100" isPrimaryKey="true"
        ></e-column>
        <e-column
          field="CustomerID" headerText="Customer ID" width="120"
        ></e-column>
        <e-column
          field="ShipCity" headerText="Ship City" width="100"
        ></e-column>
        <e-column
          field="ShipName" headerText="Ship Name" width="100"
          :template="cTemplate"
        ></e-column>
      </e-columns>
    </ejs-grid>
  </div>
</template>
<script>
import Vue from "vue";
import { GridPlugin } from "@syncfusion/ej2-vue-grids";
import { data } from './datasource.js'
Vue.use(GridPlugin);
Vue.prototype.$eventHub = new Vue();
export default {
  data() {
    return {
      data: data,
      cTemplate: function() {

```

```

        return {
            template: Vue.component("columnTemplate", {
                template: `<button v-
on:click="click(event)">button</button>`,
                data: function () {
                    return {
                        data: {},
                    };
                },
                methods: {
                    click: function (event) {
                        this.$eventHub.$emit("detail",
this.data.index);
                    },
                },
            }),
        };
    },
    created() {
        this.$eventHub.$on("detail", (event) => {
            alert("passed value:" + event);
        });
    },
};
</script>
<style>
@import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/grid/filter/default-cs13" %}

### How to create a routing sample in Vue Grid component

[Vue Router](#) is the official routing library for Vue.js which is widely used for building single-page applications (SPA's). It provides a way to handle navigation and routing within an application. In Vue routing, you can able to configure the different routes for different URL's or paths to navigate between different sections or pages of the application. This approach is particularly advantageous for SPA's since it permits the loading of content dynamically without requiring a complete page refresh.

#### Steps to create a routing sample in Vue Grid component

##### 1. Creating a vue project and integrating the Syncfusion Grid component:

A simple vue project can be created by following the steps under [getting started](#) section of this documentation. Install the Vue Router package by running the following command in your project's directory.

`

```
npm install vue-router
```

`

To use Vue routing in your application, you need to create at least two components that can be navigated from one to another.

## 2. Defining Routes:

To define the routes create a new folder **router** under src directory and create new file named **index.js** inside it (**src\router\index.js**). You can define the routes that correspond to different components in this file. Import the components and define an array of route objects that specifies the path and the component to be rendered for each route.

```
,  
  
import Vue from 'vue'  
import Router from 'vue-router'  
import About from '../components/About'  
import Home from '../components/Home'  
import Contact from '../components/Contact'  
Vue.use(Router);  
const router = new Router({  
  mode: 'history',  
  routes: [  
    {  
      path: '/',  
      name: 'Home',  
      component: Home  
    },  
    {  
      path: '/about',  
      name: 'About',  
      component: About  
    },  
    {  
      path: '/contact',  
      name: 'Contact',  
      component: Contact  
    },  
  ]  
});  
export default router;  
,
```

### 3. Configuring the router module:

Open the **src/main.js** file and import the router.js.

```
,  
  
import router from './router'
```

### 4. Setting up navigation:

Include the below code lines in **App.vue** file to set up the navigation.

```
,  
  
<template>  
<div id="app">  
<nav>  
<ul>  
<li><router-link to="/">Home</router-link></li>  
<li><router-link to="/about">About</router-link></li>  
<li><router-link to="/contact">Contact</router-link></li>  
</ul>  
</nav>  
<router-view></router-view>  
</div>  
</template>  
,
```

The `<router-link>` components are used for navigation, the `to` attribute specifies the path to navigate.

The `<router-view>` component is responsible for rendering the content of each route. It acts as a placeholder that will be replaced with the component corresponding to the active route.

[Here](#), you can find the Vue Routing sample with Syncfusion DataGrid.

### Testing the Vue application using Jest

Vue [Jest](#) testing is a popular approach to test Vue applications using the Jest testing framework. This approach involves the creation and execution of unit tests specifically designed for Vue components. By conducting unit testing, which focuses on testing isolated units of code like functions, methods, and components to ensure that they behave as expected. This approach validates the individual units of your Vue components, catch potential bugs early in the development process, and maintains the reliability and stability of your Vue application. To create a Jest test case for the Grid component, follow the below steps:

#### Step 1: Set up the Jest testing environment

##### I. Check and install the node version:

You need to verify if the installed version of Node is 14 or higher. If it is below version 14, you must install a version of Node above 14. You can refer the following link to install the [node version](#). You can select the any node version is 14 or above and installed.

## II. Create an Vue application and install the Syncfusion Grid package:

To create an Vue application and install the Syncfusion Grid package, you can refer to the [Getting started](#) documentation.

## III. Install the Jest:

Run the following command to install the Jest dependency using npm.

```
,
npm install --save-dev jest
,
```

## IV. How to add the unit jest test case environment:

Run the following command to add the unit jest test case environment.

```
,
vue add unit-jest
npm install jest-environment-jsdom --save-dev
,
```

## V. How to add the test command in the script section:

You should check that the **test** command is specified in the **script** section of the **package.json** file. If the **script** section is not specify the **test** command, you must define the **test** command in the **script** section of the **package.json** file.

```
,
"test": "jest"
,
```

## Step 2: Adding a Grid component

Refer to the [documentation](#) to add the styling for the Grid component. The following code is used in this demonstration to create a Grid component. For further information on creating the Grid component, refer to the detailed [documentation](#).

### App.vue:

```
`typescript
<template>
<div id="app">
<ejs-grid id='Grid' :dataSource='data.slice(0,5)' height='315'>
<e-columns>
<e-column field='CustomerID' headerText='CustomerID' textAlign='Right' width=90></e-column>
```



```

<e-column field='ContactName' headerText='ContactName' width=120></e-column>
<e-column field='Address' headerText='Address' width=150></e-column>
<e-column field='Country' headerText='Country' width=150></e-column>
</e-columns>
</ejs-grid>
</div>
</template>
<script>
import Vue from "vue";
import { GridPlugin } from "@syncfusion/ej2-vue-grids";
// import the your datasource instead of this
import { customerData } from './datasource';
Vue.use(GridPlugin);
export default {
  data() {
    return {
      data: customerData
    };
  }
}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
</style>
`

```

### Step 3: To implement the Jest test case

You can write the Jest test case in the spec.js extension file. You need to import the below required files in your component. You need to import the `shallowMount` function from the **@vue/test-utils** package. This function is used to render the component for vue testing. To test a specific component, you need to import it into your testing environment. In this demo, we have written the grid component in the **App** file. So, you need to import the **App** file in the Jest test case.

```

`typescript
import { shallowMount } from '@vue/test-utils'
import App from '@App.vue'

```

`

### I. Define test suite:

The `describe` function is utilized to define the test suite. Within the `describe` function, use the `it` function to specify the individual test cases.

```
`typescript
describe('App component', () => {
  it('Length of the record', () => {
  });
});
```

`

### II. Types of testing:

You need to add the different types of test cases in a `it` block.

#### 1. Snapshot Testing:

The Snapshot testing involves capturing a snapshot of the rendered output of a component and comparing it against a previously stored snapshot. If the current output matches the stored snapshot, the test case will be passed successfully.

##### Example:

In the below example, the `it` block is utilized to define a test case for the **Snapshot testing**. Within the test case, the `shallowMount` function from the `@vue/test-utils` package is used to create a shallow wrapper for the **App** component. After, the `expect` statement verifies that the rendered output matches the stored snapshot, utilizing the `toMatchSnapshot` matcher provided by Jest.

```
`typescript
it('Snapshot testing', () => {
  const wrapper = shallowMount(App);
  expect(wrapper).toMatchSnapshot();
});
```

`

```

PASS tests/unit/App.spec.js (6.806 s)
  App component
    ✓ Snapshot testing (26 ms)

    > 1 snapshot written.
  Snapshot Summary
    > 1 snapshot written from 1 test suite.

Test Suites: 1 passed, 1 total
Tests:       1 passed, 1 total
Snapshots:  1 written, 1 total
Time:        29.541 s
Ran all test suites.

```

## 2. DOM Testing:

The DOM testing involves testing the behavior and interact of Vue component. This goal is to ensure that the component function correctly and produce the expected output when interacting with the DOM. You can utilize libraries like **@vue/test-utils** or **@testing-library/vue** to manipulate the rendered component in the DOM testing.

### Example:

The `it` block is used to define a test case for the "Length of the record". Within the test case, the `shallowMount` function from the **@vue/test-utils** package is used to create a shallow wrapper for the **App** component. After, you need to create the instance of grid component. We check that the data grid in the data source has the appropriate number of data records. The `dataSource` property is employed to retrieve the record of the data. By utilizing this property, we can verify the accurate population of data in the grid component.

```

`typescript
it('Length of the record', () => {
  const wrapper = shallowMount(App);
  var gridElement = wrapper.find('#Grid');
  const gridInstance = gridElement.vm;
  expect(gridInstance.$props.dataSource).toHaveLength(5);
});
`

```

The following example illustrates how to create the grid sample and how to write the Jest test case.

### APP.VUE

```

<template>
  <div id="app">
    <ejs-grid id='Grid' :dataSource='data.slice(0,5)' height='315'>
      <e-columns>
        <e-column field='CustomerID' headerText='CustomerID'
textAlign='Right' width=90></e-column>
        <e-column field='ContactName' headerText='ContactName'
width=120></e-column>
      </e-columns>
    </ejs-grid>
  </div>
</template>

```

```

        <e-column field='Address' headerText='Address' width=150></e-
column>
        <e-column field='Country' headerText='Country' width=150></e-
column>
    </e-columns>
</ejs-grid>
</div>
</template>
<script>
import Vue from "vue";
import { GridPlugin } from "@syncfusion/ej2-vue-grids";
import { customerData } from '../datasource';
Vue.use(GridPlugin);
export default {
    data() {
        return {
            data: customerData
        };
    }
}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
</style>

```

### APP.SPEC.JS

```

import { shallowMount } from '@vue/test-utils'
import App from '@app.vue'
describe('App component', () => {
    it('Length of the record', () => {
        const wrapper = shallowMount(App);
        var gridElement = wrapper.find('#Grid');
        const gridInstance = gridElement.vm;
        expect(gridInstance.$props.dataSource).toHaveLength(5);
    })
})

```

### SNAP.SPEC.JS

```

import { shallowMount } from '@vue/test-utils'
import App from '@App.vue'
describe('App component', () => {
    it('Snapshot testing', () => {
        const wrapper = shallowMount(App);
        expect(wrapper).toMatchSnapshot();
    });
})

```

### Run the Jest test case:

Run the following command to execute the Jest test case.

,

npm run test

- \* This is only for local data. You can use the [currentViewData](#) property by rendering the remote data.
- \* You can find the sample of the Unit Jest testing in DataGrid [here](#)

### Customize the Empty Record Template in Vue Grid component

The empty record template feature in the Grid allows you to use custom content such as images, text, or other components, when the grid doesn't contain any records to display. This feature replaces the default message of 'No records to display' typically shown in the grid.

To activate this feature, set the `emptyRecordTemplate` property of the Grid. The `emptyRecordTemplate` property expects the HTML element or a function that returns the HTML element.

In the following example, an image and text have been rendered as a template to indicate that the grid has no data to display.

#### APP.VUE

```
<template>
  <div class="col-lg-12 control-section">
    <div>
      <ejs-grid :dataSource='data'
      :emptyRecordTemplate='emptyRecordTemplate' :editSettings='editSettings'
      :toolbar='toolbar' :allowPaging='true'
      :pageSettings='pageSettings'>
        <e-columns>
          <e-column field='OrderID' headerText='Order ID'
            width='120' textAlign='Right' :isPrimaryKey='true'
            :validationRules='orderidrules'></e-column>
          <e-column field='CustomerID' headerText='Customer ID'
            width='120' :validationRules='customeridrules'></e-column>
          <e-column field='Freight' headerText='Freight' width='180'
            format='C2' textAlign='Right' editType='numericedit'
            :validationRules='freightrules'></e-column>
          <e-column field='OrderDate' headerText='Order Date'
            width='130' editType='datetimepickeredit' :format='formatoptions'
            textAlign='Right'></e-column>
          <e-column field='ShipCountry' headerText='Ship Country'
            width='150' editType='dropdownedit' :edit='editparams'></e-column>
        </e-columns>
      </ejs-grid>
    </div>
  </div>
</template>
<script>
import Vue from "vue";
import { GridPlugin, GridComponent, ColumnDirective, ColumnsDirective, Edit,
Page, Toolbar } from "@syncfusion/ej2-vue-grids";
import { cascadeData } from "../datasource";
Vue.use(GridPlugin);
export default {
  components: {
    'ejs-grid': GridComponent,
    'e-column': ColumnDirective,
    'e-columns': ColumnsDirective
  }
}
```

```

    },
    data: () => {
      return {
        data: [],
        pageSettings: { pageCount: 5 },
        editSettings: { allowEditing: true, allowAdding: true,
allowDeleting: true },
        toolbar: ['Add', 'Edit', 'Delete', 'Update', 'Cancel'],
        orderidrules: { required: true, number: true },
        formatoptions: { type: 'dateTime', format: 'M/d/y hh:mm a' },
        freightrules: { required: true, number: true },
        editparams: { params: { dataSource: cascadeData, fields:
{text:"ShipCountry",value:"ShipCountry"}}},
        customeridrules: { required: true },
        emptyRecordTemplate: `
          <div class='emptyRecordTemplate'>
            <img src='./emptyRecordTemplate.svg' class='e-emptyRecord'
alt="No record">
            <span>There is no data available to display at the
moment.</span>
          </div>`,
      };
    },
    provide: {
      grid: [Edit, Page, Toolbar]
    }
  }
</script>
<style>
  @import "../node_modules/@syncfusion/ej2-vue-grids/styles/material.css";
  .emptyRecordTemplate {
    text-align: center;
  }
  .e-emptyRecord {
    display: block;
    margin: 10px auto;
  }
</style>

```

{% previewsample "page.domainurl/code-snippet/grid/how-to/empty-record-template-cs1" %}

## HeatMap Chart

### Getting Started with the Vue HeatMap Component in Vue 2

This article provides a step-by-step guide for setting up a Vue 2 project using [Vue-CLI](#) and integrating the Syncfusion Vue HeatMap component

#### Prerequisites

[System requirements for Syncfusion Vue UI components](#)

#### Dependencies

For using heat map, the following minimum requirements are needed.

`javascript

```
|-- @syncfusion/ej2-vue-heatmap  
|-- @syncfusion/ej2-base  
|-- @syncfusion/ej2-data  
|-- @syncfusion/ej2-heatmap  
|-- @syncfusion/ej2-vue-base  
|-- @syncfusion/ej2-svg-base  
`
```

### Setting up the Vue 2 project

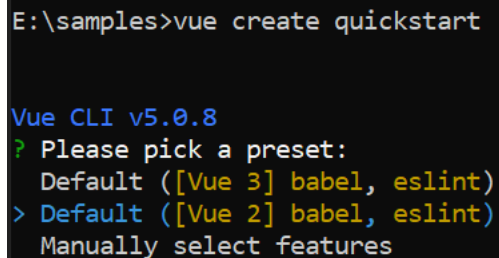
To generate a Vue 2 project using Vue-CLI, use the [vue create](#) command. Follow these steps to install Vue CLI and create a new project:

```
`bash  
npm install -g @vue/cli  
vue create quickstart  
cd quickstart  
npm run serve  
`
```

or

```
`bash  
yarn global add @vue/cli  
vue create quickstart  
cd quickstart  
yarn run serve  
`
```

When creating a new project, choose the option **Default ([Vue 2] babel, eslint)** from the menu.



```
E:\samples>vue create quickstart  
  
Vue CLI v5.0.8  
? Please pick a preset:  
  Default ([Vue 3] babel, eslint)  
> Default ([Vue 2] babel, eslint)  
  Manually select features
```

Once the **quickstart** project is set up with default settings, proceed to add Syncfusion components to the project.

### Add Syncfusion Vue packages

Syncfusion packages are available at [npmjs.com](https://www.npmjs.com). To use Vue components, install the required npm package.

This article uses the [Vue HeatMap component](#) as an example. Install the `@syncfusion/ej2-vue-heatmap` package by running the following command:

```
`bash
```

```
npm install @syncfusion/ej2-vue-heatmap --save
```

```
,
```

or

```
`bash
```

```
yarn add @syncfusion/ej2-vue-heatmap
```

```
,
```

### Adding Syncfusion Vue HeatMap component

Follow the below steps to add the Vue HeatMap component:

1\ First, import and register the HeatMap component in the `script` section of the `src/App.vue` file.

#### ~/SRC/APP.VUE

```
<script>
import { HeatMapComponent } from '@syncfusion/ej2-vue-heatmap';
export default {
  components: {
    'ejs-heatmap' : HeatMapComponent
  },
  data: function() {
    return {
      dataSource: [
        [73, 39, 26, 39, 94, 0],
        [93, 58, 53, 38, 26, 68],
        [99, 28, 22, 4, 66, 90],
        [14, 26, 97, 69, 69, 3],
        [7, 46, 47, 47, 88, 6],
        [41, 55, 73, 23, 3, 79],
        [56, 69, 21, 86, 3, 33],
        [45, 7, 53, 81, 95, 79],
        [60, 77, 74, 68, 88, 51],
        [25, 25, 10, 12, 78, 14],
        [25, 56, 55, 58, 12, 82],
        [74, 33, 88, 23, 86, 59]
      ]
    }
  }
}
</script>
```

2\ In the `template` section, define the HeatMap component.

#### ~/SRC/APP.VUE



```
<template>
<div id="app">
<ejs-heatmap id="heatmap" :dataSource='dataSource'></ejs-heatmap>
</div>
</template>
```

Here is the summarized code for the above steps in the **src/App.vue** file:

#### ~/SRC/APP.VUE

```
<template>
  <ejs-heatmap id="heatmap" :dataSource='dataSource'></ejs-heatmap>
</template>
<script>
import { HeatMapComponent } from "@syncfusion/ej2-vue-heatmap";
export default {
  components: {
    'ejs-heatmap' : HeatMapComponent
  },
  data: function() {
    return {
      dataSource: [
        [73, 39, 26, 39, 94, 0],
        [93, 58, 53, 38, 26, 68],
        [99, 28, 22, 4, 66, 90],
        [14, 26, 97, 69, 69, 3],
        [7, 46, 47, 47, 88, 6],
        [41, 55, 73, 23, 3, 79],
        [56, 69, 21, 86, 3, 33],
        [45, 7, 53, 81, 95, 79],
        [60, 77, 74, 68, 88, 51],
        [25, 25, 10, 12, 78, 14],
        [25, 56, 55, 58, 12, 82],
        [74, 33, 88, 23, 86, 59]
      ]
    }
  }
}
</script>
```

#### Run the project

To run the project, use the following command:

```
`bash
```

```
npm run serve
```

```
,
```

or

```
`bash
```

```
yarn run serve
```

```
,
```

{% previewsample "page.domainurl/code-snippet/heatmap-chart/getting-started-cs8" %}

### Module injection

The heat map components are segregated into individual feature-wise modules. To use its feature, you need to inject its feature service in the AppModule. In the current application, the basic heat map is modified to visualize sales revenue data for week, and the tooltip and legend features of the heat map are used. Find the relevant feature modules and descriptions as follows.

- Legend - Provides the legend feature by injecting it.
- Tooltip - Provides the tooltip feature by injecting it.

Now, import the above-mentioned modules from the heat map package and inject them into **provide**.

#### ~/SRC/APP.VUE

```
<script>
import Vue from "vue";
import { HeatMapComponent, Legend, Tooltip } from "@syncfusion/ej2-vue-heatmap";
export default {
  components: {
    'ejs-heatmap': HeatMapComponent
  },
  provide: {
    heatmap: [Legend, Tooltip]
  }
};
</script>
```

### Populate heat map with data

This section explains how to populate the following two-dimensional array data to the heat map.

#### ~/SRC/APP.VUE

```
<template>
  <div id="app">
    <ejs-heatmap id="heatmap" :dataSource='dataSource'></ejs-heatmap>
  </div>
</template>
<script>
import { HeatMapComponent } from '@syncfusion/ej2-vue-heatmap';
export default {
  components: {
    'ejs-heatmap': HeatMapComponent
  },
  data: function() {
    return {
      dataSource: [
        [73, 39, 26, 39, 94, 0],
        [93, 58, 53, 38, 26, 68],
        [99, 28, 22, 4, 66, 90],
        [14, 26, 97, 69, 69, 3],
        [7, 46, 47, 47, 88, 6],
        [41, 55, 73, 23, 3, 79],
        [56, 69, 21, 86, 3, 33],
      ]
    }
  }
};
```

```

        [45, 7, 53, 81, 95, 79],
        [60, 77, 74, 68, 88, 51],
        [25, 25, 10, 12, 78, 14],
        [25, 56, 55, 58, 12, 82],
        [74, 33, 88, 23, 86, 59]
    ]
}
}
}
</script>

```

{% previewsample "page.domainurl/code-snippet/heatmap-chart/getting-started-cs1" %}

### Enable axis labels

You can add axis labels to the heat map and format those labels using the [xAxis](#) and [yAxis](#) properties. Axis labels provide additional information about the data points populated in the heat map.

### ~/SRC/APP.VUE

```

<template>
  <div class="control_wrapper">
    <ejs-heatmap id="heatmap" :dataSource='dataSource' :xAxis='xAxis'
    :yAxis='yAxis'></ejs-heatmap>
  </div>
</template>
<script>
import { HeatMapComponent, Tooltip, Legend } from '@syncfusion/ej2-vue-heatmap';
export default {
  components: {
    'ejs-heatmap': HeatMapComponent
  },
  data: function() {
    return {
      xAxis: {
        labels: ['Nancy', 'Andrew', 'Janet', 'Margaret', 'Steven',
          'Michael', 'Robert', 'Laura', 'Anne', 'Paul',
          'Karin', 'Mario'],
      },
      yAxis: {
        labels: ['Mon', 'Tues', 'Wed', 'Thurs', 'Fri', 'Sat'],
      },
      dataSource: [
        [73, 39, 26, 39, 94, 0],
        [93, 58, 53, 38, 26, 68],
        [99, 28, 22, 4, 66, 90],
        [14, 26, 97, 69, 69, 3],
        [7, 46, 47, 47, 88, 6],
        [41, 55, 73, 23, 3, 79],
        [56, 69, 21, 86, 3, 33],
        [45, 7, 53, 81, 95, 79],
        [60, 77, 74, 68, 88, 51],
        [25, 25, 10, 12, 78, 14],
        [25, 56, 55, 58, 12, 82],
        [74, 33, 88, 23, 86, 59]
      ]
    }
  }
}

```

```

    }
  },
  provide: {
    heatmap: [Tooltip, Legend]
  },
}
</script>

```

{% previewsample "page.domainurl/code-snippet/heatmap-chart/getting-started-cs2" %}

### Add heat map title

Add a title using the [titleSettings](#) property to the heat map to provide quick information to the user about the data populated in the heat map.

### ~/SRC/APP.VUE

```

<template>
  <div class="control_wrapper">
    <ejs-heatmap id="heatmap" :dataSource='dataSource' :xAxis='xAxis'
    :yAxis='yAxis' :titleSettings='titleSettings'></ejs-heatmap>
  </div>
</template>
<script>
import { HeatMapComponent, Tooltip, Legend } from '@syncfusion/ej2-vue-heatmap';
export default {
  components: {
    'ejs-heatmap': HeatMapComponent
  },
  data: function() {
    return {
      xAxis: {
        labels: ['Nancy', 'Andrew', 'Janet', 'Margaret', 'Steven',
          'Michael', 'Robert', 'Laura', 'Anne', 'Paul',
          'Karin', 'Mario'],
      },
      yAxis: {
        labels: ['Mon', 'Tues', 'Wed', 'Thurs', 'Fri', 'Sat'],
      },
      titleSettings: {
        text: 'Sales Revenue per Employee (in 1000 US$)',
        textStyle: {
          size: '15px',
          fontWeight: '500',
          fontStyle: 'Normal',
          fontFamily: 'Segoe UI'
        }
      },
    },
    dataSource: [
      [73, 39, 26, 39, 94, 0],
      [93, 58, 53, 38, 26, 68],
      [99, 28, 22, 4, 66, 90],
      [14, 26, 97, 69, 69, 3],
      [7, 46, 47, 47, 88, 6],
      [41, 55, 73, 23, 3, 79],
      [56, 69, 21, 86, 3, 33],
    ]
  }
}

```

```

        [45, 7, 53, 81, 95, 79],
        [60, 77, 74, 68, 88, 51],
        [25, 25, 10, 12, 78, 14],
        [25, 56, 55, 58, 12, 82],
        [74, 33, 88, 23, 86, 59]
    ]
}
},
provide:{
    heatmap:[Tooltip, Legend]
},
}
</script>

```

{% previewsample "page.domainurl/code-snippet/heatmap-chart/getting-started-cs3" %}

### Enable legend

Use a legend for the heat map in the [legendSettings](#) object by setting the [visible](#) property to **true** and injecting the **Legend** module into the **provide**.

### ~/SRC/APP.VUE

```

<template>
    <div class="control_wrapper">
        <ejs-heatmap id="heatmap" :dataSource='dataSource' :xAxis='xAxis'
        :yAxis='yAxis' :titleSettings='titleSettings'
        :legendSettings='legendSettings'></ejs-heatmap>
    </div>
</template>
<script>
import { HeatMapComponent, Tooltip, Legend } from '@syncfusion/ej2-vue-heatmap';
export default {
    components: {
        'ejs-heatmap': HeatMapComponent
    },
    data: function() {
        return {
            xAxis: {
                labels: ['Nancy', 'Andrew', 'Janet', 'Margaret', 'Steven',
                    'Michael', 'Robert', 'Laura', 'Anne', 'Paul',
                    'Karin', 'Mario'],
            },
            yAxis:{
                labels: ['Mon', 'Tues', 'Wed', 'Thurs', 'Fri', 'Sat'],
            },
            titleSettings: {
                text: 'Sales Revenue per Employee (in 1000 US$)',
                textStyle: {
                    size: '15px',
                    fontWeight: '500',
                    fontStyle: 'Normal',
                    fontFamily: 'Segoe UI'
                }
            },
            dataSource: [

```

```

        [73, 39, 26, 39, 94, 0],
        [93, 58, 53, 38, 26, 68],
        [99, 28, 22, 4, 66, 90],
        [14, 26, 97, 69, 69, 3],
        [7, 46, 47, 47, 88, 6],
        [41, 55, 73, 23, 3, 79],
        [56, 69, 21, 86, 3, 33],
        [45, 7, 53, 81, 95, 79],
        [60, 77, 74, 68, 88, 51],
        [25, 25, 10, 12, 78, 14],
        [25, 56, 55, 58, 12, 82],
        [74, 33, 88, 23, 86, 59]
    ],
    legendSettings: {
        visible: true,
        position: 'Right',
        showLabel: true,
        height: "150"
    }
},
provide: {
    heatmap: [Tooltip, Legend]
},
}
</script>

```

{% previewsample "page.domainurl/code-snippet/heatmap-chart/getting-started-cs4" %}

#### Add data label

Add data labels to improve the readability of the heat map. This can be achieved by setting the [showLabel](#) property to **true** in the [cellSettings](#) object.

#### ~/SRC/APP.VUE

```

<template>
  <div class="control_wrapper">
    <ejs-heatmap id="heatmap" :dataSource='dataSource' :xAxis='xAxis'
    :yAxis='yAxis' :titleSettings='titleSettings'
    :legendSettings='legendSettings' :cellSettings='cellSettings'>
    </ejs-heatmap>
  </div>
</template>
<script>
import { HeatMapComponent, Tooltip, Legend } from '@syncfusion/ej2-vue-heatmap';
export default {
  components: {
    'ejs-heatmap': HeatMapComponent
  },
  data: function() {
    return {
      xAxis: {
        labels: ['Nancy', 'Andrew', 'Janet', 'Margaret', 'Steven',
          'Michael', 'Robert', 'Laura', 'Anne', 'Paul',
          'Karin', 'Mario'],

```

```

    },
    yAxis: {
      labels: ['Mon', 'Tues', 'Wed', 'Thurs', 'Fri', 'Sat'],
    },
    cellSettings: {
      showLabel: true,
    },
    titleSettings: {
      text: 'Sales Revenue per Employee (in 1000 US$)',
      textStyle: {
        size: '15px',
        fontWeight: '500',
        fontStyle: 'Normal',
        fontFamily: 'Segoe UI'
      }
    },
  },
  dataSource: [
    [73, 39, 26, 39, 94, 0],
    [93, 58, 53, 38, 26, 68],
    [99, 28, 22, 4, 66, 90],
    [14, 26, 97, 69, 69, 3],
    [7, 46, 47, 47, 88, 6],
    [41, 55, 73, 23, 3, 79],
    [56, 69, 21, 86, 3, 33],
    [45, 7, 53, 81, 95, 79],
    [60, 77, 74, 68, 88, 51],
    [25, 25, 10, 12, 78, 14],
    [25, 56, 55, 58, 12, 82],
    [74, 33, 88, 23, 86, 59]
  ],
  legendSettings: {
    visible: true,
    position: 'Right',
    showLabel: true,
    height: "150"
  }
}
},
provide: {
  heatmap: [Tooltip, Legend]
},
}
</script>

```

```
{% previewsample "page.domainurl/code-snippet/heatmap-chart/getting-started-cs5" %}
```

### Add custom cell palette

The default palette settings of the heat map cells can be customized by using the [paletteSettings](#) property. Using the [palette](#) property in `paletteSettings` object, you can change the color set for the cells. You can change the color mode of the cells to fixed or gradient mode using the [type](#) property.

### ~/SRC/APP.VUE

```

<template>
  <ejs-heatmap id="heatmap" :dataSource='dataSource' :xAxis='xAxis'
  :yAxis='yAxis' :titleSettings='titleSettings'

```

```

:legendSettings='legendSettings' :cellSettings='cellSettings'
:paletteSettings='paletteSettings'></ejs-heatmap>
</template>
<script>
import { HeatMapComponent, Tooltip, Legend } from '@syncfusion/ej2-vue-heatmap';
export default {
  components: {
    'ejs-heatmap': HeatMapComponent
  },
  data: function() {
    return {
      xAxis: {
        labels: ['Nancy', 'Andrew', 'Janet', 'Margaret', 'Steven',
          'Michael', 'Robert', 'Laura', 'Anne', 'Paul',
          'Karin', 'Mario'],
      },
      yAxis: {
        labels: ['Mon', 'Tues', 'Wed', 'Thurs', 'Fri', 'Sat'],
      },
      cellSettings: {
        showLabel: true,
      },
      titleSettings: {
        text: 'Sales Revenue per Employee (in 1000 US$)',
        textStyle: {
          size: '15px',
          fontWeight: '500',
          fontStyle: 'Normal',
          fontFamily: 'Segoe UI'
        }
      },
      dataSource: [
        [73, 39, 26, 39, 94, 0],
        [93, 58, 53, 38, 26, 68],
        [99, 28, 22, 4, 66, 90],
        [14, 26, 97, 69, 69, 3],
        [7, 46, 47, 47, 88, 6],
        [41, 55, 73, 23, 3, 79],
        [56, 69, 21, 86, 3, 33],
        [45, 7, 53, 81, 95, 79],
        [60, 77, 74, 68, 88, 51],
        [25, 25, 10, 12, 78, 14],
        [25, 56, 55, 58, 12, 82],
        [74, 33, 88, 23, 86, 59]
      ],
      legendSettings: {
        visible: true,
        position: 'Right',
        showLabel: true,
        height: "150"
      },
      paletteSettings: {
        palette: [
          { value: 0, color: '#C06C84' },
          { value: 50, color: '#6C5B7B' },
          { value: 100, color: '#355C7D' }
        ]
      }
    }
  }
}

```



```

        ],
        type: "Gradient"
    }
}
},
provide:{
    heatmap:[Tooltip, Legend]
},
}
</script>

```

{% previewsample "page.domainurl/code-snippet/heatmap-chart/getting-started-cs6" %}

### Enable tooltip

The tooltip is used when you cannot display information by using the data labels due to space constraints. You can enable the tooltip by setting the [showTooltip](#) property to **true** and injecting the **Tooltip** module into the **provide**.

### ~/SRC/APP.VUE

```

<template>
    <ejs-heatmap id="heatmap" :dataSource='dataSource' :xAxis='xAxis'
    :yAxis='yAxis' :titleSettings='titleSettings'
    :legendSettings='legendSettings' :cellSettings='cellSettings'
    :showTooltip='showTooltip'></ejs-heatmap>
</template>
<script>
import { HeatMapComponent, Tooltip, Legend } from '@syncfusion/ej2-vue-heatmap';
export default {
    components: {
        'ejs-heatmap': HeatMapComponent
    },
    data: function() {
        return {
            xAxis: {
                labels: ['Nancy', 'Andrew', 'Janet', 'Margaret', 'Steven',
                    'Michael', 'Robert', 'Laura', 'Anne', 'Paul',
                    'Karin', 'Mario'],
            },
            yAxis: {
                labels: ['Mon', 'Tues', 'Wed', 'Thurs', 'Fri', 'Sat'],
            },
            cellSettings: {
                showLabel: true,
            },
            titleSettings: {
                text: 'Sales Revenue per Employee (in 1000 US$)',
                textStyle: {
                    size: '15px',
                    fontWeight: '500',
                    fontStyle: 'Normal',
                    fontFamily: 'Segoe UI'
                }
            }
        },
        dataSource: [

```

```

        [73, 39, 26, 39, 94, 0],
        [93, 58, 53, 38, 26, 68],
        [99, 28, 22, 4, 66, 90],
        [14, 26, 97, 69, 69, 3],
        [7, 46, 47, 47, 88, 6],
        [41, 55, 73, 23, 3, 79],
        [56, 69, 21, 86, 3, 33],
        [45, 7, 53, 81, 95, 79],
        [60, 77, 74, 68, 88, 51],
        [25, 25, 10, 12, 78, 14],
        [25, 56, 55, 58, 12, 82],
        [74, 33, 88, 23, 86, 59]
    ],
    legendSettings: {
        visible: true,
        position: 'Right',
        showLabel: true,
        height: "150"
    },
    showTooltip: true
}
},
provide: {
    heatmap: [Tooltip, Legend]
},
}
</script>

```

{% previewsample "page.domainurl/code-snippet/heatmap-chart/getting-started-cs7" %}

## Getting Started with the Vue HeatMap Component in Vue 3

This article provides a step-by-step guide for setting up a [Vite](#) project with a JavaScript environment and integrating the Syncfusion Vue HeatMap component using the [Composition API](#) / [Options API](#).

The **Composition API** is a new feature introduced in Vue.js 3 that provides an alternative way to organize and reuse component logic. It allows developers to write components as functions that use smaller, reusable functions called composition functions to manage their properties and behavior.

The **Options API** is the traditional way of writing Vue.js components, where the component logic is organized into a series of options that define the component's properties and behavior. These options include data, methods, computed properties, watchers, lifecycle hooks, and more.

### Prerequisites

#### [System requirements for Syncfusion Vue UI components](#)

#### Set up the Vite project

A recommended approach for beginning with Vue is to scaffold a project using [Vite](#). To create a new Vite project, use one of the commands that are specific to either NPM or Yarn.

```
`bash
```

```
npm create vite@latest
```

```
,
```

or

```
`bash
yarn create vite
`
```

Using one of the above commands will lead you to set up additional configurations for the project as below:

1. Define the project name: We can specify the name of the project directly. Let's specify the name of the project as `my-project` for this article.

```
`bash
? Project name: » my-project
`
```

2. Select `Vue` as the framework. It will create a Vue 3 project.

```
`bash
? Select a framework: » - Use arrow-keys. Return to submit.
Vanilla
Vue
React
Preact
Lit
Svelte
Others
`
```

3. Choose `JavaScript` as the framework variant to build this Vite project using JavaScript and Vue.

```
`bash
? Select a variant: » - Use arrow-keys. Return to submit.
JavaScript
TypeScript
Customize with create-vue ↗
Nuxt ↗
`
```

4. Upon completing the aforementioned steps to create the `my-project`, run the following command to install its dependencies:

```
`bash
cd my-project
npm install
```

```
,  
  
or  
  
`bash  
cd my-project  
yarn install  
,
```

Now that `my-project` is ready to run with default settings, let's add Syncfusion components to the project.

#### Add Syncfusion Vue packages

Syncfusion Vue component packages are available at [npmjs.com](https://www.npmjs.com). To use Syncfusion Vue components in the project, install the corresponding npm package.

This article uses the [Vue HeatMap component](#) as an example. To use the Vue HeatMap component in the project, the `@syncfusion/ej2-vue-heatmap` package needs to be installed using the following command:

```
`bash  
npm install @syncfusion/ej2-vue-heatmap --save  
,  
  
or  
  
`bash  
yarn add @syncfusion/ej2-vue-heatmap  
,
```

#### Add Syncfusion Vue component

Follow the below steps to add the Vue HeatMap component using `Composition API` or `Options API`:

1.First, import and register the HeatMap component and its child directives in the `script` section of the `src/App.vue` file. If you are using the `Composition API`, you should add the `setup` attribute to the `script` tag to indicate that Vue will be using the `Composition API`.

##### COMPOSITION API (~SRC/APP.VUE)

```
<script setup>  
import { HeatMapComponent as EjsHeatmap, Tooltip, Legend } from  
"@syncfusion/ej2-vue-heatmap";  
import { HeatMap } from '@syncfusion/ej2-heatmap';  
HeatMap.Inject(Legend, Tooltip);  
</script>
```

##### OPTIONS API (~SRC/APP.VUE)

```
<script>  
import { HeatMapComponent, Tooltip, Legend } from "@syncfusion/ej2-vue-  
heatmap";  
//Component registration
```

```
export default {
  name: "App",
  components: {
    'ejs-heatmap': HeatMapComponent
  }
}
</script>
```

2. In the **template** section, define the HeatMap component with the [dataSource](#), [xAxis](#), [yAxis](#) and other property definitions.

#### ~/SRC/APP.VUE

```
<template>
<ejs-heatmap id="heatmap" :dataSource='dataSource' :xAxis='xAxis'
:yAxis='yAxis' :titleSettings='titleSettings'
:legendSettings='legendSettings' :cellSettings='cellSettings'
:showTooltip='showTooltip'></ejs-heatmap>
</template>
```

3. In the **script** section, declare the values for the properties defined in the **template** section.

#### COMPOSITION API (~/SRC/APP.VUE)

```
<script setup>
const xAxis = {
  labels: ['Nancy', 'Andrew', 'Janet', 'Margaret', 'Steven', 'Michael',
    'Robert', 'Laura', 'Anne', 'Paul', 'Karin', 'Mario'],
};
const yAxis = {
  labels: ['Mon', 'Tues', 'Wed', 'Thurs', 'Fri', 'Sat'],
};
const cellSettings = {
  showLabel: true,
};
const titleSettings = {
  text: 'Sales Revenue per Employee (in 1000 US$)',
  textStyle: {
    size: '15px',
    fontWeight: '500',
    fontStyle: 'Normal',
    fontFamily: 'Segoe UI'
  }
};
const dataSource = [
  [73, 39, 26, 39, 94, 0],
  [93, 58, 53, 38, 26, 68],
  [99, 28, 22, 4, 66, 90],
  [14, 26, 97, 69, 69, 3],
  [7, 46, 47, 47, 88, 6],
  [41, 55, 73, 23, 3, 79],
  [56, 69, 21, 86, 3, 33],
  [45, 7, 53, 81, 95, 79],
  [60, 77, 74, 68, 88, 51],
  [25, 25, 10, 12, 78, 14],
  [25, 56, 55, 58, 12, 82],
```

```
[74, 33, 88, 23, 86, 59]
];
const legendSettings = {
  visible:true,
  position: 'Right',
  showLabel: true,
  height: "150"
};
const showTooltip = true;
</script>
```

### OPTIONS API (~SRC/APP.VUE)

```
<script>
data() {
  return {
    xAxis: {
      labels: ['Nancy', 'Andrew', 'Janet', 'Margaret', 'Steven', 'Michael',
        'Robert', 'Laura', 'Anne', 'Paul', 'Karin', 'Mario'],
    },
    yAxis:{
      labels: ['Mon', 'Tues', 'Wed', 'Thurs', 'Fri', 'Sat'],
    },
    cellSettings: {
      showLabel: true,
    },
    titleSettings: {
      text: 'Sales Revenue per Employee (in 1000 US$)',
      textStyle: {
        size: '15px',
        fontWeight: '500',
        fontStyle: 'Normal',
        fontFamily: 'Segoe UI'
      }
    },
    dataSource: [
      [73, 39, 26, 39, 94, 0],
      [93, 58, 53, 38, 26, 68],
      [99, 28, 22, 4, 66, 90],
      [14, 26, 97, 69, 69, 3],
      [7, 46, 47, 47, 88, 6],
      [41, 55, 73, 23, 3, 79],
      [56, 69, 21, 86, 3, 33],
      [45, 7, 53, 81, 95, 79],
      [60, 77, 74, 68, 88, 51],
      [25, 25, 10, 12, 78, 14],
      [25, 56, 55, 58, 12, 82],
      [74, 33, 88, 23, 86, 59]
    ],
    legendSettings: {
      visible:true,
      position: 'Right',
      showLabel: true,
      height: "150"
    },
    showTooltip:true
  }
}
```

```
};
}
</script>
```

Here is the summarized code for the above steps in the **src/App.vue** file:

### COMPOSITION API (~SRC/APP.VUE)

```
<template>
<ejs-heatmap id="heatmap" :dataSource='dataSource' :xAxis='xAxis'
: yAxis='yAxis' :titleSettings='titleSettings'
: legendSettings='legendSettings' :cellSettings='cellSettings'
: showTooltip='showTooltip'></ejs-heatmap>
</template>
<script setup>
import { HeatMapComponent as EjsHeatmap, Tooltip, Legend } from
"@syncfusion/ej2-vue-heatmap";
import { HeatMap } from "@syncfusion/ej2-heatmap";
HeatMap.Inject(Legend, Tooltip);
const xAxis = {
labels: ['Nancy', 'Andrew', 'Janet', 'Margaret', 'Steven', 'Michael',
'Robert', 'Laura', 'Anne', 'Paul', 'Karin', 'Mario'],
};
const yAxis = {
labels: ['Mon', 'Tues', 'Wed', 'Thurs', 'Fri', 'Sat'],
};
const cellSettings = {
showLabel: true,
};
const titleSettings = {
text: 'Sales Revenue per Employee (in 1000 US$)',
textStyle: {
size: '15px',
fontWeight: '500',
fontStyle: 'Normal',
fontFamily: 'Segoe UI'
}
};
const dataSource = [
[73, 39, 26, 39, 94, 0],
[93, 58, 53, 38, 26, 68],
[99, 28, 22, 4, 66, 90],
[14, 26, 97, 69, 69, 3],
[7, 46, 47, 47, 88, 6],
[41, 55, 73, 23, 3, 79],
[56, 69, 21, 86, 3, 33],
[45, 7, 53, 81, 95, 79],
[60, 77, 74, 68, 88, 51],
[25, 25, 10, 12, 78, 14],
[25, 56, 55, 58, 12, 82],
[74, 33, 88, 23, 86, 59]
];
const legendSettings = {
visible: true,
position: 'Right',
showLabel: true,
```

```
height: "150"
};
const showTooltip = true;
</script>
```

### OPTIONS API (~SRC/APP.VUE)

```
<template>
<ejs-heatmap id="heatmap" :dataSource='dataSource' :xAxis='xAxis'
:yAxis='yAxis' :titleSettings='titleSettings'
:legendSettings='legendSettings' :cellSettings='cellSettings'
:showTooltip='showTooltip'></ejs-heatmap>
</template>
<script>
import { HeatMapComponent, Tooltip, Legend } from "@syncfusion/ej2-vue-heatmap";
// Component registration
export default {
name: "App",
// Declaring component and its directives
components: {
'ejs-heatmap': HeatMapComponent
},
// Bound properties declarations
data() {
return {
xAxis: {
labels: ['Nancy', 'Andrew', 'Janet', 'Margaret', 'Steven', 'Michael',
'Robert', 'Laura', 'Anne', 'Paul', 'Karin', 'Mario'],
},
yAxis: {
labels: ['Mon', 'Tues', 'Wed', 'Thurs', 'Fri', 'Sat'],
},
cellSettings: {
showLabel: true,
},
titleSettings: {
text: 'Sales Revenue per Employee (in 1000 US$)',
textStyle: {
size: '15px',
fontWeight: '500',
fontStyle: 'Normal',
fontFamily: 'Segoe UI'
}
},
},
dataSource: [
[73, 39, 26, 39, 94, 0],
[93, 58, 53, 38, 26, 68],
[99, 28, 22, 4, 66, 90],
[14, 26, 97, 69, 69, 3],
[7, 46, 47, 47, 88, 6],
[41, 55, 73, 23, 3, 79],
[56, 69, 21, 86, 3, 33],
[45, 7, 53, 81, 95, 79],
[60, 77, 74, 68, 88, 51],
[25, 25, 10, 12, 78, 14],
```



```
[25, 56, 55, 58, 12, 82],
[74, 33, 88, 23, 86, 59]
],
legendSettings: {
  visible: true,
  position: 'Right',
  showLabel: true,
  height: "150"
},
showTooltip: true
};
},
provide: {
  heatmap: [Tooltip, Legend]
}
};
</script>
```

### Run the project

To run the project, use the following command:

```
`bash
```

```
npm run dev
```

```
`
```

or

```
`bash
```

```
yarn run dev
```

```
`
```

The output will appear as follows:



For migrating from Vue 2 to Vue 3, refer to the [migration](#) documentation.

See also

- [Getting Started with Vue UI Components using Composition API and TypeScript](#)
- [Getting Started with Vue UI Components using Options API and TypeScript](#)

### Working with data in Vue Heatmap chart component

Heat map visualizes the JSON data and two-dimensional array data. Using the data adaptor support, data can be bound to the heat map.

#### Data adaptor

Heat map supports the following types of data binding with the adaptor support.

- Array
- Table Binding
- Cell Binding
- JSON data
- Table Binding
- Cell Binding

#### Array - table binding

This data type is a collection of one dimensional array objects, at which each inner array contains data points for an X-axis data label. This is the default data binding type for heat map. You can also directly bind the array object to the [dataSource](#) property.

#### APP.VUE

```
<template>
  <div id="app">
    <ejs-heatmap id="heatmap" :titleSettings='titleSettings'
    :xAxis='xAxis' :yAxis='yAxis' :paletteSettings='paletteSettings'
    :legendSettings='legendSettings' :tooltipRender='tooltipRender'
    :dataSource='dataSource'></ejs-heatmap>
  </div>
</template>
<script>
import Vue from 'vue';
import { HeatMapPlugin, Tooltip, Legend } from '@syncfusion/ej2-vue-heatmap';
Vue.use(HeatMapPlugin);
export default {
  data: function() {
    return {
      dataSource: [
        [9.5, 2.2, 4.2, 8.2, -0.5, 3.2, 5.4, 7.4, 6.2, 1.4],
        [4.3, 8.9, 10.8, 6.5, 5.1, 6.2, 7.6, 7.5, 6.1, 7.6],
        [3.9, 2.7, 2.5, 3.7, 2.6, 5.1, 5.8, 2.9, 4.5, 5.1],
        [2.4, -3.7, 4.1, 6.0, 5.0, 2.4, 3.3, 4.6, 4.3, 2.7],
        [2.0, 7.0, -4.1, 8.9, 2.7, 5.9, 5.6, 1.9, -1.7, 2.9],
        [5.4, 1.1, 6.9, 4.5, 2.9, 3.4, 1.5, -2.8, -4.6, 1.2],
        [-1.3, 3.9, 3.5, 6.6, 5.2, 7.7, 1.4, -3.6, 6.6, 4.3],
        [-1.6, 2.3, 2.9, -2.5, 1.3, 4.9, 10.1, 3.2, 4.8, 2.0],
        [10.8, -1.6, 4.0, 6.0, 7.7, 2.6, 5.6, -2.5, 3.8, -1.9],
        [6.2, 9.8, -1.5, 2.0, -1.5, 4.3, 6.7, 3.8, -1.2, 2.4],
      ]
    }
  }
}
```

```

        [1.2, 10.9, 4.0, -1.4, 2.2, 1.6, -2.6, 2.3, 1.7, 2.4],
        [5.1, -2.4, 8.2, -1.1, 3.5, 6.0, -1.3, 7.2, 9.0, 4.2]
    ],
    titleSettings: {
        text: 'GDP Growth Rate for Major Economies (in Percentage)',
        textStyle: {
            size: '15px',
            fontWeight: '500',
            fontStyle: 'Normal',
            fontFamily: 'Segoe UI'
        }
    },
    xAxis: {
        labels: ['China', 'India', 'Australia', 'Mexico', 'Canada',
        'Brazil',
            'USA', 'UK', 'Germany', 'Russia', 'France', 'Japan'],
        labelRotation: 45,
        labelIntersectAction: 'None'
    },
    yAxis: {
        labels: ['2008', '2009', '2010', '2011', '2012', '2013',
        '2014', '2015', '2016', '2017']
    },
    paletteSettings: {
        palette: [
            { value: -1, color: '#F0D6AD' },
            { value: 0, color: '#9da49a' },
            { value: 3.5, color: '#d7c7a7' },
            { value: 6.0, color: '#6e888f' },
            { value: 7.5, color: '#466f86' },
            { value: 10, color: '#19547B' },
        ]
    },
    legendSettings: {
        visible: false
    }
},
provide: {
    heatmap: [Tooltip, Legend]
},
methods: {
    tooltipRender: function(args)
    {
        args.content = [args.yLabel + ' | ' + args.xLabel + ' : ' +
args.value + ' %'];
    }
}
}
</script>
<style>
@import
'https://ej2.syncfusion.com/vue/documentation/node_modules/@syncfusion/ej2-
vue-heatmap/styles/material.css';
</style>

```

```
{% previewsample "page.domainurl/code-snippet/heatmap-chart/working-with-data-cs1" %}
```

### Array - cell binding

This data type is a collection of array objects that contain information about the row index, column index, and data value for each cell. You can bind the data to heat map by using the [data](#) property in the [dataSource](#) and setting the [adaptorType](#) property to **Cell**.

### APP.VUE

```
<template>
  <div id="app">
    <ejs-heatmap id="heatmap" :titleSettings='titleSettings'
    :xAxis='xAxis' :yAxis='yAxis' :paletteSettings='paletteSettings'
    :legendSettings='legendSettings' :tooltipRender='tooltipRender'
    :dataSource='dataSource' :cellSettings='cellSettings'></ejs-heatmap>
  </div>
</template>
<script>
import Vue from 'vue';
import { HeatMapPlugin, Tooltip, Legend, Adaptor } from '@syncfusion/ej2-
vue-heatmap';
Vue.use(HeatMapPlugin);
export default {
  data: function() {
    return {
      dataSource: [
        [0, 0, 10.75], [0, 1, 14.5], [0, 2, 25.5], [0, 3, 39.5],
        [0, 4, 59.75], [0, 5, 35.50], [0, 6, 75.5],
        [1, 0, 20.75], [1, 1, 35.5], [1, 2, 29.5], [1, 3, 75.5],
        [1, 4, 80], [1, 5, 65], [1, 6, 85],
        [2, 0, 6], [2, 1, 18.5], [2, 2, 30.05], [2, 3, 35.5],
        [2, 4, 40.75], [2, 5, 50.75], [2, 6, 65],
        [3, 0, 30.5], [3, 1, 20.5], [3, 2, 45.30], [3, 3, 50],
        [3, 4, 55], [3, 5, 85.80], [3, 6, 87.5],
        [4, 0, 10.5], [4, 1, 20.75], [4, 2, 35.5], [4, 3, 35.5],
        [4, 4, 45.5], [4, 5, 65.], [4, 6, 75.5],
        [5, 0, 45.5], [5, 1, 20.75], [5, 2, 45.5], [5, 3,
        50.75], [5, 4, 79.30], [5, 5, 84.20], [5, 6, 87.36],
        [6, 0, 26.82], [6, 1, 70], [6, 2, 75], [6, 3, 79.5], [6,
        4, 88.5], [6, 5, 89.5], [6, 6, 91.75],
        [7, 0, 15.75], [7, 1, 20.75], [7, 2, 25.5], [7, 3,
        42.35], [7, 4, 45.15], [7, 5, 76.5], [7, 6, 80.5],
        [8, 0, 1.98], [8, 1, 15.23], [8, 2, 43], [8, 3, 49], [8,
        4, 63.80], [8, 5, 67.97], [8, 6, 70.52],
        [9, 0, 14.31], [9, 1, 42.87], [9, 2, 77.28], [9, 3,
        77.82], [9, 4, 81.44], [9, 5, 81.92], [9, 6, 83.75],
        [10, 0, 25.5], [10, 1, 35.5], [10, 2, 40.5], [10, 3,
        45.05], [10, 4, 50.5], [10, 5, 75.5], [10, 6, 90.58]
      ],
      dataSourceSettings: {
        isJsonData: false,
        adaptorType: 'Cell'
      },
      titleSettings: {
        text: 'Percentage of Individuals Using Internet by Country',
        textStyle: {
          size: '15px',

```

```

        fontWeight: '500',
        fontStyle: 'Normal',
        fontFamily: 'Segoe UI'
    },
    },
    xAxis: {
        labels: ['China', 'Australia', 'Mexico', 'Canada',
'Brazil', 'USA',
        'UK', 'Germany', 'Russia', 'France', 'Japan']
    },
    yAxis: {
        labels: ['2000', '2005', '2010', '2011', '2012', '2013',
'2014']
    },
    cellSettings: {
        border: {
            width: 0
        },
        textStyle: {
            color: 'white'
        },
        format: '{value} %'
    },
    legendSettings: {
        visible: false,
    },
    paletteSettings: {
        palette: [
            { color: '#3498DB' },
            { color: '#2C3E50' }
        ]
    },
    legendSettings: {
        visible: false
    }
    },
    provide:{
        heatmap:[Tooltip, Legend, Adaptor]
    },
    methods: {
        tooltipRender: function(args)
        {
            args.content = [args.yLabel + ' | ' + args.xLabel + ' : ' +
args.value + ' %'];
        }
    }
}
</script>
<style>
@import
'https://ej2.syncfusion.com/vue/documentation/node_modules/@syncfusion/ej2-
vue-heatmap/styles/material.css';
</style>

```

{% previewsample "page.domainurl/code-snippet/heatmap-chart/working-with-data-cs2" %}

*JSON data - table binding*

In JSON table data binding, each JSON object contains an X-axis data point as row header and all the corresponding Y-axis data values. You can bind the JSON table data to the heat map using the [data](#) property in [dataSource](#). To achieve this, you should enable the [isJsonData](#) property and define the [adaptorType](#) property as **Table**. The [xDataMapping](#) property is used to map the row header in JSON data.

**APP.VUE**

```
<template>
  <div id="app">
    <ejs-heatmap id="heatmap" :titleSettings='titleSettings'
    :xAxis='xAxis' :yAxis='yAxis' :paletteSettings='paletteSettings'
    :dataSource='dataSource' :dataSourceSettings='dataSourceSettings'
    :cellSettings='cellSettings'></ejs-heatmap>
  </div>
</template>
<script>
import Vue from 'vue';
import { HeatMapPlugin, Tooltip, Legend, Adaptor } from '@syncfusion/ej2-
vue-heatmap';
Vue.use(HeatMapPlugin);
export default {
  data: function() {
    return {
      dataSource: [
        { 'Region': 'USA', '2000': 93, '2004': 101, '2008': 112,
'2012': 103, '2016': 121 },
        { 'Region': 'GBR', '2000': 28, '2004': 30, '2008': 49,
'2012': 65, '2016': 67 },
        { 'Region': 'China', '2000': 58, '2004': 63, '2008':
100, '2012': 91, '2016': 70 },
        { 'Region': 'Russia', '2000': 89, '2004': 90, '2008':
60, '2012': 69, '2016': 55 },
        { 'Region': 'Germany', '2000': 56, '2004': 49, '2008':
41, '2012': 44, '2016': 42 },
        { 'Region': 'Japan', '2000': 18, '2004': 37, '2008': 25,
'2012': 38, '2016': 41 },
        { 'Region': 'France', '2000': 38, '2004': 33, '2008':
43, '2012': 35, '2016': 42 },
        { 'Region': 'KOR', '2000': 28, '2004': 30, '2008': 32,
'2012': 30, '2016': 21 },
        { 'Region': 'Italy', '2000': 34, '2004': 32, '2008': 27,
'2012': 28, '2016': 28 }
      ],
      dataSourceSettings: {
        isJsonData: true,
        adaptorType: 'Table',
        xDataMapping: 'Region'
      },
      titleSettings: {
        text: 'Olympic Medal Achievements of most Successful
Countries',
        textStyle: {
          size: '15px',
          fontWeight: '500',

```

```

        fontStyle: 'Normal',
        fontFamily: 'Segoe UI'
      },
      },
      xAxis: {
        labels: ['China', 'France', 'GBR', 'Germany', 'Italy',
'Japan', 'KOR', 'Russia', 'USA'],
        labelRotation: 45,
        labelIntersectAction: 'None'
      },
      yAxis: {
        title : {text: 'Olympic Year'},
        labels: ['2000', '2004', '2008', '2012', '2016'],
      },
      cellSettings: {
        border: {
          width: 1,
          radius: 4,
          color: 'white'
        }
      },
      paletteSettings: {
        palette: [
          { color: '#F0C27B' },
          { color: '#4B1248' }
        ]
      },
    },
    provide:{
      heatmap:[Tooltip, Legend, Adaptor]
    }
  }
</script>
<style>
@import
'https://ej2.syncfusion.com/vue/documentation/node_modules/@syncfusion/ej2-
vue-heatmap/styles/material.css';
</style>

```

{% previewsample "page.domainurl/code-snippet/heatmap-chart/working-with-data-cs3" %}

### JSON data - Cell binding

In JSON cell data binding, each JSON object consists a value for each cell along with a mapping value for row and column. You can bind the JSON cell data having information for each cell to the heat map using the [data](#) property in [dataSource](#). To achieve this, you should define the [adaptorType](#) property as **Cell**, and enable the [isJsonData](#) property. Now, map the fields of data by using the [valueMapping](#), [xDataMapping](#) and [yDataMapping](#) properties.

### APP.VUE

```

<template>
  <div id="app">
    <ejs-heatmap id="heatmap" :titleSettings='titleSettings'
:xAxis='xAxis' :yAxis='yAxis' :paletteSettings='paletteSettings'

```

```

:dataSource='dataSource' :dataSourceSettings='dataSourceSettings'
:cellSettings='cellSettings'></ejs-heatmap>
</div>
</template>
<script>
import Vue from 'vue';
import { HeatMapPlugin, Tooltip, Legend, Adaptor } from '@syncfusion/ej2-vue-heatmap';
Vue.use(HeatMapPlugin);
export default {
  data: function() {
    return {
      dataSource: [
        { 'rowid': 'France', 'columnid': '2010', 'value': '77.6' },
        { 'rowid': 'France', 'columnid': '2011', 'value': '79.4' },
        { 'rowid': 'France', 'columnid': '2012', 'value': '80.8' },
        { 'rowid': 'France', 'columnid': '2013', 'value': '86.6' },
        { 'rowid': 'France', 'columnid': '2014', 'value': '83.7' },
        { 'rowid': 'France', 'columnid': '2015', 'value': '84.5' },
        { 'rowid': 'France', 'columnid': '2016', 'value': '82.6' },
        { 'rowid': 'USA', 'columnid': '2010', 'value': '60.6' },
        { 'rowid': 'USA', 'columnid': '2011', 'value': '65.4' },
        { 'rowid': 'USA', 'columnid': '2012', 'value': '70.8' },
        { 'rowid': 'USA', 'columnid': '2013', 'value': '73.8' },
        { 'rowid': 'USA', 'columnid': '2014', 'value': '75.3' },
        { 'rowid': 'USA', 'columnid': '2015', 'value': '77.5' },
        { 'rowid': 'USA', 'columnid': '2016', 'value': '77.6' },
        { 'rowid': 'Spain', 'columnid': '2010', 'value': '64.9' },
        { 'rowid': 'Spain', 'columnid': '2011', 'value': '52.6' },
        { 'rowid': 'Spain', 'columnid': '2012', 'value': '60.8' },
        { 'rowid': 'Spain', 'columnid': '2013', 'value': '65.6' },
        { 'rowid': 'Spain', 'columnid': '2014', 'value': '52.6' },
        { 'rowid': 'Spain', 'columnid': '2015', 'value': '68.5' },
        { 'rowid': 'Spain', 'columnid': '2016', 'value': '75.6' },
        { 'rowid': 'China', 'columnid': '2010', 'value': '55.6' },
        { 'rowid': 'China', 'columnid': '2011', 'value': '52.3' },
        { 'rowid': 'China', 'columnid': '2012', 'value': '54.8' },
        { 'rowid': 'China', 'columnid': '2013', 'value': '51.1' }
      ]
    };
  }
};

```



```

    { 'rowid': 'China', 'columnid': '2014', 'value': '55.6'
  },
    { 'rowid': 'China', 'columnid': '2015', 'value': '56.9'
  },
    { 'rowid': 'China', 'columnid': '2016', 'value': '59.3'
  },
    { 'rowid': 'Italy', 'columnid': '2010', 'value': '43.6'
  },
    { 'rowid': 'Italy', 'columnid': '2011', 'value': '43.2'
  },
    { 'rowid': 'Italy', 'columnid': '2012', 'value': '55.8'
  },
    { 'rowid': 'Italy', 'columnid': '2013', 'value': '50.1'
  },
    { 'rowid': 'Italy', 'columnid': '2014', 'value': '48.5'
  },
    { 'rowid': 'Italy', 'columnid': '2015', 'value': '50.7'
  },
    { 'rowid': 'Italy', 'columnid': '2016', 'value': '52.4'
  },
    { 'rowid': 'UK', 'columnid': '2010', 'value': '28.2' },
    { 'rowid': 'UK', 'columnid': '2011', 'value': '31.6' },
    { 'rowid': 'UK', 'columnid': '2012', 'value': '29.8' },
    { 'rowid': 'UK', 'columnid': '2013', 'value': '33.1' },
    { 'rowid': 'UK', 'columnid': '2014', 'value': '32.6' },
    { 'rowid': 'UK', 'columnid': '2015', 'value': '34.4' },
    { 'rowid': 'UK', 'columnid': '2016', 'value': '35.8' },
    { 'rowid': 'Germany', 'columnid': '2010', 'value':
      '26.8' },
    { 'rowid': 'Germany', 'columnid': '2011', 'value': '29'
  },
    { 'rowid': 'Germany', 'columnid': '2012', 'value':
      '26.8' },
    { 'rowid': 'Germany', 'columnid': '2013', 'value':
      '27.6' },
    { 'rowid': 'Germany', 'columnid': '2014', 'value': '33'
  },
    { 'rowid': 'Germany', 'columnid': '2015', 'value': '35'
  },
    { 'rowid': 'Germany', 'columnid': '2016', 'value':
      '35.6' },
    { 'rowid': 'Mexico', 'columnid': '2010', 'value': '23.2'
  },
    { 'rowid': 'Mexico', 'columnid': '2011', 'value': '24.9'
  },
    { 'rowid': 'Mexico', 'columnid': '2012', 'value': '30.1'
  },
    { 'rowid': 'Mexico', 'columnid': '2013', 'value': '22.2'
  },
    { 'rowid': 'Mexico', 'columnid': '2014', 'value': '29.3'
  },
    { 'rowid': 'Mexico', 'columnid': '2015', 'value': '32.1'
  },
    { 'rowid': 'Mexico', 'columnid': '2016', 'value': '35'
  },
    { 'rowid': 'Thailand', 'columnid': '2010', 'value':
      '15.9' },

```

```

        { 'rowid': 'Thailand', 'columnid': '2011', 'value':
'19.8' },
        { 'rowid': 'Thailand', 'columnid': '2012', 'value':
'21.8' },
        { 'rowid': 'Thailand', 'columnid': '2013', 'value':
'23.5' },
        { 'rowid': 'Thailand', 'columnid': '2014', 'value':
'24.8' },
        { 'rowid': 'Thailand', 'columnid': '2015', 'value':
'29.9' },
        { 'rowid': 'Thailand', 'columnid': '2016', 'value':
'32.6' },
    },
        { 'rowid': 'Austria', 'columnid': '2010', 'value': '22'
},
        { 'rowid': 'Austria', 'columnid': '2011', 'value':
'21.3' },
        { 'rowid': 'Austria', 'columnid': '2012', 'value':
'24.2' },
        { 'rowid': 'Austria', 'columnid': '2013', 'value':
'23.2' },
        { 'rowid': 'Austria', 'columnid': '2014', 'value': '25'
},
        { 'rowid': 'Austria', 'columnid': '2015', 'value':
'26.7' },
        { 'rowid': 'Austria', 'columnid': '2016', 'value':
'28.1' }
    ],
    dataSourceSettings: {
        isJsonData: true,
        adaptorType: 'Cell',
        xDataMapping: 'rowid',
        yDataMapping: 'columnid',
        valueMapping: 'value'
    },
    titleSettings: {
        text: 'Most Visited Destinations by International Tourist
Arrivals',
        textStyle: {
            size: '15px',
            fontWeight: '500',
            fontStyle: 'Normal',
            fontFamily: 'Segoe UI'
        }
    },
    xAxis: {
        labels: ['Austria', 'China', 'France', 'Germany', 'Italy',
'Mexico', 'Spain', 'Thailand', 'UK', 'USA']
    },
    yAxis: {
        labels: ['2010', '2011', '2012', '2013', '2014', '2015',
'2016']
    },
    cellSettings: {
        border: {
            radius: 4,
            width: 1,
            color: 'white'
        }
    }
}

```

```

        },
        showLabel: true,
        format: '{value} M'
    },
    paletteSettings: {
        palette: [
            { color: '#DCD57E' },
            { color: '#A6DC7E' },
            { color: '#7EDCA2' },
            { color: '#6EB5D0' }
        ]
    },
    },
    },
    provide: {
        heatmap: [Tooltip, Legend, Adaptor]
    }
}
</script>
<style>
@import
'https://ej2.syncfusion.com/vue/documentation/node_modules/@syncfusion/ej2-
vue-heatmap/styles/material.css';
</style>

```

{% previewsample "page.domainurl/code-snippet/heatmap-chart/working-with-data-cs4" %}

### Empty points

The data points that use the **null** or **undefined** or empty string as value are considered as empty points. Empty data points are ignored and not displayed in the heat map, and these points are rendered with default palette. You can customize the empty data point color value using the [emptyPointColor](#) property.

### APP.VUE

```

<template>
  <div id="app">
    <ejs-heatmap id="heatmap" :xAxis='xAxis' :yAxis='yAxis'
    :dataSource='dataSource' :legendSettings='legendSettings'></ejs-heatmap>
  </div>
</template>
<script>
import Vue from 'vue';
import { HeatMapPlugin, Tooltip, Legend, Adaptor } from '@syncfusion/ej2-
vue-heatmap';
Vue.use(HeatMapPlugin);
export default {
  data: function() {
    return {
      dataSource: [
        [73, 39, 26, 39, 94, 0],
        [93, 58, 53, 38, 26, 68],
        [99, 28, null, 4, 66, 90],
        [14, 26, 97, 69, 69, 3],
        [7, 46, 47, 47, 88, 6],
        [41, 55, 73, 23, "", 79],

```

```

        [56, 69, 21, 86, 3, 33],
        [45, 7, undefined, 81, 95, 79],
        [60, 77, 74, 68, 88, 51],
        [25, 25, 10, 12, 78, 14],
        [25, 56, 55, 58, 12, 82],
        [74, 33, 88, 23, 86, 59]
    ],
    xAxis: {
        valueType: "DateTime",
        minimum: new Date(2007, 0, 1),
        intervalType: "Years",
        labelFormat: "YYYY"
    },
    yAxis: {
        valueType: "Numeric"
    },
    legendSettings: {
        visible: false,
    },
    },
    provide: {
        heatmap: [Tooltip, Legend, Adaptor]
    }
}
</script>
<style>
@import
'https://ej2.syncfusion.com/vue/documentation/node_modules/@syncfusion/ej2-
vue-heatmap/styles/material.css';
</style>

```

{% previewsample "page.domainurl/code-snippet/heatmap-chart/rendering-mode-cs2" %}

### Binding nested JSON data

In complex data binding, you can bind the nested JSON data to the data points in the heat map.

The nested data can be mapped using the [xDataMapping](#), [yDataMapping](#), [valueMapping](#) and [bubbleDataMapping](#) properties as string value concatenated by a dot.

### APP.VUE

```

<template>
  <div id="app">
    <ejs-heatmap id="heatmap" :titleSettings='titleSettings'
    :xAxis='xAxis' :yAxis='yAxis' :paletteSettings='paletteSettings'
    :dataSource='dataSource':dataSourceSettings='dataSourceSettings'
    :cellSettings='cellSettings'></ejs-heatmap>
  </div>
</template>
<script>
import Vue from 'vue';
import { HeatMapPlugin, Tooltip, Legend, Adaptor } from '@syncfusion/ej2-
vue-heatmap';
Vue.use(HeatMapPlugin);
export default {

```

```

data: function() {
  return {
    dataSource: [
      { 'Labels':{ 'Xlabel': 'France', 'Ylabel': '2010' }, 'data':{ 'value': '77.6' } },
      { 'Labels':{ 'Xlabel': 'France', 'Ylabel': '2011' }, 'data':{ 'value': '79.4' } },
      { 'Labels':{ 'Xlabel': 'France', 'Ylabel': '2012' }, 'data':{ 'value': '80.8' } },
      { 'Labels':{ 'Xlabel': 'France', 'Ylabel': '2013' }, 'data':{ 'value': '86.6' } },
      { 'Labels':{ 'Xlabel': 'France', 'Ylabel': '2014' }, 'data':{ 'value': '83.7' } },
      { 'Labels':{ 'Xlabel': 'France', 'Ylabel': '2015' }, 'data':{ 'value': '84.5' } },
      { 'Labels':{ 'Xlabel': 'France', 'Ylabel': '2016' }, 'data':{ 'value': '82.6' } },
      { 'Labels':{ 'Xlabel': 'USA', 'Ylabel': '2010' }, 'data':{ 'value': '60.6' } },
      { 'Labels':{ 'Xlabel': 'USA', 'Ylabel': '2011' }, 'data':{ 'value': '65.4' } },
      { 'Labels':{ 'Xlabel': 'USA', 'Ylabel': '2012' }, 'data':{ 'value': '70.8' } },
      { 'Labels':{ 'Xlabel': 'USA', 'Ylabel': '2013' }, 'data':{ 'value': '73.8' } },
      { 'Labels':{ 'Xlabel': 'USA', 'Ylabel': '2014' }, 'data':{ 'value': '75.3' } },
      { 'Labels':{ 'Xlabel': 'USA', 'Ylabel': '2015' }, 'data':{ 'value': '77.5' } },
      { 'Labels':{ 'Xlabel': 'USA', 'Ylabel': '2016' }, 'data':{ 'value': '77.6' } },
      { 'Labels':{ 'Xlabel': 'Spain', 'Ylabel': '2010' }, 'data':{ 'value': '64.9' } },
      { 'Labels':{ 'Xlabel': 'Spain', 'Ylabel': '2011' }, 'data':{ 'value': '52.6' } },
      { 'Labels':{ 'Xlabel': 'Spain', 'Ylabel': '2012' }, 'data':{ 'value': '60.8' } },
      { 'Labels':{ 'Xlabel': 'Spain', 'Ylabel': '2013' }, 'data':{ 'value': '65.6' } },
      { 'Labels':{ 'Xlabel': 'Spain', 'Ylabel': '2014' }, 'data':{ 'value': '52.6' } },
      { 'Labels':{ 'Xlabel': 'Spain', 'Ylabel': '2015' }, 'data':{ 'value': '68.5' } },
      { 'Labels':{ 'Xlabel': 'Spain', 'Ylabel': '2016' }, 'data':{ 'value': '75.6' } },
      { 'Labels':{ 'Xlabel': 'China', 'Ylabel': '2010' }, 'data':{ 'value': '55.6' } },
      { 'Labels':{ 'Xlabel': 'China', 'Ylabel': '2011' }, 'data':{ 'value': '52.3' } },
      { 'Labels':{ 'Xlabel': 'China', 'Ylabel': '2012' }, 'data':{ 'value': '54.8' } },
      { 'Labels':{ 'Xlabel': 'China', 'Ylabel': '2013' }, 'data':{ 'value': '51.1' } },
      { 'Labels':{ 'Xlabel': 'China', 'Ylabel': '2014' }, 'data':{ 'value': '55.6' } },
      { 'Labels':{ 'Xlabel': 'China', 'Ylabel': '2015' }, 'data':{ 'value': '56.9' } },
      { 'Labels':{ 'Xlabel': 'China', 'Ylabel': '2016' }, 'data':{ 'value': '59.3' } },
      { 'Labels':{ 'Xlabel': 'Italy', 'Ylabel': '2010' }, 'data':{ 'value': '43.6' } },
      { 'Labels':{ 'Xlabel': 'Italy', 'Ylabel': '2011' }, 'data':{ 'value': '43.2' } },
      { 'Labels':{ 'Xlabel': 'Italy', 'Ylabel': '2012' }, 'data':{ 'value': '55.8' } },
      { 'Labels':{ 'Xlabel': 'Italy', 'Ylabel': '2013' }, 'data':{ 'value': '50.1' } },
      { 'Labels':{ 'Xlabel': 'Italy', 'Ylabel': '2014' }, 'data':{ 'value': '48.5' } },
      { 'Labels':{ 'Xlabel': 'Italy', 'Ylabel': '2015' }, 'data':{ 'value': '50.7' } },
      { 'Labels':{ 'Xlabel': 'Italy', 'Ylabel': '2016' }, 'data':{ 'value': '52.4' } },
      { 'Labels':{ 'Xlabel': 'UK', 'Ylabel': '2010' }, 'data':{ 'value': '28.2' } },
      { 'Labels':{ 'Xlabel': 'UK', 'Ylabel': '2011' }, 'data':{ 'value': '31.6' } },
      { 'Labels':{ 'Xlabel': 'UK', 'Ylabel': '2012' }, 'data':{ 'value': '29.8' } },
      { 'Labels':{ 'Xlabel': 'UK', 'Ylabel': '2013' }, 'data':{ 'value': '33.1' } },
      { 'Labels':{ 'Xlabel': 'UK', 'Ylabel': '2014' }, 'data':{ 'value': '32.6' } },
      { 'Labels':{ 'Xlabel': 'UK', 'Ylabel': '2015' }, 'data':{ 'value': '34.4' } },
      { 'Labels':{ 'Xlabel': 'UK', 'Ylabel': '2016' }, 'data':{ 'value': '35.8' } },
      { 'Labels':{ 'Xlabel': 'Germany', 'Ylabel': '2010' }, 'data':{ 'value': '26.8' } },
      { 'Labels':{ 'Xlabel': 'Germany', 'Ylabel': '2011' }, 'data':{ 'value': '29' } },
      { 'Labels':{ 'Xlabel': 'Germany', 'Ylabel': '2012' }, 'data':{ 'value': '26.8' } },
      { 'Labels':{ 'Xlabel': 'Germany', 'Ylabel': '2013' }, 'data':{ 'value': '27.6' } },
      { 'Labels':{ 'Xlabel': 'Germany', 'Ylabel': '2014' }, 'data':{ 'value': '33' } },
      { 'Labels':{ 'Xlabel': 'Germany', 'Ylabel': '2015' }, 'data':{ 'value': '35' } },
      { 'Labels':{ 'Xlabel': 'Germany', 'Ylabel': '2016' }, 'data':{ 'value': '35.6' } },
      { 'Labels':{ 'Xlabel': 'Mexico', 'Ylabel': '2010' }, 'data':{ 'value': '23.2' } },
      { 'Labels':{ 'Xlabel': 'Mexico', 'Ylabel': '2011' }, 'data':{ 'value': '24.9' } },
      { 'Labels':{ 'Xlabel': 'Mexico', 'Ylabel': '2012' }, 'data':{ 'value': '30.1' } },
      { 'Labels':{ 'Xlabel': 'Mexico', 'Ylabel': '2013' }, 'data':{ 'value': '22.2' } },
    ]
  }
}

```

```

{ 'Labels':{ 'Xlabel': 'Mexico', 'Ylabel': '2014'}, 'data':{ 'value': '29.3' }},
{ 'Labels':{ 'Xlabel': 'Mexico', 'Ylabel': '2015'}, 'data':{ 'value': '32.1' }},
{ 'Labels':{ 'Xlabel': 'Mexico', 'Ylabel': '2016'}, 'data':{ 'value': '35' }},
{ 'Labels':{ 'Xlabel': 'Thailand', 'Ylabel': '2010'}, 'data':{ 'value':
'15.9' }},
{ 'Labels':{ 'Xlabel': 'Thailand', 'Ylabel': '2011'}, 'data':{ 'value':
'19.8' }},
{ 'Labels':{ 'Xlabel': 'Thailand', 'Ylabel': '2012'}, 'data':{ 'value':
'21.8' }},
{ 'Labels':{ 'Xlabel': 'Thailand', 'Ylabel': '2013'}, 'data':{ 'value':
'23.5' }},
{ 'Labels':{ 'Xlabel': 'Thailand', 'Ylabel': '2014'}, 'data':{ 'value':
'24.8' }},
{ 'Labels':{ 'Xlabel': 'Thailand', 'Ylabel': '2015'}, 'data':{ 'value':
'29.9' }},
{ 'Labels':{ 'Xlabel': 'Thailand', 'Ylabel': '2016'}, 'data':{ 'value':
'32.6' }},
{ 'Labels':{ 'Xlabel': 'Austria', 'Ylabel': '2010'}, 'data':{ 'value': '22' }},
{ 'Labels':{ 'Xlabel': 'Austria', 'Ylabel': '2011'}, 'data':{ 'value': '21.3' }},
{ 'Labels':{ 'Xlabel': 'Austria', 'Ylabel': '2012'}, 'data':{ 'value': '24.2' }},
{ 'Labels':{ 'Xlabel': 'Austria', 'Ylabel': '2013'}, 'data':{ 'value': '23.2' }},
{ 'Labels':{ 'Xlabel': 'Austria', 'Ylabel': '2014'}, 'data':{ 'value': '25' }},
{ 'Labels':{ 'Xlabel': 'Austria', 'Ylabel': '2015'}, 'data':{ 'value': '26.7'
}},
{ 'Labels':{ 'Xlabel': 'Austria', 'Ylabel': '2016'}, 'data':{ 'value': '28.1' }}
],
    dataSourceSettings: {
        isJsonData: true,
        adaptorType: 'Cell',
        xDataMapping: 'Labels.Xlabel',
        yDataMapping: 'Labels.Ylabel',
        valueMapping: 'data.value'
    },
    titleSettings: {
        text: 'Most Visited Destinations by International Tourist
Arrivals',
        textStyle: {
            size: '15px',
            fontWeight: '500',
            fontStyle: 'Normal',
            fontFamily: 'Segoe UI'
        }
    },
    xAxis: {
        labels: ['Austria', 'China', 'France', 'Germany', 'Italy',
'Mexico', 'Spain', 'Thailand', 'UK', 'USA']
    },
    yAxis: {
        labels: ['2010', '2011', '2012', '2013', '2014', '2015',
'2016']
    },
    cellSettings: {
        border: {
            radius: 4,
            width: 1,
            color: 'white'
        }
    },

```

```

        showLabel: true,
        format: '{value} M'
      },
      paletteSettings: {
        palette: [
          { color: '#DCD57E' },
          { color: '#A6DC7E' },
          { color: '#7EDCA2' },
          { color: '#6EB5D0' }
        ]
      },
    },
    provide: {
      heatmap: [Tooltip, Legend, Adaptor]
    }
  }
</script>
<style>
@import '../node_modules/@syncfusion/ej2-vue-heatmap/styles/material.css';
</style>

```

{% previewsample "page.domainurl/code-snippet/heatmap-chart/working-with-data-cs5" %}

See Also

- [To bind data for bubble heat map with size and color attributes](#)

### Bubble heatmap in Vue Heatmap chart component

Data points represent the data source values with **gradient** or **fixed** colors in the HeatMap. You can customize the appearance of these data points by changing the **color** and **size** attributes.

The data points can be represented in color fill or bubble shape by defining the **tileType** property. By default, the data points are color filled with gradient or fixed colors and this depiction of data points is defined as **Rect** in the **tileType** property.

The cell customizations and color mapping for rect tile type is defined in [appearance](#) and [palette](#) sections in detail.

#### Bubble types

The data points can be represented in the bubble along with its attributes by setting the **tileType** property to **Bubble**.

In bubble HeatMap, you can display the data points with bubble size, bubble colors, and sector attributes of the bubble.

#### Bubble size

In this bubble HeatMap type, the size factor of the bubble is used to denote the data variations. The radius of the bubble varies according to data values.

By default, the bubble with small size denotes the data value with small magnitude and the larger bubble size denotes the data value with larger magnitude. This behavior can be inverted by using the [isInversedbubblesize](#) property.

To render a bubble HeatMap with size series, set the [bubbleType](#) property to **Size**.

### APP.VUE

```
<template>
  <div id="app">
    <ejs-heatmap id="heatmap" :titleSettings='titleSettings'
    :xAxis='xAxis' :yAxis='yAxis' :dataSource='dataSource'
    :paletteSettings='paletteSettings' :cellSettings='cellSettings'
    :legendSettings='legendSettings'></ejs-heatmap>
  </div>
</template>
<script>
import Vue from 'vue';
import { HeatMapPlugin, Legend, Adaptor, Tooltip } from '@syncfusion/ej2-
vue-heatmap';
Vue.use(HeatMapPlugin);
export default {
  data: function() {
    return {
      dataSource: [
        [73, 39, 26, 39, 94, 0],
        [93, 58, 53, 38, 26, 68],
        [99, 28, 22, 4, 66, 90],
        [14, 26, 97, 69, 69, 3],
        [7, 46, 47, 47, 88, 6],
        [41, 55, 73, 23, 3, 79],
        [56, 69, 21, 86, 3, 33],
        [45, 7, 53, 81, 95, 79],
        [60, 77, 74, 68, 88, 51],
        [25, 25, 10, 12, 78, 14],
        [25, 56, 55, 58, 12, 82],
        [74, 33, 88, 23, 86, 59]
      ],
      titleSettings: {
        text: 'Sales Revenue per Employee (in 1000 US$)',
        textStyle: {
          size: '15px',
          fontWeight: '500',
          fontStyle: 'Normal',
          fontFamily: 'Segoe UI'
        }
      },
      xAxis: {
        labels: ['Nancy', 'Andrew', 'Janet', 'Margaret', 'Steven',
'Michael', 'Robert', 'Laura', 'Anne', 'Paul', 'Karin',
'Mario'],
      },
      yAxis: {
        labels: ['Mon', 'Tues', 'Wed', 'Thurs', 'Fri', 'Sat'],
      },
      paletteSettings: {
        palette: [
          { color: '#C06C84' },
          { color: '#6C5B7B' },
          { color: '#355C7D' }
        ]
      }
    }
  }
}
```



```

        type: "Gradient"
      },
      legendSettings: {
        visible: true,
      },
      cellSettings: {
        border: {
          width: 1
        },
        tileType: 'Bubble',
        bubbleType: 'Size',
        showLabel: false
      },
    },
  },
  provide: {
    heatmap: [Legend, Adaptor, Tooltip]
  }
}
</script>
<style>
@import
'https://ej2.syncfusion.com/vue/documentation/node_modules/@syncfusion/ej2-
vue-heatmap/styles/material.css';
</style>

```

{% previewsample "page.domainurl/code-snippet/heatmap-chart/bubble-heatmap-cs1" %}

#### Bubble color

In HeatMap, defined with this tile type, the data points will be represented with same sized bubbles and the data value variations are represented with the bubble colors.

To represent the data points with variations in bubble colors, set the [bubbleType](#) property to **Color**.

#### APP.VUE

```

<template>
  <div id="app">
    <ejs-heatmap id="heatmap" :titleSettings='titleSettings'
    :xAxis='xAxis' :yAxis='yAxis' :dataSource='dataSource'
    :paletteSettings='paletteSettings' :cellSettings='cellSettings'
    :legendSettings='legendSettings'></ejs-heatmap>
  </div>
</template>
<script>
import Vue from 'vue';
import { HeatMapPlugin, Legend, Adaptor, Tooltip } from '@syncfusion/ej2-
vue-heatmap';
Vue.use(HeatMapPlugin);
export default {
  data: function() {
    return {
      dataSource: [
        [73, 39, 26, 39, 94, 0],
        [93, 58, 53, 38, 26, 68],
        [99, 28, 22, 4, 66, 90],
        [14, 26, 97, 69, 69, 3],

```

```

        [7, 46, 47, 47, 88, 6],
        [41, 55, 73, 23, 3, 79],
        [56, 69, 21, 86, 3, 33],
        [45, 7, 53, 81, 95, 79],
        [60, 77, 74, 68, 88, 51],
        [25, 25, 10, 12, 78, 14],
        [25, 56, 55, 58, 12, 82],
        [74, 33, 88, 23, 86, 59]
    ],
    titleSettings: {
        text: 'Sales Revenue per Employee (in 1000 US$)',
        textStyle: {
            size: '15px',
            fontWeight: '500',
            fontStyle: 'Normal',
            fontFamily: 'Segoe UI'
        }
    },
    xAxis: {
        labels: ['Nancy', 'Andrew', 'Janet', 'Margaret', 'Steven',
            'Michael', 'Robert', 'Laura', 'Anne', 'Paul', 'Karin',
            'Mario'],
        labelRotation: 45,
        labelIntersectAction: 'None'
    },
    yAxis: {
        labels: ['Mon', 'Tues', 'Wed', 'Thurs', 'Fri', 'Sat'],
    },
    paletteSettings: {
        palette: [
            { color: '#C06C84'},
            { color: '#6C5B7B'},
            { color: '#355C7D'}
        ],
        type: "Gradient"
    },
    legendSettings: {
        visible: true,
    },
    cellSettings: {
        border: {
            width: 1
        },
        tileType: 'Bubble',
        bubbleType: 'Color'
    },
    },
    },
    provide:{
        heatmap:[Legend, Adaptor, Tooltip]
    }
}
</script>
<style>
@import
'https://ej2.syncfusion.com/vue/documentation/node_modules/@syncfusion/ej2-
vue-heatmap/styles/material.css';

```

```
</style>
```

```
{% previewsample "page.domainurl/code-snippet/heatmap-chart/bubble-heatmap-cs2" %}
```

### Bubble sector

In this bubble HeatMap type, the sector of the bubble decides the magnitude of data point. If the sector is large, then the data point value will be high.

To render the data points with bubble sector, set the [bubbleType](#) property to **Sector**.

### APP.VUE

```
<template>
  <div id="app">
    <ejs-heatmap id="heatmap" :titleSettings='titleSettings'
    :xAxis='xAxis' :yAxis='yAxis' :dataSource='dataSource'
    :paletteSettings='paletteSettings' :cellSettings='cellSettings'
    :legendSettings='legendSettings'></ejs-heatmap>
  </div>
</template>
<script>
import Vue from 'vue';
import { HeatMapPlugin, Legend, Adaptor, Tooltip } from '@syncfusion/ej2-
vue-heatmap';
Vue.use(HeatMapPlugin);
export default {
  data: function() {
    return {
      dataSource: [
        [73, 39, 26, 39, 94, 0],
        [93, 58, 53, 38, 26, 68],
        [99, 28, 22, 4, 66, 90],
        [14, 26, 97, 69, 69, 3],
        [7, 46, 47, 47, 88, 6],
        [41, 55, 73, 23, 3, 79],
        [56, 69, 21, 86, 3, 33],
        [45, 7, 53, 81, 95, 79],
        [60, 77, 74, 68, 88, 51],
        [25, 25, 10, 12, 78, 14],
        [25, 56, 55, 58, 12, 82],
        [74, 33, 88, 23, 86, 59]
      ],
      titleSettings: {
        text: 'Sales Revenue per Employee (in 1000 US$)',
        textStyle: {
          size: '15px',
          fontWeight: '500',
          fontStyle: 'Normal',
          fontFamily: 'Segoe UI'
        }
      },
      xAxis: {
        labels: ['Nancy', 'Andrew', 'Janet', 'Margaret', 'Steven',
        'Michael', 'Robert', 'Laura', 'Anne', 'Paul', 'Karin',
        'Mario'],
      },
      yAxis: {
```

```

      labels: ['Mon', 'Tues', 'Wed', 'Thurs', 'Fri', 'Sat'],
    },
    paletteSettings: {
      palette: [
        { color: '#C06C84' },
        { color: '#6C5B7B' },
        { color: '#355C7D' }
      ],
      type: "Gradient"
    },
    legendSettings: {
      visible: true,
    },
    cellSettings: {
      border: {
        width: 1
      },
      tileType: 'Bubble',
      bubbleType: 'Sector'
    },
  },
},
provide: {
  heatmap: [Legend, Adaptor, Tooltip]
}
}
</script>
<style>
@import
'https://ej2.syncfusion.com/vue/documentation/node_modules/@syncfusion/ej2-
vue-heatmap/styles/material.css';
</style>

```

{% previewsample "page.domainurl/code-snippet/heatmap-chart/bubble-heatmap-cs3" %}

### *Combining size and color bubble types*

In this bubble HeatMap type, size and color of the bubble represents the data value variation. To render this bubble HeatMap type, set the [bubbleType](#) property to **SizeAndColor**.

The following examples demonstrate different data binding with the **SizeAndColor** bubble type set in the HeatMap.

<!-- markdownlint-disable MD036 -->

### **Array binding**

When an array of numbers is specified as the data source, the bubble HeatMap can be rendered with different sizes and colors depending on the bound data.

<!-- markdownlint-disable MD036 -->

### **Table**

The following example illustrates how to render a bubble HeatMap with different sizes and colors using array table binding.

### **APP.VUE**

```

<template>
  <div id="app">
    <ejs-heatmap id="heatmap" :titleSettings='titleSettings'
    :xAxis='xAxis' :yAxis='yAxis' :dataSource='dataSource'
    :paletteSettings='paletteSettings' :cellSettings='cellSettings'
    :legendSettings='legendSettings' :tooltipRender='tooltipRender'></ejs-
    heatmap>
  </div>
</template>
<script>
import Vue from 'vue';
import { HeatMapPlugin, Legend, Adaptor, Tooltip } from '@syncfusion/ej2-
vue-heatmap';
Vue.use(HeatMapPlugin);
export default {
  data: function() {
    return {
      dataSource: [
        [[4,39], [3,8], [1,3], [1,10], [4,4], [2,15]],
        [[4,28], [5,92], [5,73], [3,1], [3,4], [4,126]],
        [[4,45], [5,152], [0,44], [4,54], [5,243], [2,45]]
      ],
      titleSettings: {
        text: 'Commercial Aviation Accidents and Fatalities by year 2012
- 2017',
        textStyle: {
          size: '15px',
          fontWeight: '500',
          fontStyle: 'Normal',
          fontFamily: 'Segoe UI'
        }
      },
      xAxis: {
        labels: ['2017', '2016', '2015']
      },
      yAxis: {
        labels: ['Jan-Feb', 'Mar-Apr', 'May-Jun', 'Jul-Aug', 'Sep-Oct',
'Nov-Dec']
      },
      paletteSettings: {
        palette: [
          { color: '#C06C84'},
          { color: '#6C5B7B'},
          { color: '#355C7D'}
        ],
        type: "Gradient"
      },
      legendSettings: {
        visible: true,
      },
      cellSettings: {
        border: {
          width: 1
        },
        tileType: 'Bubble',
        bubbleType: 'SizeAndColor'
      },
    },
  },

```

```

    }
  },
  provide: {
    heatmap: [Legend, Adaptor, Tooltip]
  },
  methods: {
    tooltipRender: function (args) {
      {
        args.content = ['Year ' + ' : ' + args.xLabel + '<br/>' +
          'Months ' + ' : ' + args.yLabel + '<br/>'
          + 'Accidents ' + ' : ' + args.value[0].bubbleData + '<br/>'
          + 'Fatalities ' + ' : '
          + args.value[1].bubbleData];
      }
    }
  }
}
</script>
<style>
@import '../node_modules/@syncfusion/ej2-vue-heatmap/styles/material.css';
</style>

```

{% previewsample "page.domainurl/code-snippet/heatmap-chart/bubble-heatmap-cs4" %}

<!-- markdownlint-disable MD036 -->

## Cell

The following example illustrates how to render a bubble HeatMap with different sizes and colors using array cell binding.

## APP.VUE

```

<template>
  <div id="app">
    <ejs-heatmap id="heatmap" :titleSettings='titleSettings'
      :xAxis='xAxis' :yAxis='yAxis' :dataSource='dataSource'
      :dataSourceSettings='dataSourceSettings' :paletteSettings='paletteSettings'
      :cellSettings='cellSettings' :legendSettings='legendSettings'
      :tooltipRender='tooltipRender'></ejs-heatmap>
  </div>
</template>
<script>
import Vue from 'vue';
import { HeatMapPlugin, Legend, Adaptor, Tooltip } from '@syncfusion/ej2-
vue-heatmap';
Vue.use(HeatMapPlugin);
export default {
  data: function () {
    return {
      dataSource: [
        [0,0,[4,39]], [0,1,[3,8]], [0,2,[1,3]], [0,3,[1,10]], [0,4,[4,4]],
        [0,5,[2,15]],
        [1,0,[4,28]], [1,1,[5,92]], [1,2,[5,73]], [1,3,[3,1]], [1,4,[3,4]],
        [1,5,[4,126]],
        [2,0,[4,45]], [2,1,[5,152]], [2,2,[0,44]], [2,3,[4,54]], [2,4,[5,243]],
        [2,5,[2,45]]
      ],
    }
  }
}

```

```

        dataSourceSettings: {
            isJsonData: false,
            adaptorType: 'Cell'},
        titleSettings: {
            text: 'Commercial Aviation Accidents and Fatalities by year 2012
- 2017',
            textStyle: {
                size: '15px',
                fontWeight: '500',
                fontStyle: 'Normal',
                fontFamily: 'Segoe UI'
            }
        },
        xAxis: {
            labels: ['2017', '2016', '2015']
        },
        yAxis: {
            labels: ['Jan-Feb', 'Mar-Apr', 'May-Jun', 'Jul-Aug', 'Sep-Oct',
'Nov-Dec']
        },
        paletteSettings: {
            palette: [
                { color: '#C06C84'},
                { color: '#6C5B7B'},
                { color: '#355C7D'}
            ],
            type: "Gradient"
        },
        legendSettings: {
            visible: true,
        },
        cellSettings: {
            border: {
                width: 1
            },
            tileType: 'Bubble',
            bubbleType: 'SizeAndColor'
        },
    },
    },
    provide:{
        heatmap:[Legend, Adaptor, Tooltip]
    },
    methods: {
        tooltipRender: function(args)
        {
            args.content = ['Year ' + ' : ' + args.xLabel + '<br/>' +
'Months ' + ' : ' + args.yLabel + '<br/>'
                + 'Accidents ' + ' : ' + args.value[0].bubbleData + '<br/>'
+ 'Fatalities ' + ' : '
                + args.value[1].bubbleData];
        }
    }
}
</script>
<style>
@import '../node_modules/@syncfusion/ej2-vue-heatmap/styles/material.css';

```

```
</style>
```

```
{% previewsample "page.domainurl/code-snippet/heatmap-chart/bubble-heatmap-cs5" %}
```

```
<!-- markdownlint-disable MD036 -->
```

### JSON binding

When a list of JSON objects are specified as data source, the bubble HeatMap can be rendered with different sizes and colors depending on the bound data.

```
<!-- markdownlint-disable MD036 -->
```

### Table

The following example illustrates how to render a bubble HeatMap with different sizes and colors using JSON table binding.

### APP.VUE

```
<template>
  <div id="app">
    <ejs-heatmap id="heatmap" :titleSettings='titleSettings'
      :xAxis='xAxis' :yAxis='yAxis' :dataSource='dataSource' :dataSourceSettings=
      'dataSourceSettings' :paletteSettings='paletteSettings'
      :cellSettings='cellSettings' :legendSettings='legendSettings'
      :tooltipRender='tooltipRender'></ejs-heatmap>
    </div>
  </template>
<script>
import Vue from 'vue';
import { HeatMapPlugin, Legend, Adaptor, Tooltip } from '@syncfusion/ej2-
vue-heatmap';
Vue.use(HeatMapPlugin);
export default {
  data: function() {
    return {
      dataSource: [
        { 'Year': '2017', 'Jan-Feb': [4,39], 'Mar-Apr': [3,8], 'May-Jun':
[1,3], 'Jul-Aug': [1,10], 'Sep-Oct': [4,4], 'Nov-Dec': [2,15]},
        { 'Year': '2016', 'Jan-Feb': [4,28], 'Mar-Apr': [5,92], 'May-Jun':
[5,73], 'Jul-Aug': [3,1], 'Sep-Oct': [3,4], 'Nov-Dec': [4,126]},
        { 'Year': '2015', 'Jan-Feb': [4,45], 'Mar-Apr': [5,152], 'May-Jun':
[0,44], 'Jul-Aug': [4,54], 'Sep-Oct': [5,243], 'Nov-Dec': [2,45]}
      ],
      dataSourceSettings: {
        isJsonData: true,
        adaptorType: 'Table',
        xDataMapping: 'Year'
      },
      titleSettings: {
        text: 'Commercial Aviation Accidents and Fatalities by year 2012
- 2017',
        textStyle: {
          size: '15px',
          fontWeight: '500',
          fontStyle: 'Normal',
          fontFamily: 'Segoe UI'
        }
      }
    }
  }
}
```



```

    },
    xAxis: {
      labels: ['2017', '2016', '2015']
    },
    yAxis: {
      labels: ['Jan-Feb', 'Mar-Apr', 'May-Jun', 'Jul-Aug', 'Sep-Oct',
'Nov-Dec']
    },
    paletteSettings: {
      palette: [
        { color: '#C06C84'},
        { color: '#6C5B7B'},
        { color: '#355C7D'}
      ],
      type: "Gradient"
    },
    legendSettings: {
      visible: true,
    },
    cellSettings: {
      border: {
        width: 1
      },
      tileType: 'Bubble',
      bubbleType: 'SizeAndColor'
    },
  },
},
provide:{
  heatmap:[Legend, Adaptor, Tooltip]
},
methods: {
  tooltipRender: function(args)
  {
    args.content = ['Year ' + ' : ' + args.xLabel + '<br/>' +
'Months ' + ' : ' + args.yLabel + '<br/>'
      + 'Accidents ' + ' : ' + args.value[0].bubbleData + '<br/>'
+ 'Fatalities ' + ' : '
      + args.value[1].bubbleData];
  }
}
}
</script>
<style>
@import '../node_modules/@syncfusion/ej2-vue-heatmap/styles/material.css';
</style>

```

{% previewsample "page.domainurl/code-snippet/heatmap-chart/bubble-heatmap-cs6" %}

<!-- markdownlint-disable MD036 -->

### Cell

The following example illustrates how to render a bubble HeatMap with different sizes and colors using JSON cell binding.

### APP.VUE

```

<template>
  <div id="app">
    <ejs-heatmap
      id="heatmap"
      :titleSettings="titleSettings"
      :xAxis="xAxis"
      :yAxis="yAxis"
      :dataSource="dataSource"
      :dataSourceSettings="dataSourceSettings"
      :paletteSettings="paletteSettings"
      :cellSettings="cellSettings"
      :legendSettings="legendSettings"
      :tooltipRender="tooltipRender"
    ></ejs-heatmap>
  </div>
</template>
<script>
import Vue from "vue";
import {
  HeatMapPlugin,
  Legend,
  Adaptor,
  Tooltip
} from "@syncfusion/ej2-vue-heatmap";
Vue.use(HeatMapPlugin);
export default {
  data: function() {
    return {
      dataSource: [
        { Year: "2017", Months: "Jan-Feb", Accidents: 4, Fatalities: 39 },
        { Year: "2017", Months: "Mar-Apr", Accidents: 3, Fatalities: 8 },
        { Year: "2017", Months: "May-Jun", Accidents: 1, Fatalities: 3 },
        { Year: "2017", Months: "Jul-Aug", Accidents: 1, Fatalities: 10 },
        { Year: "2017", Months: "Sep-Oct", Accidents: 4, Fatalities: 4 },
        { Year: "2017", Months: "Nov-Dec", Accidents: 2, Fatalities: 15 },
        { Year: "2016", Months: "Jan-Feb", Accidents: 4, Fatalities: 28 },
        { Year: "2016", Months: "Mar-Apr", Accidents: 5, Fatalities: 92 },
        { Year: "2016", Months: "May-Jun", Accidents: 5, Fatalities: 73 },
        { Year: "2016", Months: "Jul-Aug", Accidents: 3, Fatalities: 1 },
        { Year: "2016", Months: "Sep-Oct", Accidents: 3, Fatalities: 4 },
        { Year: "2016", Months: "Nov-Dec", Accidents: 4, Fatalities: 126 },
      ],
      { Year: "2015", Months: "Jan-Feb", Accidents: 4, Fatalities: 45 },
      { Year: "2015", Months: "Mar-Apr", Accidents: 5, Fatalities: 152 },
    },
    { Year: "2015", Months: "May-Jun", Accidents: 0, Fatalities: 0 },
    { Year: "2015", Months: "Jul-Aug", Accidents: 4, Fatalities: 54 },
    { Year: "2015", Months: "Sep-Oct", Accidents: 5, Fatalities: 243 },
  },
    { Year: "2015", Months: "Nov-Dec", Accidents: 2, Fatalities: 45 }
  ],
  dataSourceSettings: {
    isJsonData: true,
    adaptorType: "Cell",
    xDataMapping: "Year",
    yDataMapping: "Months",
    bubbleDataMapping: { size: "Accidents", color: "Fatalities" }
  }
}

```

```

    },
    titleSettings: {
      text:
        "Commercial Aviation Accidents and Fatalities by year 2012 -
2017",
      textStyle: {
        size: "15px",
        fontWeight: "500",
        fontStyle: "Normal",
        fontFamily: "Segoe UI"
      }
    },
    xAxis: {
      labels: ["2017", "2016", "2015"]
    },
    yAxis: {
      labels: [
        "Jan-Feb",
        "Mar-Apr",
        "May-Jun",
        "Jul-Aug",
        "Sep-Oct",
        "Nov-Dec"
      ]
    },
    paletteSettings: {
      palette: [
        { color: "#C06C84" },
        { color: "#6C5B7B" },
        { color: "#355C7D" }
      ],
      type: "Gradient"
    },
    legendSettings: {
      visible: true
    },
    cellSettings: {
      border: {
        width: 1
      },
      tileType: "Bubble",
      bubbleType: "SizeAndColor"
    }
  };
},
provide: {
  heatmap: [Legend, Adaptor, Tooltip]
},
methods: {
  tooltipRender: function(args) {
    args.content = [
      "Year " +
      " : " +
      args.xLabel +
      "<br/>" +
      "Months " +
      " : " +

```

```

        args.yLabel +
        "<br/>" +
        "Accidents " +
        " : " +
        args.value[0].bubbleData +
        "<br/>" +
        "Fatalities " +
        " : " +
        args.value[1].bubbleData
    ];
    }
}
};
</script>

```

```
{% previewsample "page.domainurl/code-snippet/heatmap-chart/bubble-heatmap-cs7" %}
```

```
<!-- markdownlint-disable MD036 -->
```

### Binding size and color values from datasource

The size and color of the bubbles in the **SizeAndColor** bubble HeatMap type can be customized by binding the datasource field name that holds the size and color values to the [size](#) and [color](#) properties in the [bubbleDataMapping](#).

The `bubbleDataMapping` supports only for the JSON data with cell adaptor type.

### APP.VUE

```

<template>
  <div id="app">
    <ejs-heatmap
      id="heatmap"
      :titleSettings="titleSettings"
      :xAxis="xAxis"
      :yAxis="yAxis"
      :dataSource="dataSource"
      :dataSourceSettings="dataSourceSettings"
      :paletteSettings="paletteSettings"
      :cellSettings="cellSettings"
      :legendSettings="legendSettings"
      :tooltipRender="tooltipRender"
    ></ejs-heatmap>
  </div>
</template>
<script>
import Vue from "vue";
import {
  HeatMapPlugin,
  Legend,
  Adaptor,
  Tooltip
} from "@syncfusion/ej2-vue-heatmap";
Vue.use(HeatMapPlugin);
export default {
  data: function() {
    return {

```

```

dataSource: [
  { Year: "2017", Months: "Jan-Feb", Accidents: 4, Fatalities: 39 },
  { Year: "2017", Months: "Mar-Apr", Accidents: 3, Fatalities: 8 },
  { Year: "2017", Months: "May-Jun", Accidents: 1, Fatalities: 3 },
  { Year: "2017", Months: "Jul-Aug", Accidents: 1, Fatalities: 10 },
  { Year: "2017", Months: "Sep-Oct", Accidents: 4, Fatalities: 4 },
  { Year: "2017", Months: "Nov-Dec", Accidents: 2, Fatalities: 15 },
  { Year: "2016", Months: "Jan-Feb", Accidents: 4, Fatalities: 28 },
  { Year: "2016", Months: "Mar-Apr", Accidents: 5, Fatalities: 92 },
  { Year: "2016", Months: "May-Jun", Accidents: 5, Fatalities: 73 },
  { Year: "2016", Months: "Jul-Aug", Accidents: 3, Fatalities: 1 },
  { Year: "2016", Months: "Sep-Oct", Accidents: 3, Fatalities: 4 },
  { Year: "2016", Months: "Nov-Dec", Accidents: 4, Fatalities: 126 },
},
  { Year: "2015", Months: "Jan-Feb", Accidents: 4, Fatalities: 45 },
  { Year: "2015", Months: "Mar-Apr", Accidents: 5, Fatalities: 152 },
},
  { Year: "2015", Months: "May-Jun", Accidents: 0, Fatalities: 0 },
  { Year: "2015", Months: "Jul-Aug", Accidents: 4, Fatalities: 54 },
  { Year: "2015", Months: "Sep-Oct", Accidents: 5, Fatalities: 243 },
},
  { Year: "2015", Months: "Nov-Dec", Accidents: 2, Fatalities: 45 }
],
dataSourceSettings: {
  isJsonData: true,
  adaptorType: "Cell",
  xDataMapping: "Year",
  yDataMapping: "Months",
  bubbleDataMapping: { size: "Accidents", color: "Fatalities" }
},
titleSettings: {
  text:
    "Commercial Aviation Accidents and Fatalities by year 2012 -
2017",
  textStyle: {
    size: "15px",
    fontWeight: "500",
    fontStyle: "Normal",
    fontFamily: "Segoe UI"
  }
},
xAxis: {
  labels: ["2017", "2016", "2015"]
},
yAxis: {
  labels: [
    "Jan-Feb",
    "Mar-Apr",
    "May-Jun",
    "Jul-Aug",
    "Sep-Oct",
    "Nov-Dec"
  ]
},
paletteSettings: {
  palette: [
    { color: "#C06C84" },

```

```

        { color: "#6C5B7B" },
        { color: "#355C7D" }
    ],
    type: "Gradient"
  },
  legendSettings: {
    visible: true
  },
  cellSettings: {
    border: {
      width: 1
    },
    tileType: "Bubble",
    bubbleType: "SizeAndColor"
  }
};
},
provide: {
  heatmap: [Legend, Adaptor, Tooltip]
},
methods: {
  tooltipRender: function(args) {
    args.content = [
      "Year " +
      " : " +
      args.xLabel +
      "<br/>" +
      "Months " +
      " : " +
      args.yLabel +
      "<br/>" +
      "Accidents " +
      " : " +
      args.value[0].bubbleData +
      "<br/>" +
      "Fatalities " +
      " : " +
      args.value[1].bubbleData
    ];
  }
}
};
</script>

```

{% previewsample "page.domainurl/code-snippet/heatmap-chart/bubble-heatmap-cs8" %}

### Rendering mode in Vue Heatmap chart component

Heat map can be displayed using **Canvas** or **Scalable Vector Graphics (SVG)** rendering logic to improve the initial load performance and scalability. Heat map can also be automatically switched between **Canvas** and **SVG** modes based on dataset size. You can enable this mode by setting the [renderingMode](#) property to **Auto**.

If the **Auto** mode is enabled in the heat map and there are more than 10,000 data points, then the heat map will be rendered in a **Canvas** mode; Otherwise, the heat map will be rendered in a **SVG** mode.

**APP.VUE**

```

<template>
  <div id="app">
    <ejs-heatmap id="heatmap" :titleSettings='titleSettings'
:cellSettings='cellSettings' :xAxis='xAxis' :yAxis='yAxis'
renderingMode='SVG' :dataSource='dataSource'
:showTooltip='showTooltip'></ejs-heatmap>
  </div>
</template>
<script>
import Vue from 'vue';
import { HeatMapPlugin, Tooltip, Legend } from '@syncfusion/ej2-vue-heatmap';
Vue.use(HeatMapPlugin);
export default {
  data: function() {
    return {
      dataSource: [
        [73, 39, 26, 39, 94, 0],
        [93, 58, 53, 38, 26, 68],
        [99, 28, 22, 4, 66, 90],
        [14, 26, 97, 69, 69, 3],
        [7, 46, 47, 47, 88, 6],
        [41, 55, 73, 23, 3, 79],
        [56, 69, 21, 86, 3, 33],
        [45, 7, 53, 81, 95, 79],
        [60, 77, 74, 68, 88, 51],
        [25, 25, 10, 12, 78, 14],
        [25, 56, 55, 58, 12, 82],
        [74, 33, 88, 23, 86, 59]
      ],
      titleSettings: {
        text: 'Sales Revenue per Employee (in 1000 US$)',
        textStyle: {
          size: '15px',
          fontWeight: '500',
          fontStyle: 'Normal',
          fontFamily: 'Segoe UI'
        }
      },
      cellSettings: {
        showLabel: true,
      },
      xAxis: {
        labels: ['Nancy', 'Andrew', 'Janet', 'Margaret', 'Steven',
'Michael', 'Robert', 'Laura', 'Anne', 'Paul', 'Karin',
'Mario'],
      },
      yAxis: {
        labels: ['Mon', 'Tues', 'Wed', 'Thurs', 'Fri', 'Sat'],
      },
      showTooltip: true
    }
  },
  provide: {
    heatmap: [Tooltip, Legend]
  }
}

```

```

}
}
</script>
<style>
@import '../node_modules/@syncfusion/ej2-vue-heatmap/styles/material.css';
</style>

```

{% previewsample "page.domainurl/code-snippet/heatmap-chart/rendering-mode-cs1" %}

### Axis in Vue HeatMap chart component

HeatMap consists of two axes namely, X-axis and Y-axis that displays the row headers and column headers to plot the data points respectively. You can define the type, format, and other customizing options for both axes in the HeatMap.

#### Types

There are three different axis types available in the HeatMap, which defines the data type of the axis labels. You can define the axis type by using the [valueType](#) property in the HeatMap.

#### Category axis

Category axis type is used to represent the string values in axis labels.

#### APP.VUE

```

<template>
  <div id="app">
    <ejs-heatmap id="heatmap" :xAxis='xAxis' :yAxis='yAxis'
:dataSource='dataSource'></ejs-heatmap>
  </div>
</template>
<script>
import Vue from 'vue';
import { HeatMapPlugin, Tooltip, Legend } from '@syncfusion/ej2-vue-heatmap';
Vue.use(HeatMapPlugin);
export default {
  data: function() {
    return {
      dataSource: [
        [73, 39, 26, 39, 94, 0],
        [93, 58, 53, 38, 26, 68],
        [99, 28, 22, 4, 66, 90],
        [14, 26, 97, 69, 69, 3],
        [7, 46, 47, 47, 88, 6],
        [41, 55, 73, 23, 3, 79],
        [56, 69, 21, 86, 3, 33],
        [45, 7, 53, 81, 95, 79],
        [60, 77, 74, 68, 88, 51],
        [25, 25, 10, 12, 78, 14],
        [25, 56, 55, 58, 12, 82],
        [74, 33, 88, 23, 86, 59]
      ],
      xAxis: {
        valueType: "Category",
        labels: ['Nancy', 'Andrew', 'Janet', 'Margaret', 'Steven',
'Michael', 'Robert', 'Laura', 'Anne', 'Paul', 'Karin',
'Mario'],

```



```

    },
    yAxis: {
      valueType: "Category",
      labels: ['Mon', 'Tues', 'Wed', 'Thurs', 'Fri', 'Sat'],
    }
  }
},
provide: {
  heatmap: [Tooltip, Legend]
}
}
</script>
<style>
@import '../node_modules/@syncfusion/ej2-vue-heatmap/styles/material.css';
</style>

```

{% previewsample "page.domainurl/code-snippet/heatmap-chart/axis-cs1" %}

### Numeric axis

Numeric axis type is used to represent the numeric values in axis labels.

### APP.VUE

```

<template>
  <div id="app">
    <ejs-heatmap id="heatmap" :xAxis='xAxis' :yAxis='yAxis'
    :dataSource='dataSource'></ejs-heatmap>
  </div>
</template>
<script>
import Vue from 'vue';
import { HeatMapPlugin, Tooltip, Legend } from '@syncfusion/ej2-vue-heatmap';
Vue.use(HeatMapPlugin);
export default {
  data: function() {
    return {
      dataSource: [
        [73, 39, 26, 39, 94, 0],
        [93, 58, 53, 38, 26, 68],
        [99, 28, 22, 4, 66, 90],
        [14, 26, 97, 69, 69, 3],
        [7, 46, 47, 47, 88, 6],
        [41, 55, 73, 23, 3, 79],
        [56, 69, 21, 86, 3, 33],
        [45, 7, 53, 81, 95, 79],
        [60, 77, 74, 68, 88, 51],
        [25, 25, 10, 12, 78, 14],
        [25, 56, 55, 58, 12, 82],
        [74, 33, 88, 23, 86, 59]
      ],
      xAxis: {
        valueType: "Numeric",
        minimum: 0,
        maximum: 11
      },
    },
  },

```

```

        yAxis: {
            valueType: "Numeric",
            minimum: 0,
            maximum: 5
        }
    },
    provide: {
        heatmap: [Tooltip, Legend]
    }
}
</script>
<style>
@import
'https://ej2.syncfusion.com/vue/documentation/node_modules/@syncfusion/ej2-vue-heatmap/styles/material.css';
</style>

```

{% previewsample "page.domainurl/code-snippet/heatmap-chart/axis-cs2" %}

#### *Date-time axis*

Date-time axis type is used to represent the date-time values in axis labels with a specific format. In date-time axis, you can define the start and end date/time using the [minimum](#) and [maximum](#) properties.

#### **APP.VUE**

```

<template>
    <div id="app">
        <ejs-heatmap id="heatmap" :titleSettings='titleSettings'
        :xAxis='xAxis' :yAxis='yAxis' :legendSettings='legendSettings'
        :cellSettings='cellSettings' :dataSource='dataSource'></ejs-heatmap>
    </div>
</template>
<script>
import Vue from 'vue';
import { HeatMapPlugin, Tooltip, Legend } from '@syncfusion/ej2-vue-heatmap';
Vue.use(HeatMapPlugin);
export default {
    data: function() {
        return {
            dataSource: [
                [36371, 25675, 28292, 33399, 35980, 38585, 39351, 39964, 36543,
                30529, 33298, 36985],
                [34702, 27618, 31063, 34525, 36772, 35410, 38750, 39467, 35390,
                34196, 35302, 35703],
                [34522, 31324, 32128, 34231, 36817, 34381, 37180, 38255, 32776,
                32645, 31539, 32981],
                [32213, 28755, 29517, 31214, 33747, 33507, 35763, 36837, 32910,
                33437, 30659, 31965],
                [31282, 28663, 32952, 33941, 34506, 36875, 38836, 35497, 34285,
                34094, 32256, 33699],
                [31714, 29405, 33745, 32838, 33461, 35034, 36122, 37943, 34128,
                30624, 32398, 33522],
            ]
        }
    }
}

```

```

        [32064, 28387, 33751, 32537, 34034, 35977, 37196, 38301, 33627,
        34115, 31072, 33939],
        [32417, 27868, 30807, 33386, 35284, 36126, 39753, 40978, 35777,
        35277, 31281, 35411],
        [32494, 29848, 34385, 35804, 37943, 38722, 41315, 41335, 37177,
        37443, 32457, 37304],
        [34378, 29576, 30547, 35664, 36622, 38145, 40347, 41868, 38252,
        36505, 29576, 36450],
        [35219, 31670, 32589, 34927, 36998, 39825, 41126, 42002, 37021,
        36583, 32408, 37108]
    ],
    titleSettings: {
        text: 'Monthly Flight Traffic at JFK Airport',
        textStyle: {
            size: '15px',
            fontWeight: '500',
            fontStyle: 'Normal',
            fontFamily: 'Segoe UI'
        }
    },
    xAxis: {
        valueType: "DateTime",
        minimum: new Date(2007, 0, 1),
        maximum: new Date(2017, 0, 1),
        intervalType: "Years",
        labelFormat: "yyyy",
        labelRotation: 45,
        labelIntersectAction: 'None'
    },
    yAxis: {
        labels: ['Jan', 'Feb', 'Mar', 'Apr', 'May',
            'Jun', 'July', 'Aug', 'Sept', 'Oct', 'Nov', 'Dec']
    },
    legendSettings: {
        visible: false,
    },
    cellSettings: {
        showLabel: false,
        border: {
            width: 0,
        },
        format: '{value} flights'
    }
}
},
provide: {
    heatmap: [Tooltip, Legend]
}
}
</script>
<style>
@import
'https://ej2.syncfusion.com/vue/documentation/node_modules/@syncfusion/ej2-
vue-heatmap/styles/material.css';
</style>

```

{% previewsample "page.domainurl/code-snippet/heatmap-chart/axis-cs3" %}

### Inversed axis

HeatMap supports inverting the axis origin for both axes, where the axis labels are placed in an inversed manner. You can enable axis inverting by enabling the [isInversed](#) property.

### APP.VUE

```
<template>
  <div id="app">
    <ejs-heatmap id="heatmap" :titleSettings='titleSettings'
    :xAxis='xAxis' :yAxis='yAxis' :legendSettings='legendSettings'
    :cellSettings='cellSettings' :dataSource='dataSource'></ejs-heatmap>
  </div>
</template>
<script>
import Vue from 'vue';
import { HeatMapPlugin, Tooltip, Legend } from '@syncfusion/ej2-vue-heatmap';
Vue.use(HeatMapPlugin);
export default {
  data: function() {
    return {
      dataSource: [
        [36371, 25675, 28292, 33399, 35980, 38585, 39351, 39964, 36543,
        30529, 33298, 36985],
        [34702, 27618, 31063, 34525, 36772, 35410, 38750, 39467, 35390,
        34196, 35302, 35703],
        [34522, 31324, 32128, 34231, 36817, 34381, 37180, 38255, 32776,
        32645, 31539, 32981],
        [32213, 28755, 29517, 31214, 33747, 33507, 35763, 36837, 32910,
        33437, 30659, 31965],
        [31282, 28663, 32952, 33941, 34506, 36875, 38836, 35497, 34285,
        34094, 32256, 33699],
        [31714, 29405, 33745, 32838, 33461, 35034, 36122, 37943, 34128,
        30624, 32398, 33522],
        [32064, 28387, 33751, 32537, 34034, 35977, 37196, 38301, 33627,
        34115, 31072, 33939],
        [32417, 27868, 30807, 33386, 35284, 36126, 39753, 40978, 35777,
        35277, 31281, 35411],
        [32494, 29848, 34385, 35804, 37943, 38722, 41315, 41335, 37177,
        37443, 32457, 37304],
        [34378, 29576, 30547, 35664, 36622, 38145, 40347, 41868, 38252,
        36505, 29576, 36450],
        [35219, 31670, 32589, 34927, 36998, 39825, 41126, 42002, 37021,
        36583, 32408, 37108]
      ],
      titleSettings: {
        text: 'Monthly Flight Traffic at JFK Airport',
        textStyle: {
          size: '15px',
          fontWeight: '500',
          fontStyle: 'Normal',
          fontFamily: 'Segoe UI'
        }
      },
      xAxis: {
```

```

        valueType: "DateTime",
        minimum: new Date(2007, 0, 1),
        maximum: new Date(2017, 0, 1),
        intervalType: "Years",
        labelFormat: "yyyy",
        labelRotation: 45,
        labelIntersectAction: 'None',
        isInversed: true
    },
    yAxis: {
        labels: ['Jan', 'Feb', 'Mar', 'Apr', 'May',
            'Jun', 'July', 'Aug', 'Sept', 'Oct', 'Nov', 'Dec']
    },
    legendSettings: {
        visible: false,
    },
    cellSettings: {
        showLabel: false,
        border: {
            width: 0,
        },
        format: '{value} flights'
    }
}
},
provide: {
    heatmap: [Tooltip, Legend]
}
}
</script>
<style>
@import
'https://ej2.syncfusion.com/vue/documentation/node_modules/@syncfusion/ej2-
vue-heatmap/styles/material.css';
</style>

```

{% previewsample "page.domainurl/code-snippet/heatmap-chart/axis-cs4" %}

### Opposed axis

In HeatMap, you can place the axis label in an opposite position of its default axis label position by using the [opposedPosition](#) property.

### APP.VUE

```

<template>
    <div id="app">
        <ejs-heatmap id="heatmap" :titleSettings='titleSettings'
        :xAxis='xAxis' :yAxis='yAxis' :legendSettings='legendSettings'
        :cellSettings='cellSettings' :dataSource='dataSource'></ejs-heatmap>
    </div>
</template>
<script>
import Vue from 'vue';
import { HeatMapPlugin, Tooltip, Legend } from '@syncfusion/ej2-vue-
heatmap';
Vue.use(HeatMapPlugin);

```

```

export default {
  data: function() {
    return {
      dataSource: [
        [36371, 25675, 28292, 33399, 35980, 38585, 39351, 39964, 36543,
        30529, 33298, 36985],
        [34702, 27618, 31063, 34525, 36772, 35410, 38750, 39467, 35390,
        34196, 35302, 35703],
        [34522, 31324, 32128, 34231, 36817, 34381, 37180, 38255, 32776,
        32645, 31539, 32981],
        [32213, 28755, 29517, 31214, 33747, 33507, 35763, 36837, 32910,
        33437, 30659, 31965],
        [31282, 28663, 32952, 33941, 34506, 36875, 38836, 35497, 34285,
        34094, 32256, 33699],
        [31714, 29405, 33745, 32838, 33461, 35034, 36122, 37943, 34128,
        30624, 32398, 33522],
        [32064, 28387, 33751, 32537, 34034, 35977, 37196, 38301, 33627,
        34115, 31072, 33939],
        [32417, 27868, 30807, 33386, 35284, 36126, 39753, 40978, 35777,
        35277, 31281, 35411],
        [32494, 29848, 34385, 35804, 37943, 38722, 41315, 41335, 37177,
        37443, 32457, 37304],
        [34378, 29576, 30547, 35664, 36622, 38145, 40347, 41868, 38252,
        36505, 29576, 36450],
        [35219, 31670, 32589, 34927, 36998, 39825, 41126, 42002, 37021,
        36583, 32408, 37108]
      ],
      titleSettings: {
        text: 'Monthly Flight Traffic at JFK Airport',
        textStyle: {
          size: '15px',
          fontWeight: '500',
          fontStyle: 'Normal',
          fontFamily: 'Segoe UI'
        }
      },
      xAxis: {
        valueType: "DateTime",
        minimum: new Date(2007, 0, 1),
        maximum: new Date(2017, 0, 1),
        intervalType: "Years",
        labelFormat: "yyyy",
        labelRotation: 45,
        labelIntersectAction: 'None',
        opposedPosition: true
      },
      yAxis: {
        labels: ['Jan', 'Feb', 'Mar', 'Apr', 'May',
        'Jun', 'July', 'Aug', 'Sept', 'Oct', 'Nov', 'Dec']
      },
      legendSettings: {
        visible: false,
      },
      cellSettings: {
        showLabel: false,
        border: {
          width: 0,

```

```

        },
        format: '{value} flights'
    }
}
},
provide:{
    heatmap:[Tooltip, Legend]
}
}
</script>
<style>
@import
'https://ej2.syncfusion.com/vue/documentation/node_modules/@syncfusion/ej2-
vue-heatmap/styles/material.css';
</style>

```

{% previewsample "page.domainurl/code-snippet/heatmap-chart/axis-cs5" %}

### Axis labels customization

#### *Customizing the text style*

The text style of the axis labels can be customized using the following options available in the [textStyle](#) property.

- [color](#) - It is used to change the text color of the axis labels.
- [fontFamily](#) - It is used to change the font family of the axis labels.
- [fontStyle](#) - It is used to change the font style of the axis labels.
- [fontWeight](#) - It is used to change the font weight of the axis labels.
- [size](#) - It is used to change the font size of the axis labels.
- [textAlignment](#) - It is used to position and align the axis labels. This property allows you to specify values such as **Near**, **Center**, and **Far**.
- [textOverflow](#) - When the axis label exceeds the intended space, this property is used to trim or wrap it. This property takes values such as **None**, **Trim**, and **Wrap**.

### **APP.VUE**

```

<template>
  <div id="app">
    <ejs-heatmap id="heatmap" :xAxis='xAxis' :yAxis='yAxis'
    :dataSource='dataSource'
    :titleSettings='titleSettings' :paletteSettings='paletteSettings'
    :legendSettings='legendSettings'></ejs-heatmap>
  </div>
</template>
<script>
import Vue from 'vue';
import { HeatMapPlugin, Tooltip } from '@syncfusion/ej2-vue-heatmap';
Vue.use(HeatMapPlugin);
export default {
  data: function() {
    return {
      dataSource: [
        [52, 65, 67, 45, 37, 52, 32],
        [68, 52, 63, 51, 30, 51, 51],

```

```

        [60, 50, 42, 53, 66, 70, 41],
        [66, 64, 46, 40, 47, 41, 45],
        [65, 42, 58, 32, 36, 44, 49],
        [54, 46, 61, 46, 40, 39, 41],
        [48, 46, 61, 47, 49, 41, 41],
        [69, 52, 41, 44, 41, 52, 46],
        [50, 59, 44, 43, 27, 42, 26],
        [47, 49, 66, 53, 50, 34, 31],
        [61, 40, 62, 26, 34, 54, 56]
    ],
    titleSettings: {
        text: 'Product wise Monthly sales revenue for a e-commerce
website',
        textStyle: {
            size: '15px',
            fontWeight: '500',
            fontStyle: 'Normal',
            fontFamily: 'Segoe UI'
        }
    },
    xAxis: {
        textStyle: {
            color: 'red',
            size: '15px',
            fontWeight: '650',
            fontStyle: 'Normal',
            fontFamily: 'Segoe UI',
            textAlignment: 'Center',
            textOverflow: 'Wrap'
        },
        opposedPosition: true,
        labels: [
            'Month of Feburary 2023',
            'Month of March 2023',
            'Month of April 2023',
            'Month of May 2023',
            'Month of June 2023',
            'Month of July 2023',
            'Month of August 2023',
            'Month of September 2023',
            'Month of October 2023',
            'Month of November 2023',
            'Month of December 2023'
        ]
    },
    yAxis: {
        textStyle: {
            color: 'red',
            size: '15px',
            fontWeight: '650',
            fontStyle: 'Normal',
            fontFamily: 'Segoe UI',
            textAlignment: 'Center',
            textOverflow: 'Wrap',
        },
        maxLabelLength: 70,
        labels: [

```



```

        'Ace Apparels',
        'Alpha Apparels',
        'RL Garments',
        'RRD Garments',
        'RRD Apparels',
        'RR Garments',
        'SR Garments'
    ],
    },
    legendSettings: {
        visible: false
    },
    paletteSettings: {
        palette: [
            { color: '#F0C27B' },
            { color: '#4B1248' }
        ],
    }
},
provide: {
    heatmap: [Tooltip]
}
}
</script>

```

{% previewsample "page.domainurl/code-snippet/heatmap-chart/axis-cs11" %}

#### Providing line breaks

Axis labels with line breaks improve the readability of the HeatMap by splitting the text on an axis into multiple lines. The “`<br>`” character is used to add line breaks to the axis labels.

#### APP.VUE

```

<template>
  <div id="app">
    <ejs-heatmap id="heatmap" :xAxis='xAxis' :yAxis='yAxis'
    :dataSource='dataSource'></ejs-heatmap>
  </div>
</template>
<script>
import Vue from 'vue';
import { HeatMapPlugin, Legend, Tooltip } from '@syncfusion/ej2-vue-heatmap';
Vue.use(HeatMapPlugin);
export default {
  data: function () {
    return {
      dataSource: [
        [1, 76],
        [19, 3]
      ],
      xAxis: {
        labels: ['Actual<br>Accept', 'Actual<br>Reject'],
        opposedPosition: true
      },
    },
  },

```

```

        yAxis: {
            labels: ['Actual<br>Accept', 'Actual<br>Reject'],
            maxLabelLength: 50
        }
    },
    provide: {
        heatmap: [Tooltip, Legend]
    }
}
</script>

```

{% previewsample "page.domainurl/code-snippet/heatmap-chart/axis-cs14" %}

#### *Customizing labels when intersecting with other labels*

When the axis labels intersect, [labelIntersectAction](#) property is used to handle the intersection. The [labelIntersectAction](#) property can take the following values.

- **None** - It specifies that no action is taken when the axis labels intersect.
- **Trim** - It specifies to trim the axis labels when they intersect.
- **Rotate45** - When the axis labels intersect, they are rotated to 45 degrees.
- **MultipleRows** - It specifies to show all the axis labels as multiple rows when they intersect.

The below example demonstrates to trim the axis labels by using the [labelIntersectAction](#) property.

#### **APP.VUE**

```

<template>
  <div id="app">
    <ejs-heatmap id="heatmap" :xAxis='xAxis' :yAxis='yAxis'
    :dataSource='dataSource'
    :titleSettings='titleSettings' :paletteSettings='paletteSettings'
    :legendSettings='legendSettings'></ejs-heatmap>
  </div>
</template>
<script>
import Vue from 'vue';
import { HeatMapPlugin, Tooltip } from '@syncfusion/ej2-vue-heatmap';
Vue.use(HeatMapPlugin);
export default {
  data: function() {
    return {
      dataSource:[
        [52, 65, 67, 45, 37, 52, 32],
        [68, 52, 63, 51, 30, 51, 51],
        [60, 50, 42, 53, 66, 70, 41],
        [66, 64, 46, 40, 47, 41, 45],
        [65, 42, 58, 32, 36, 44, 49],
        [54, 46, 61, 46, 40, 39, 41],
        [48, 46, 61, 47, 49, 41, 41],
        [69, 52, 41, 44, 41, 52, 46],
        [50, 59, 44, 43, 27, 42, 26],
        [47, 49, 66, 53, 50, 34, 31],
        [61, 40, 62, 26, 34, 54, 56]
      ],

```

```

    titleSettings: {
      text: 'Product wise Monthly sales revenue for a e-commerce
website',
      textStyle: {
        size: '15px',
        fontWeight: '500',
        fontStyle: 'Normal',
        fontFamily: 'Segoe UI'
      }
    },
    xAxis: {
      enableTrim: true,
      labelIntersectAction: 'Trim',
      opposedPosition: true,
      labels: [
        'Month of Feburary 2023',
        'Month of March 2023',
        'Month of April 2023',
        'Month of May 2023',
        'Month of June 2023',
        'Month of July 2023',
        'Month of August 2023',
        'Month of September 2023',
        'Month of October 2023',
        'Month of November 2023',
        'Month of December 2023'
      ]
    },
    yAxis: {
      enableTrim: true,
      labelIntersectAction: 'Trim',
      labels: [
        'Ace Apparels',
        'Alpha Apparels',
        'RL Garments',
        'RRD Garments',
        'RRD Apparels',
        'RR Garments',
        'SR Garments'
      ]
    },
    legendSettings: {
      visible: false
    },
    paletteSettings: {
      palette: [
        { color: '#F0C27B' },
        { color: '#4B1248' }
      ]
    },
    provide: {
      heatmap: [Tooltip]
    }
  }
</script>

```

{% previewsample "page.domainurl/code-snippet/heatmap-chart/axis-cs12" %}

### Rotating labels

The axis labels can be rotated to the desired angles by using the [labelRotation](#) property.

### APP.VUE

```
<template>
  <div id="app">
    <ejs-heatmap id="heatmap" :xAxis='xAxis' :yAxis='yAxis'
      :dataSource='dataSource'
      :titleSettings='titleSettings' :paletteSettings='paletteSettings'
      :legendSettings='legendSettings'></ejs-heatmap>
    </div>
  </template>
<script>
import Vue from 'vue';
import { HeatMapPlugin, Tooltip } from '@syncfusion/ej2-vue-heatmap';
Vue.use(HeatMapPlugin);
export default {
  data: function() {
    return {
      dataSource:[
        [52, 65, 67, 45, 37, 52, 32],
        [68, 52, 63, 51, 30, 51, 51],
        [60, 50, 42, 53, 66, 70, 41],
        [66, 64, 46, 40, 47, 41, 45],
        [65, 42, 58, 32, 36, 44, 49],
        [54, 46, 61, 46, 40, 39, 41],
        [48, 46, 61, 47, 49, 41, 41],
        [69, 52, 41, 44, 41, 52, 46],
        [50, 59, 44, 43, 27, 42, 26],
        [47, 49, 66, 53, 50, 34, 31],
        [61, 40, 62, 26, 34, 54, 56]
      ],
      titleSettings: {
        text: 'Product wise Monthly sales revenue for a e-commerce
website',
        textStyle: {
          size: '15px',
          fontWeight: '500',
          fontStyle: 'Normal',
          fontFamily: 'Segoe UI'
        }
      },
      xAxis: {
        labelRotation:90,
        opposedPosition: true,
        labels: [
          'Month of Feburary 2023',
          'Month of March 2023',
          'Month of April 2023',
          'Month of May 2023',
          'Month of June 2023',
          'Month of July 2023',
          'Month of August 2023',
        ]
      }
    }
  }
}
```

```

        'Month of September 2023',
        'Month of October 2023',
        'Month of November 2023',
        'Month of December 2023'
    ],
    },
    yAxis: {
        labelRotation: 45,
        labels: [
            'Ace Apparels',
            'Alpha Apparels',
            'RL Garments',
            'RRD Garments',
            'RRD Apparels',
            'RR Garments',
            'SR Garments'
        ]
    },
    legendSettings: {
        visible: false
    },
    paletteSettings: {
        palette: [
            { color: '#F0C27B' },
            { color: '#4B1248' }
        ]
    }
    },
    provide: {
        heatmap: [Tooltip]
    }
}
</script>

```

{% previewsample "page.domainurl/code-snippet/heatmap-chart/axis-cs13" %}

### Label formatting

HeatMap supports formatting the axis labels by using the [labelFormat](#) property. Using this property, you can customize the axis label by global string format ('P', 'C', etc) or customized format like '{value}°C'.

### APP.VUE

```

<template>
  <div id="app">
    <ejs-heatmap id="heatmap" :xAxis='xAxis' :yAxis='yAxis'
    :legendSettings='legendSettings' :dataSource='dataSource'></ejs-heatmap>
  </div>
</template>
<script>
import Vue from 'vue';
import { HeatMapPlugin, Tooltip, Legend } from '@syncfusion/ej2-vue-heatmap';
Vue.use(HeatMapPlugin);
export default {
  data: function() {

```

```

return {
  dataSource: [
    [73, 39, 26, 39, 94, 0],
    [93, 58, 53, 38, 26, 68],
    [99, 28, 22, 4, 66, 90],
    [14, 26, 97, 69, 69, 3],
    [7, 46, 47, 47, 88, 6],
    [41, 55, 73, 23, 3, 79],
    [56, 69, 21, 86, 3, 33],
    [45, 7, 53, 81, 95, 79],
    [60, 77, 74, 68, 88, 51],
    [25, 25, 10, 12, 78, 14],
    [25, 56, 55, 58, 12, 82],
    [74, 33, 88, 23, 86, 59]
  ],
  xAxis: {
    valueType: "DateTime",
    minimum: new Date(2007, 0, 1),
    intervalType: "Years",
    labelFormat: "yyyy"
  },
  yAxis: {
    valueType: "Numeric",
    labelFormat: "${value}"
  },
  legendSettings: {
    visible: false,
  }
},
provide: {
  heatmap: [Tooltip, Legend]
}
}
</script>
<style>
@import
'https://ej2.syncfusion.com/vue/documentation/node_modules/@syncfusion/ej2-
vue-heatmap/styles/material.css';
</style>

```

{% previewsample "page.domainurl/code-snippet/heatmap-chart/axis-cs6" %}

### Axis intervals

In HeatMap, you can define an interval between the axis labels using the [interval](#) property. In date-time axis, you can change the interval mode by using the [intervalType](#) property. The date-time axis supports the following interval types:

- Years
- Months
- Days
- Hours
- Minutes

**APP.VUE**

```

<template>
  <div id="app">
    <ejs-heatmap id="heatmap" :titleSettings='titleSettings'
    :xAxis='xAxis' :yAxis='yAxis' :legendSettings='legendSettings'
    :cellSettings='cellSettings' :dataSource='dataSource'></ejs-heatmap>
  </div>
</template>
<script>
import Vue from 'vue';
import { HeatMapPlugin, Tooltip, Legend } from '@syncfusion/ej2-vue-heatmap';
Vue.use(HeatMapPlugin);
export default {
  data: function() {
    return {
      dataSource: [
        [36371, 25675, 28292, 33399, 35980, 38585, 39351, 39964, 36543,
        30529, 33298, 36985],
        [34702, 27618, 31063, 34525, 36772, 35410, 38750, 39467, 35390,
        34196, 35302, 35703],
        [34522, 31324, 32128, 34231, 36817, 34381, 37180, 38255, 32776,
        32645, 31539, 32981],
        [32213, 28755, 29517, 31214, 33747, 33507, 35763, 36837, 32910,
        33437, 30659, 31965],
        [31282, 28663, 32952, 33941, 34506, 36875, 38836, 35497, 34285,
        34094, 32256, 33699],
        [31714, 29405, 33745, 32838, 33461, 35034, 36122, 37943, 34128,
        30624, 32398, 33522],
        [32064, 28387, 33751, 32537, 34034, 35977, 37196, 38301, 33627,
        34115, 31072, 33939],
        [32417, 27868, 30807, 33386, 35284, 36126, 39753, 40978, 35777,
        35277, 31281, 35411],
        [32494, 29848, 34385, 35804, 37943, 38722, 41315, 41335, 37177,
        37443, 32457, 37304],
        [34378, 29576, 30547, 35664, 36622, 38145, 40347, 41868, 38252,
        36505, 29576, 36450],
        [35219, 31670, 32589, 34927, 36998, 39825, 41126, 42002, 37021,
        36583, 32408, 37108]
      ],
      titleSettings: {
        text: 'Monthly Flight Traffic at JFK Airport',
        textStyle: {
          size: '15px',
          fontWeight: '500',
          fontStyle: 'Normal',
          fontFamily: 'Segoe UI'
        }
      },
      xAxis: {
        valueType: "DateTime",
        minimum: new Date(2007, 0, 1),
        maximum: new Date(2017, 0, 1),
        intervalType: "Years",
        interval: 2,
        labelFormat: "yyyy"
      }
    }
  }
}

```

```

    },
    yAxis: {
      labels: ['Jan', 'Feb', 'Mar', 'Apr', 'May',
        'Jun', 'July', 'Aug', 'Sept', 'Oct', 'Nov', 'Dec']
    },
    legendSettings: {
      visible: false,
    },
    cellSettings: {
      showLabel: false,
      border: {
        width: 0,
      },
      format: '{value} flights'
    }
  }
},
provide: {
  heatmap: [Tooltip, Legend]
}
}
</script>
<style>
@import
'https://ej2.syncfusion.com/vue/documentation/node_modules/@syncfusion/ej2-
vue-heatmap/styles/material.css';
</style>

```

{% previewsample "page.domainurl/code-snippet/heatmap-chart/axis-cs7" %}

### Axis label increment

Axis label increment in the HeatMap is used to display the axis labels with regular interval values in numeric and date-time axes. The labels will be displayed with tick gaps when you set the label interval. But, to achieve the same behavior without tick gaps, use the label increment. You can set the axis label increment using the [increment](#) property and the default value of this property is 1.

### APP.VUE

```

<template>
  <div id="app">
    <ejs-heatmap id="heatmap" :xAxis='xAxis' :yAxis='yAxis'
      :dataSource='dataSource'></ejs-heatmap>
    </div>
  </template>
<script>
import Vue from 'vue';
import { HeatMapPlugin, Tooltip, Legend } from '@syncfusion/ej2-vue-heatmap';
Vue.use(HeatMapPlugin);
export default {
  data: function() {
    return {
      dataSource: [
        [73, 39, 26, 39, 94, 0],
        [93, 58, 53, 38, 26, 68],
        [99, 28, 22, 4, 66, 90],

```



```

        [14, 26, 97, 69, 69, 3],
        [7, 46, 47, 47, 88, 6],
        [41, 55, 73, 23, 3, 79],
        [56, 69, 21, 86, 3, 33],
        [45, 7, 53, 81, 95, 79],
        [60, 77, 74, 68, 88, 51],
        [25, 25, 10, 12, 78, 14],
        [25, 56, 55, 58, 12, 82],
        [74, 33, 88, 23, 86, 59]
    ],
    xAxis: {
        valueType: "Numeric",
        minimum: 0,
        increment: 2
    },
    yAxis: {
        valueType: "Numeric",
        minimum: 0,
        increment: 3
    },
    },
    },
    provide: {
        heatmap: [Tooltip, Legend]
    }
}
</script>
<style>
@import
'https://ej2.syncfusion.com/vue/documentation/node_modules/@syncfusion/ej2-
vue-heatmap/styles/material.css';
</style>

```

{% previewsample "page.domainurl/code-snippet/heatmap-chart/axis-cs8" %}

### Limiting labels in date-time axis

You can display the axis labels at specific time intervals along with the date-time axis using the [showLabelOn](#) property. This property supports the following types:

- **None**: Displays the axis labels based on the `intervalType` and `interval` property of the axis. This type is default value of the `showLabelOn` property.
- **Years**: Displays the axis labels on every year between given date-time range.
- **Months**: Displays the axis labels on every month between given date-time range.
- **Days**: Displays the axis labels on every day between given date-time range.
- **Minutes**: Displays the axis labels on every minute between given date-time range.

### APP.VUE

```

<template>
  <div id="app">
    <ejs-heatmap
      id="heatmap"
      :titleSettings="titleSettings"
      :xAxis="xAxis"
    >

```

```

      :yAxis="yAxis"
      :legendSettings="legendSettings"
      :cellSettings="cellSettings"
      height="280px"
      :paletteSettings="paletteSettings"
      :dataSource="dataSource"
      :tooltipRender="tooltipRender"
    ></ejs-heatmap>
  </div>
</template>
<script>
import Vue from "vue";
import { HeatMapPlugin, Tooltip, Legend } from "@syncfusion/ej2-vue-heatmap";
import { Internationalization } from "@syncfusion/ej2-base";
Vue.use(HeatMapPlugin);
export default {
  data: function() {
    return {
      dataSource: [
        [null, null, null, null, 16, 48, 0],
        [0, 15, 0, 24, 0, 39, 0],
        [0, 18, 37, 0, 0, 50, 0],
        [0, 10, 0, 0, 44, 5, 0],
        [0, 36, 0, 45, 20, 18, 0],
        [0, 28, 1, 42, 0, 10, 0],
        [0, 16, 32, 0, 1, 25, 0],
        [0, 31, 2, 9, 24, 0, 0],
        [0, 8, 47, 0, 0, 35, 0],
        [0, 31, 0, 0, 0, 40, 0],
        [0, 8, 0, 27, 0, 35, 0],
        [0, 12, 9, 45, 0, 8, 0],
        [0, 0, 13, 0, 22, 10, 0],
        [0, 16, 32, 0, 1, 25, 0],
        [0, 31, 2, 9, 24, 0, 0],
        [0, 8, 47, 27, 0, 35, 0],
        [0, 28, 14, 10, 0, 0, 0],
        [0, 36, 0, 45, 20, 18, 0],
        [0, 28, 1, 42, 0, 10, 0],
        [0, 31, 0, 24, 0, 40, 0],
        [0, 8, 47, 27, 0, 35, 0],
        [0, 36, 0, 45, 20, 18, 0],
        [0, 28, 1, 42, 0, 10, 0],
        [0, 31, 0, 24, 0, 40, 0],
        [0, 16, 32, 0, 1, 25, 0],
        [0, 31, 2, 9, 24, 0, 0],
        [0, 8, 47, 27, 0, 35, 0],
        [0, 10, 0, 36, 23, 19, 0],
        [0, 18, 37, 23, 0, 50, 0],
        [0, 28, 14, 10, 0, 0, 0],
        [0, 18, 37, 23, 0, 50, 0],
        [0, 18, 37, 23, 0, 50, 0],
        [0, 28, 14, 10, 0, 0, 0],
        [0, 31, 2, 9, 24, 0, 0],
        [0, 8, 47, 27, 0, 35, 0],
        [0, 10, 2, 0, 44, 5, 0],
        [0, 36, 0, 45, 20, 18, 0],

```

```

        [0, 28, 1, 42, 0, 10, 0],
        [0, 31, 0, 24, 0, 40, 1],
        [0, 16, 32, 0, 1, 25, 0],
        [0, 31, 2, 9, 24, 0, 0],
        [0, 8, 47, 27, 0, 35, 0],
        [0, 10, 2, 0, 44, 5, 0],
        [0, 12, 9, 45, 0, 8, 0],
        [0, 0, 13, 35, 22, 10, 0],
        [0, 28, 14, 10, 0, 0, 0],
        [0, 36, 0, 45, 20, 18, 2],
        [0, 28, 1, 42, 0, 10, 0],
        [0, 31, 0, 24, 0, 40, 1],
        [0, 8, 47, 27, 0, 35, 0],
        [0, 10, 2, 0, 44, 5, 0],
        [0, 31, 2, 9, 24, 0, 1],
        [0, 8, 47, 27, 0, 35, 40],
        [0, 10, 2, 0, 44, 5, null]
    ],
    titleSettings: {
        text: "Annual Summary of User Activities in GitLab",
        textStyle: {
            size: "15px",
            fontWeight: "500",
            fontStyle: "Normal",
            fontFamily: "Segoe UI"
        }
    },
    xAxis: {
        opposedPosition: true,
        valueType: "DateTime",
        minimum: new Date(2017, 6, 23),
        maximum: new Date(2018, 6, 30),
        intervalType: "Days",
        showLabelOn: "Months",
        labelFormat: "MMM",
        increment: 7
    },
    yAxis: {
        labels: ["Sun", "Mon", "Tue", "Wed", "Thu", "Fri", "Sat"],
        isInversed: true
    },
    cellSettings: {
        showLabel: false,
        border: {
            color: "white"
        }
    },
    paletteSettings: {
        palette: [
            { value: 0, color: "rgb(238,238,238)", label: "no contributions" },
            {
                value: 1,
                color: "rgb(172, 213, 242)",
                label: "1-15 contributions"
            },
            {

```

```

        value: 16,
        color: "rgb(127, 168, 201)",
        label: "16-31 contributions"
    },
    {
        value: 32,
        color: "rgb(82, 123, 160)",
        label: "31-49 contributions"
    },
    { value: 50, color: "rgb(37, 78, 119)", label: "50+ contributions"
}

],
type: "Fixed",
emptyPointColor: "white"
},
showTooltip: true,
legendSettings: {
    position: "Bottom",
    width: "20%",
    alignment: "Near",
    showLabel: true,
    labelDisplayType: "None",
    enableSmartLegend: true
},
tooltipRender: function(args) {
    let intl = new Internationalization();
    let format = intl.getDateFormat({ format: "EEE MMM dd, yyyy" });
    let newDate = new Date(args.xValue);
    let date = new Date(newDate.getTime());
    let axisLabel = args.heatmap.axisCollections[1].axisLabels;
    let index = axisLabel.indexOf(args.yLabel);
    date.setDate(date.getDate() + index);
    let value = format(date);
    args.content = [
        (args.value === 0 ? "No" : args.value) +
        " " +
        "contributions" +
        "<br>" +
        value
    ];
}
};
},
provide: {
    heatmap: [Tooltip, Legend]
}
};
</script>

```

{% previewsample "page.domainurl/code-snippet/heatmap-chart/axis-cs9" %}

### Multilevel Labels

Multilevel labels are used to classify a group of axis labels as a single category, which is then displayed with a label. By using [multiLevelLabels](#), you can add multiple levels on top of the axis labels.

To divide and group the axis labels, you can use `multiLevelLabels` property. The starting and ending indexes of the axis labels can be set using the `start` and `end` properties in the `categories`. The `text` property can be used to specify a name for the grouped axis labels.

The multilevel labels can be customized by using the following properties.

- [overflow](#) - It is used to trim or wrap the multilevel labels when the label overflows the intended space. NOTE: This property is only for x-axis.
- [alignment](#) - It is used to place and align the multilevel labels.
- [maximumTextWidth](#) - It is used to set the maximum width of the text. When the text length exceeds the maximum text width, the overflow action will be performed.
- [textStyle](#) - It is used to customize the font style of the multilevel labels.
- [border](#) - It is used to customize the border of the multilevel labels displayed in the x-axis and y-axis.

## APP.VUE

```
<template>
  <div id="app">
    <ejs-heatmap id="heatmap" :xAxis='xAxis' :yAxis='yAxis'
:dataSource='dataSource'
      :titleSettings='titleSettings' :paletteSettings='paletteSettings'
:legendSettings='legendSettings'></ejs-heatmap>
    </div>
  </template>
<script>
import Vue from 'vue';
import { HeatMapPlugin, Tooltip, Legend } from '@syncfusion/ej2-vue-heatmap';
Vue.use(HeatMapPlugin);
export default {
  data: function() {
    return {
      dataSource:[
        [52, 65, 67, 45, 37, 52,32, 76, 60, 64, 82, 91],
        [68, 52, 63, 51, 30, 51,51, 81, 70, 60, 88, 80],
        [60, 50, 42, 53, 66, 70,41, 69, 76, 74, 86, 97],
        [66, 64, 46, 40, 47, 41, 45, 76, 83, 69, 92,84],
        [65, 42, 58, 32, 36, 44,49, 79, 83, 69, 83, 93],
        [54, 46, 61, 46, 40, 39,41, 69, 61, 84, 84, 87],
        [48, 46, 61, 47, 49, 41,41, 67, 78, 83, 98, 87],
        [69, 52, 41, 44, 41, 52,46, 71, 63, 84, 83, 91],
        [50, 59, 44, 43, 27, 42,26, 64, 76, 65, 81, 86],
        [47, 49, 66, 53, 50, 34,31, 79, 78, 79, 89, 95],
        [61, 40, 62, 26, 34, 54,56, 74, 83, 78, 95, 98]
      ],
      titleSettings: {
        text: 'Product wise Monthly sales revenue for a e-commerce
website',
        textStyle: {
          size: '15px',
          fontWeight: '500',
          fontStyle: 'Normal',
          fontFamily: 'Segoe UI'
        }
      }
    }
  }
}
```

```

    },
    xAxis: {
      labels: ['Laptop', 'Mobile', 'Gaming', 'Cosmetics', 'Fragrance',
'Watches', 'Handbags', 'Apparels', 'Kitchenware', 'Furniture', 'Home
Decor'],
      border: { type: 'Rectangle', width:1, color: '#a19d9d' },
      multiLevelLabels: [
        {
          overflow:'Trim',
          alignment: 'Near',
          textStyle: {
            color: 'black',
            fontWeight: 'Bold'
          },
        },
        border: { type: 'Rectangle', color: '#a19d9d' },
        categories: [
          { start: 0, end: 2, text: 'Electronics', },
          { start: 3, end: 4, text: 'Beauty and personal
care', maximumTextWidth: 50},
          { start: 5, end: 7, text: 'Fashion', },
          { start: 8, end: 10, text: 'Household', }
        ]
      ]
    },
    yAxis:{
      labels: ['Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun', 'Jul', 'Aug',
'Sep', 'Oct', 'Nov', 'Dec'],
      multiLevelLabels: [
        {
          border: { type: 'Brace', color: '#a19d9d' },
          categories: [
            { start: 0, end: 2, text: 'Q1' },
            { start: 3, end: 5, text: 'Q2' },
            { start: 6, end: 8, text: 'Q3' },
            { start: 9, end: 11, text: 'Q4' }
          ]
        },
        {
          border: { type: 'Brace', color: '#a19d9d' },
          categories: [
            { start: 0, end: 5, text: 'First Half Yearly' },
            {
              start: 6,
              end: 11,
              text: 'Second Half Yearly'
            }
          ]
        },
        {
          border: { type: 'Brace', color: '#a19d9d' }
          categories: [
            { start: 0, end: 11, text: 'Yearly' }
          ]
        }
      ]
    },
  },

```

```

        legendSettings: {
            visible: false
        },
        paletteSettings: {
            palette: [{ color: '#F0C27B' },
                { color: '#4B1248' }
            ],
        }
    },
    provide: {
        heatmap: [Tooltip, Legend]
    }
}
</script>
<style>
@import '../node_modules/@syncfusion/ej2-vue-heatmap/styles/material.css';
</style>

```

{% previewsample "page.domainurl/code-snippet/heatmap-chart/axis-cs10" %}

### Palette in Vue Heatmap chart component

In heat map, each data point is displayed as a cell with applied color based on the data value. The palette in the heat map is used to define the color range for cells and gradient type for colors. You can define the colors either in RGB or hex codes using the [color](#) property in the [palette](#). The defined colors are applied to the cell background based on the palette type and cell value.

#### Palette types

You can display the heat map cells either in gradient colors or fixed colors.

##### Gradient

The smooth transition between the given palette colors can be applied for the heat map cells based on value. The heat map calculates all the gradient colors between the start and end colors for all distinct data values. Default start color and end color will be considered for gradient calculation, if the colors are not defined. The palette type must be defined as **Gradient** for the [type](#) property in the [paletteSettings](#) property.

#### APP.VUE

```

<template>
  <div id="app">
    <ejs-heatmap id="heatmap" :titleSettings='titleSettings'
      :xAxis='xAxis' :yAxis='yAxis' :dataSource='dataSource'
      :paletteSettings='paletteSettings' :legendSettings='legendSettings'></ejs-heatmap>
  </div>
</template>
<script>
import Vue from 'vue';
import { HeatMapPlugin, Tooltip, Legend } from '@syncfusion/ej2-vue-heatmap';
Vue.use(HeatMapPlugin);
export default {
  data: function() {
    return {

```

```

        dataSource: [
            [73, 39, 26, 39, 94, 0],
            [93, 58, 53, 38, 26, 68],
            [99, 28, 22, 4, 66, 90],
            [14, 26, 97, 69, 69, 3],
            [7, 46, 47, 47, 88, 6],
            [41, 55, 73, 23, 3, 79],
            [56, 69, 21, 86, 3, 33],
            [45, 7, 53, 81, 95, 79],
            [60, 77, 74, 68, 88, 51],
            [25, 25, 10, 12, 78, 14],
            [25, 56, 55, 58, 12, 82],
            [74, 33, 88, 23, 86, 59]
        ],
        titleSettings: {
            text: 'Sales Revenue per Employee (in 1000 US$)',
            textStyle: {
                size: '15px',
                fontWeight: '500',
                fontStyle: 'Normal',
                fontFamily: 'Segoe UI'
            }
        },
        xAxis: {
            labels: ['Nancy', 'Andrew', 'Janet', 'Margaret', 'Steven',
                'Michael', 'Robert', 'Laura', 'Anne', 'Paul', 'Karin',
                'Mario'],
        },
        yAxis: {
            labels: ['Mon', 'Tues', 'Wed', 'Thurs', 'Fri', 'Sat'],
        },
        paletteSettings: {
            palette: [
                { color: '#C06C84' },
                { color: '#6C5B7B' },
                { color: '#355C7D' }
            ],
            type: "Gradient"
        },
        legendSettings: {
            visible: true,
        }
    },
    provide: {
        heatmap: [Tooltip, Legend]
    }
}
</script>
<style>
@import
'https://ej2.syncfusion.com/vue/documentation/node_modules/@syncfusion/ej2-
vue-heatmap/styles/material.css';
</style>

```

{% previewsample "page.domainurl/code-snippet/heatmap-chart/palette-cs1" %}



*Fixed*

In fixed palette type, solid colors are applied to the heat map cells. The data values can be grouped based on the number of colors defined for the heat map. The palette type should be defined as **Fixed** for the [type](#) property in the `paletteSettings` property.

**APP.VUE**

```
<template>
  <div id="app">
    <ejs-heatmap id="heatmap" :titleSettings='titleSettings'
      :xAxis='xAxis' :yAxis='yAxis' :dataSource='dataSource'
      :paletteSettings='paletteSettings' :legendSettings='legendSettings'></ejs-
      heatmap>
    </div>
  </template>
<script>
import Vue from 'vue';
import { HeatMapPlugin, Tooltip, Legend } from '@syncfusion/ej2-vue-
heatmap';
Vue.use(HeatMapPlugin);
export default {
  data: function() {
    return {
      dataSource: [
        [73, 39, 26, 39, 94, 0],
        [93, 58, 53, 38, 26, 68],
        [99, 28, 22, 4, 66, 90],
        [14, 26, 97, 69, 69, 3],
        [7, 46, 47, 47, 88, 6],
        [41, 55, 73, 23, 3, 79],
        [56, 69, 21, 86, 3, 33],
        [45, 7, 53, 81, 95, 79],
        [60, 77, 74, 68, 88, 51],
        [25, 25, 10, 12, 78, 14],
        [25, 56, 55, 58, 12, 82],
        [74, 33, 88, 23, 86, 59]
      ],
      titleSettings: {
        text: 'Sales Revenue per Employee (in 1000 US$)',
        textStyle: {
          size: '15px',
          fontWeight: '500',
          fontStyle: 'Normal',
          fontFamily: 'Segoe UI'
        }
      },
      xAxis: {
        labels: ['Nancy', 'Andrew', 'Janet', 'Margaret', 'Steven',
          'Michael', 'Robert', 'Laura', 'Anne', 'Paul', 'Karin',
          'Mario'],
      },
      yAxis: {
        labels: ['Mon', 'Tues', 'Wed', 'Thurs', 'Fri', 'Sat'],
      },
      paletteSettings: {
        palette: [
          { color: '#C06C84'},

```

```

        { color: '#6C5B7B'},
        { color: '#355C7D'}
    ],
    type: "Fixed"
  },
  legendSettings: {
    visible: true,
  }
}
},
provide:{
  heatmap:[Tooltip, Legend]
}
}
</script>
<style>
@import
'https://ej2.syncfusion.com/vue/documentation/node_modules/@syncfusion/ej2-
vue-heatmap/styles/material.css';
</style>

```

{% previewsample "page.domainurl/code-snippet/heatmap-chart/palette-cs2" %}

### Defining color stops

You can define the colors ranges or color stops for data values in both gradient and fixed palette types. You need to define the data value in the [value](#) property for [palette](#) property to calculate the color stops. The heat map automatically calculates the color stops if the [value](#) property is not defined. The [label](#) property is used to provide the additional information about the color that is to be displayed in the legend. If the label is not provided, the value is displayed in the legend. The labels can be automatically calculated based on data values, if both the values and labels are not defined.

### APP.VUE

```

<template>
  <div id="app">
    <ejs-heatmap id="heatmap" :titleSettings='titleSettings'
    :xAxis='xAxis' :yAxis='yAxis' :dataSource='dataSource'
    :paletteSettings='paletteSettings' :legendSettings='legendSettings'></ejs-
    heatmap>
  </div>
</template>
<script>
import Vue from 'vue';
import { HeatMapPlugin, Tooltip, Legend } from '@syncfusion/ej2-vue-
heatmap';
Vue.use(HeatMapPlugin);
export default {
  data: function() {
    return {
      dataSource: [
        [73, 39, 26, 39, 94, 0],
        [93, 58, 53, 38, 26, 68],
        [99, 28, 22, 4, 66, 90],
        [14, 26, 97, 69, 69, 3],
        [7, 46, 47, 47, 88, 6],

```

```

        [41, 55, 73, 23, 3, 79],
        [56, 69, 21, 86, 3, 33],
        [45, 7, 53, 81, 95, 79],
        [60, 77, 74, 68, 88, 51],
        [25, 25, 10, 12, 78, 14],
        [25, 56, 55, 58, 12, 82],
        [74, 33, 88, 23, 86, 59]
    ],
    titleSettings: {
        text: 'Sales Revenue per Employee (in 1000 US$)',
        textStyle: {
            size: '15px',
            fontWeight: '500',
            fontStyle: 'Normal',
            fontFamily: 'Segoe UI'
        }
    },
    xAxis: {
        labels: ['Nancy', 'Andrew', 'Janet', 'Margaret', 'Steven',
            'Michael', 'Robert', 'Laura', 'Anne', 'Paul', 'Karin',
            'Mario'],
    },
    yAxis: {
        labels: ['Mon', 'Tues', 'Wed', 'Thurs', 'Fri', 'Sat'],
    },
    paletteSettings: {
        palette: [
            { color: '#C06C84', label: 'Low', value: 50 },
            { color: '#6C5B7B', label: 'Moderate', value: 80 },
            { color: '#355C7D', label: 'High', value: 100 }
        ],
        type: "Gradient"
    },
    legendSettings: {
        visible: true,
    }
    },
    provide: {
        heatmap: [Tooltip, Legend]
    }
}
</script>
<style>
@import '../node_modules/@syncfusion/ej2-vue-heatmap/styles/material.css';
</style>

```

{% previewsample "page.domainurl/code-snippet/heatmap-chart/palette-cs3" %}

### Color range

The color range support is used to provide a specific color for specific range in heat map. The `startValue` and `endValue` properties are used to define the range start and end value. The `minColor` and `maxColor` properties represent the colors of given range. It's possible to set the cell color for the value not in the given range using the `fillColor` property.

In Fixed type, the `minColor` value is used for cell color.

#### APP.VUE

```
<template>
  <div id="app">
    <ejs-heatmap id="heatmap" :titleSettings='titleSettings'
    :xAxis='xAxis' :yAxis='yAxis' :dataSource='dataSource'
    :paletteSettings='paletteSettings' :legendSettings='legendSettings'></ejs-
    heatmap>
  </div>
</template>
<script>
import Vue from 'vue';
import { HeatMapPlugin, Tooltip, Legend } from '@syncfusion/ej2-vue-
heatmap';
Vue.use(HeatMapPlugin);
export default {
  data: function() {
    return {
      dataSource: [
        [73, 39, 26, 39, 94, 1],
        [93, 58, 53, 38, 26, 68],
        [99, 28, 22, 4, 66, 90],
        [14, 26, 97, 69, 69, 3],
        [7, 46, 47, 47, 88, 6],
        [41, 55, 73, 23, 3, 79],
        [56, 69, 21, 86, 3, 33],
        [45, 7, 53, 81, 95, 79],
        [60, 77, 74, 68, 88, 51],
        [25, 25, 10, 12, 78, 14],
        [25, 56, 55, 58, 12, 82],
        [74, 33, 88, 23, 86, 59]
      ],
      titleSettings: {
        text: 'Sales Revenue per Employee (in 1000 US$)',
        textStyle: {
          size: '15px',
          fontWeight: '500',
          fontStyle: 'Normal',
          fontFamily: 'Segoe UI'
        }
      },
      xAxis: {
        labels: ['Nancy', 'Andrew', 'Janet', 'Margaret', 'Steven',
        'Michael', 'Robert', 'Laura', 'Anne', 'Paul', 'Karin',
        'Mario'],
      },
      yAxis: {
        labels: ['Mon', 'Tues', 'Wed', 'Thurs', 'Fri', 'Sat'],
      },
      paletteSettings: {
        palette: [
          { startValue:1, endValue:30, minColor: '#C2E7EC', maxColor:
          '#AEDFE6' },
          { startValue:30, endValue:60, minColor: '#9AD7E0', maxColor:
          '#72C7D4' },

```

```

        { startValue:60, endValue:90, minColor: '#5EBFCE', maxColor:
        '#4AB7C8' }
      ],
      type: "Gradient"
    },
    legendSettings: {
      visible: true,
    }
  }
},
provide:{
  heatmap:[Tooltip, Legend]
}
}
</script>
<style>
@import '../node_modules/@syncfusion/ej2-vue-heatmap/styles/material.css';
</style>

```

{% previewsample "page.domainurl/code-snippet/heatmap-chart/palette-cs4" %}

See Also

- [How to enable smart legend](#)

### Legend in Vue Heatmap chart component

The legend is used to provide the information about the heat map cell. You can enable the legend by setting the [visible](#) property to **true** and injecting the **Legend** module into the **provide**.

#### APP.VUE

```

<template>
  <div id="app">
    <ejs-heatmap id="heatmap" :titleSettings='titleSettings'
    :xAxis='xAxis' :yAxis='yAxis' :dataSource='dataSource'
    :paletteSettings='paletteSettings' :cellSettings='cellSettings'
    :legendSettings='legendSettings'></ejs-heatmap>
  </div>
</template>
<script>
import Vue from 'vue';
import { HeatMapPlugin, Tooltip, Legend } from '@syncfusion/ej2-vue-heatmap';
Vue.use(HeatMapPlugin);
export default {
  data: function() {
    return {
      dataSource: [
        [73, 39, 26, 39, 94, 0],
        [93, 58, 53, 38, 26, 68],
        [99, 28, 22, 4, 66, 90],
        [14, 26, 97, 69, 69, 3],
        [7, 46, 47, 47, 88, 6],
        [41, 55, 73, 23, 3, 79],
        [56, 69, 21, 86, 3, 33],

```

```

        [45, 7, 53, 81, 95, 79],
        [60, 77, 74, 68, 88, 51],
        [25, 25, 10, 12, 78, 14],
        [25, 56, 55, 58, 12, 82],
        [74, 33, 88, 23, 86, 59]
    ],
    titleSettings: {
        text: 'Sales Revenue per Employee (in 1000 US$)',
        textStyle: {
            size: '15px',
            fontWeight: '500',
            fontStyle: 'Normal',
            fontFamily: 'Segoe UI'
        }
    },
    xAxis: {
        labels: ['Nancy', 'Andrew', 'Janet', 'Margaret', 'Steven',
            'Michael', 'Robert', 'Laura', 'Anne', 'Paul', 'Karin',
            'Mario']
    },
    yAxis: {
        labels: ['Mon', 'Tues', 'Wed', 'Thurs', 'Fri', 'Sat']
    },
    cellSettings: {
        showLabel: false,
    },
    paletteSettings: {
        palette: [
            { value: 0, color: '#C2E7EC' },
            { value: 10, color: '#AEDFE6' },
            { value: 20, color: '#9AD7E0' },
            { value: 30, color: '#72C7D4' },
            { value: 40, color: '#5EBFCE' },
            { value: 50, color: '#4AB7C8' },
            { value: 60, color: '#309DAE' },
            { value: 70, color: '#2B8C9B' },
            { value: 80, color: '#206974' },
            { value: 90, color: '#15464D' },
            { value: 100, color: '#000000' },
        ]
    },
    legendSettings: {
        position: 'Right',
    }
},
provide: {
    heatmap: [Tooltip, Legend]
}
</script>
<style>
@import
'https://ej2.syncfusion.com/vue/documentation/node_modules/@syncfusion/ej2-
vue-heatmap/styles/material.css';
</style>

```

```
{% previewsample "page.domainurl/code-snippet/heatmap-chart/legend-cs1" %}
```

### Legend types

Heat map supports two legend types: Gradient and list type.

- Gradient - This is a continuous color legend with smooth color transition between palette color values.
- List - List is a fixed color legend. Each palette color information is shown separately in the list item.

You can change the legend type by using the [type](#) property in the `paletteSettings` property.

### APP.VUE

```
<template>
  <div id="app">
    <ejs-heatmap id="heatmap" :titleSettings='titleSettings'
    :xAxis='xAxis' :yAxis='yAxis' :dataSource='dataSource'
    :paletteSettings='paletteSettings' :cellSettings='cellSettings'
    :legendSettings='legendSettings'></ejs-heatmap>
  </div>
</template>
<script>
import Vue from 'vue';
import { HeatMapPlugin, Tooltip, Legend } from '@syncfusion/ej2-vue-heatmap';
Vue.use(HeatMapPlugin);
export default {
  data: function() {
    return {
      dataSource: [
        [73, 39, 26, 39, 94, 0],
        [93, 58, 53, 38, 26, 68],
        [99, 28, 22, 4, 66, 90],
        [14, 26, 97, 69, 69, 3],
        [7, 46, 47, 47, 88, 6],
        [41, 55, 73, 23, 3, 79],
        [56, 69, 21, 86, 3, 33],
        [45, 7, 53, 81, 95, 79],
        [60, 77, 74, 68, 88, 51],
        [25, 25, 10, 12, 78, 14],
        [25, 56, 55, 58, 12, 82],
        [74, 33, 88, 23, 86, 59]
      ],
      titleSettings: {
        text: 'Sales Revenue per Employee (in 1000 US$)',
        textStyle: {
          size: '15px',
          fontWeight: '500',
          fontStyle: 'Normal',
          fontFamily: 'Segoe UI'
        }
      },
      xAxis: {
```

```

        labels: ['Nancy', 'Andrew', 'Janet', 'Margaret', 'Steven',
        'Michael', 'Robert', 'Laura', 'Anne', 'Paul', 'Karin',
        'Mario']
    },
    yAxis: {
        labels: ['Mon', 'Tues', 'Wed', 'Thurs', 'Fri', 'Sat']
    },
    cellSettings: {
        showLabel: false,
    },
    paletteSettings: {
        type: 'Fixed'
    },
    legendSettings: {
        position: 'Right',
    }
    }
    },
    provide: {
        heatmap: [Tooltip, Legend]
    }
    }
</script>
<style>
@import
'https://ej2.syncfusion.com/vue/documentation/node_modules/@syncfusion/ej2-
vue-heatmap/styles/material.css';
</style>

```

{% previewsample "page.domainurl/code-snippet/heatmap-chart/legend-cs2" %}

### Placement

You can place the legend at left, right, top, or bottom to the heat map layout by using the [position](#) property. The legend is positioned at the right to the heat map by default.

### APP.VUE

```

<template>
  <div id="app">
    <ejs-heatmap id="heatmap" :titleSettings='titleSettings'
    :xAxis='xAxis' :yAxis='yAxis' :dataSource='dataSource'
    :legendSettings='legendSettings'></ejs-heatmap>
  </div>
</template>
<script>
import Vue from 'vue';
import { HeatMapPlugin, Tooltip, Legend } from '@syncfusion/ej2-vue-
heatmap';
Vue.use(HeatMapPlugin);
export default {
  data: function() {
    return {
      dataSource: [
        [73, 39, 26, 39, 94, 0],
        [93, 58, 53, 38, 26, 68],
        [99, 28, 22, 4, 66, 90],

```



```

        [14, 26, 97, 69, 69, 3],
        [7, 46, 47, 47, 88, 6],
        [41, 55, 73, 23, 3, 79],
        [56, 69, 21, 86, 3, 33],
        [45, 7, 53, 81, 95, 79],
        [60, 77, 74, 68, 88, 51],
        [25, 25, 10, 12, 78, 14],
        [25, 56, 55, 58, 12, 82],
        [74, 33, 88, 23, 86, 59]
    ],
    titleSettings: {
        text: 'Sales Revenue per Employee (in 1000 US$)',
        textStyle: {
            size: '15px',
            fontWeight: '500',
            fontStyle: 'Normal',
            fontFamily: 'Segoe UI'
        }
    },
    xAxis: {
        labels: ['Nancy', 'Andrew', 'Janet', 'Margaret', 'Steven',
            'Michael', 'Robert', 'Laura', 'Anne', 'Paul', 'Karin',
            'Mario']
    },
    yAxis: {
        labels: ['Mon', 'Tues', 'Wed', 'Thurs', 'Fri', 'Sat']
    },
    legendSettings: {
        position: 'Top',
    }
}
},
provide:{
    heatmap:[Tooltip, Legend]
}
}
</script>
<style>
@import
'https://ej2.syncfusion.com/vue/documentation/node_modules/@syncfusion/ej2-
vue-heatmap/styles/material.css';
</style>

```

{% previewsample "page.domainurl/code-snippet/heatmap-chart/legend-cs3" %}

### Alignment

You can align the legend as center, far, or near to the heat map using the [alignment](#) property.

### APP.VUE

```

<template>
    <div id="app">
        <ejs-heatmap id="heatmap" :titleSettings='titleSettings'
        :xAxis='xAxis' :yAxis='yAxis' :dataSource='dataSource'
        :legendSettings='legendSettings'></ejs-heatmap>
    </div>

```

```

</template>
<script>
import Vue from 'vue';
import { HeatMapPlugin, Tooltip, Legend } from '@syncfusion/ej2-vue-heatmap';
Vue.use(HeatMapPlugin);
export default {
  data: function() {
    return {
      dataSource: [
        [73, 39, 26, 39, 94, 0],
        [93, 58, 53, 38, 26, 68],
        [99, 28, 22, 4, 66, 90],
        [14, 26, 97, 69, 69, 3],
        [7, 46, 47, 47, 88, 6],
        [41, 55, 73, 23, 3, 79],
        [56, 69, 21, 86, 3, 33],
        [45, 7, 53, 81, 95, 79],
        [60, 77, 74, 68, 88, 51],
        [25, 25, 10, 12, 78, 14],
        [25, 56, 55, 58, 12, 82],
        [74, 33, 88, 23, 86, 59]
      ],
      titleSettings: {
        text: 'Sales Revenue per Employee (in 1000 US$)',
        textStyle: {
          size: '15px',
          fontWeight: '500',
          fontStyle: 'Normal',
          fontFamily: 'Segoe UI'
        }
      },
      xAxis: {
        labels: ['Nancy', 'Andrew', 'Janet', 'Margaret', 'Steven', 'Michael', 'Robert', 'Laura', 'Anne', 'Paul', 'Karin', 'Mario']
      },
      yAxis: {
        labels: ['Mon', 'Tues', 'Wed', 'Thurs', 'Fri', 'Sat']
      },
      legendSettings: {
        position: 'Right',
        alignment: 'Near',
        height: '150px'
      }
    }
  },
  provide: {
    heatmap: [Tooltip, Legend]
  }
}
</script>
<style>
@import
'https://ej2.syncfusion.com/vue/documentation/node_modules/@syncfusion/ej2-vue-heatmap/styles/material.css';
</style>

```

{% previewsample "page.domainurl/code-snippet/heatmap-chart/legend-cs4" %}

### Legend dimensions

You can change the legend dimensions with values in pixels or percentage by using the [width](#) and [height](#) properties.

### APP.VUE

```
<template>
  <div id="app">
    <ejs-heatmap id="heatmap" :titleSettings='titleSettings'
    :xAxis='xAxis' :yAxis='yAxis' :dataSource='dataSource'
    :legendSettings='legendSettings'></ejs-heatmap>
  </div>
</template>
<script>
import Vue from 'vue';
import { HeatMapPlugin, Tooltip, Legend } from '@syncfusion/ej2-vue-heatmap';
Vue.use(HeatMapPlugin);
export default {
  data: function() {
    return {
      dataSource: [
        [73, 39, 26, 39, 94, 0],
        [93, 58, 53, 38, 26, 68],
        [99, 28, 22, 4, 66, 90],
        [14, 26, 97, 69, 69, 3],
        [7, 46, 47, 47, 88, 6],
        [41, 55, 73, 23, 3, 79],
        [56, 69, 21, 86, 3, 33],
        [45, 7, 53, 81, 95, 79],
        [60, 77, 74, 68, 88, 51],
        [25, 25, 10, 12, 78, 14],
        [25, 56, 55, 58, 12, 82],
        [74, 33, 88, 23, 86, 59]
      ],
      titleSettings: {
        text: 'Sales Revenue per Employee (in 1000 US$)',
        textStyle: {
          size: '15px',
          fontWeight: '500',
          fontStyle: 'Normal',
          fontFamily: 'Segoe UI'
        }
      },
      xAxis: {
        labels: ['Nancy', 'Andrew', 'Janet', 'Margaret', 'Steven',
        'Michael', 'Robert', 'Laura', 'Anne', 'Paul', 'Karin',
        'Mario']
      },
      yAxis: {
        labels: ['Mon', 'Tues', 'Wed', 'Thurs', 'Fri', 'Sat']
      },
      legendSettings: {
```

```

        position: 'Right',
        height: '150px'
    }
    },
    provide:{
        heatmap:[Tooltip, Legend]
    }
}
</script>
<style>
@import '../node_modules/@syncfusion/ej2-vue-heatmap/styles/material.css';
</style>

```

{% previewsample "page.domainurl/code-snippet/heatmap-chart/legend-cs5" %}

### Paging for legend

Paging is available only for the list type legend in the heat map, and it can be enabled by default, when the legend items exceed the legend bounds. You can view each legend items by navigating between the pages using navigation buttons.

### APP.VUE

```

<template>
    <div id="app">
        <ejs-heatmap id="heatmap" :titleSettings='titleSettings'
        :xAxis='xAxis' :yAxis='yAxis' :dataSource='dataSource'
        :legendSettings='legendSettings' :paletteSettings='paletteSettings'></ejs-
        heatmap>
    </div>
</template>
<script>
import Vue from 'vue';
import { HeatMapPlugin, Tooltip, Legend } from '@syncfusion/ej2-vue-
heatmap';
Vue.use(HeatMapPlugin);
export default {
    data: function() {
        return {
            dataSource: [
                [73, 39, 26, 39, 94, 0],
                [93, 58, 53, 38, 26, 68],
                [99, 28, 22, 4, 66, 90],
                [14, 26, 97, 69, 69, 3],
                [7, 46, 47, 47, 88, 6],
                [41, 55, 73, 23, 3, 79],
                [56, 69, 21, 86, 3, 33],
                [45, 7, 53, 81, 95, 79],
                [60, 77, 74, 68, 88, 51],
                [25, 25, 10, 12, 78, 14],
                [25, 56, 55, 58, 12, 82],
                [74, 33, 88, 23, 86, 59]
            ],
            titleSettings: {
                text: 'Sales Revenue per Employee (in 1000 US$)',
                textStyle: {

```

```

        size: '15px',
        fontWeight: '500',
        fontStyle: 'Normal',
        fontFamily: 'Segoe UI'
    },
    },
    xAxis: {
        labels: ['Nancy', 'Andrew', 'Janet', 'Margaret', 'Steven',
        'Michael', 'Robert', 'Laura', 'Anne', 'Paul', 'Karin',
'Mario'],
    },
    yAxis: {
        labels: ['Mon', 'Tues', 'Wed', 'Thurs', 'Fri', 'Sat']
    },
    paletteSettings: {
        palette: [
            { value: 0, color: '#C2E7EC' },
            { value: 10, color: '#AEDFE6' },
            { value: 20, color: '#9AD7E0' },
            { value: 25, color: '#86CFDA' },
            { value: 30, color: '#72C7D4' },
            { value: 40, color: '#5EBFCE' },
            { value: 50, color: '#4AB7C8' },
            { value: 55, color: '#36AFC2' },
            { value: 60, color: '#309DAE' },
            { value: 70, color: '#2B8C9B' },
            { value: 75, color: '#257A87' },
            { value: 80, color: '#206974' },
            { value: 85, color: '#1B5761' },
            { value: 90, color: '#15464D' },
            { value: 100, color: '#000000' },
        ],
        type: "Fixed"
    },
    legendSettings: {
        position: 'Right',
        height: '150px'
    }
    },
    provide:{
        heatmap:[Tooltip, Legend]
    }
}
</script>
<style>
@import
'https://ej2.syncfusion.com/vue/documentation/node_modules/@syncfusion/ej2-
vue-heatmap/styles/material.css';
</style>

```

{% previewsample "page.domainurl/code-snippet/heatmap-chart/legend-cs6" %}

### Smart Legend

Smart legend is another way of showing list type legend with responsiveness and readability, when the palette has more number of items. You can enable this smart legend by using the [enableSmartLegend](#) property when the palette type is set to **Fixed**.

In smart legend, you can change the display type of legend labels by using the [labelDisplayType](#) property.

The following are the legend label display types:

- All: Displays all labels in the legend.
- Edge: Displays the legend labels only at extreme ends.
- None: None of the labels are displayed. The tooltip will appear for this type of label display when hovering over the legend item.

### APP.VUE

```
<template>
  <div id="app">
    <ejs-heatmap id="heatmap" :titleSettings='titleSettings'
    :xAxis='xAxis' :yAxis='yAxis' :dataSource='dataSource'
    :legendSettings='legendSettings' :cellSettings='cellSettings'
    :paletteSettings='paletteSettings'></ejs-heatmap>
  </div>
</template>
<script>
import Vue from 'vue';
import { HeatMapPlugin, Tooltip, Legend } from '@syncfusion/ej2-vue-heatmap';
Vue.use(HeatMapPlugin);
export default {
  data: function() {
    return {
      dataSource: [
        [73, 39, 26, 39, 94, 0],
        [93, 58, 53, 38, 26, 68],
        [99, 28, 22, 4, 66, 90],
        [14, 26, 97, 69, 69, 3],
        [7, 46, 47, 47, 88, 6],
        [41, 55, 73, 23, 3, 79],
        [56, 69, 21, 86, 3, 33],
        [45, 7, 53, 81, 95, 79],
        [60, 77, 74, 68, 88, 51],
        [25, 25, 10, 12, 78, 14],
        [25, 56, 55, 58, 12, 82],
        [74, 33, 88, 23, 86, 59]
      ],
      titleSettings: {
        text: 'Sales Revenue per Employee (in 1000 US$)',
        textStyle: {
          size: '15px',
          fontWeight: '500',
          fontStyle: 'Normal',
          fontFamily: 'Segoe UI'
        }
      }
    }
  }
}
```

```

    },
    xAxis: {
      labels: ['Nancy', 'Andrew', 'Janet', 'Margaret', 'Steven',
        'Michael', 'Robert', 'Laura', 'Anne', 'Paul', 'Karin',
'Mario']
    },
    yAxis: {
      labels: ['Mon', 'Tues', 'Wed', 'Thurs', 'Fri', 'Sat']
    },
    cellSettings: {
      showLabel: false,
    },
    paletteSettings: {
      palette: [
        { value: 0, color: '#C2E7EC' },
        { value: 10, color: '#AEDFE6' },
        { value: 20, color: '#9AD7E0' },
        { value: 30, color: '#72C7D4' },
        { value: 40, color: '#5EBFCE' },
        { value: 50, color: '#4AB7C8' },
        { value: 60, color: '#309DAE' },
        { value: 70, color: '#2B8C9B' },
        { value: 80, color: '#206974' },
        { value: 90, color: '#15464D' },
        { value: 100, color: '#000000' },
      ],
      type: "Fixed"
    },
    legendSettings: {
      position: 'Bottom',
      enableSmartLegend: true,
      width: '75%'
    }
  },
  provide: {
    heatmap: [Tooltip, Legend]
  }
}
</script>
<style>
@import
'https://ej2.syncfusion.com/vue/documentation/node_modules/@syncfusion/ej2-
vue-heatmap/styles/material.css';
</style>

```

{% previewsample "page.domainurl/code-snippet/heatmap-chart/legend-cs7" %}

### Legend Selection

In the HeatMap, the legend selection is used to toggle the visibility of cell for viewing the specific range value. You can enable the legend selection using the [toggleVisibility](#) property.

### APP.VUE

```

<template>
  <div id="app">

```

```

    <ejs-heatmap id="heatmap" :titleSettings='titleSettings'
    :xAxis='xAxis' :yAxis='yAxis' :dataSource='dataSource'
    :legendSettings='legendSettings' :cellSettings='cellSettings'
    :paletteSettings='paletteSettings'></ejs-heatmap>
  </div>
</template>
<script>
import Vue from 'vue';
import { HeatMapPlugin, Tooltip, Legend } from '@syncfusion/ej2-vue-heatmap';
Vue.use(HeatMapPlugin);
export default {
  data: function() {
    return {
      dataSource: [
        [73, 39, 26, 39, 94, 0],
        [93, 58, 53, 38, 26, 68],
        [99, 28, 22, 4, 66, 90],
        [14, 26, 97, 69, 69, 3],
        [7, 46, 47, 47, 88, 6],
        [41, 55, 73, 23, 3, 79],
        [56, 69, 21, 86, 3, 33],
        [45, 7, 53, 81, 95, 79],
        [60, 77, 74, 68, 88, 51],
        [25, 25, 10, 12, 78, 14],
        [25, 56, 55, 58, 12, 82],
        [74, 33, 88, 23, 86, 59]
      ],
      titleSettings: {
        text: 'Sales Revenue per Employee (in 1000 US$)',
        textStyle: {
          size: '15px',
          fontWeight: '500',
          fontStyle: 'Normal',
          fontFamily: 'Segoe UI'
        }
      },
      xAxis: {
        labels: ['Nancy', 'Andrew', 'Janet', 'Margaret', 'Steven',
          'Michael', 'Robert', 'Laura', 'Anne', 'Paul', 'Karin',
          'Mario']
      },
      yAxis: {
        labels: ['Mon', 'Tues', 'Wed', 'Thurs', 'Fri', 'Sat']
      },
      cellSettings: {
        showLabel: false,
      },
      paletteSettings: {
        palette: [
          { value: 0, color: '#C2E7EC' },
          { value: 10, color: '#AEDFE6' },
          { value: 20, color: '#9AD7E0' },
          { value: 30, color: '#72C7D4' },
          { value: 40, color: '#5EBFCE' },
          { value: 50, color: '#4AB7C8' },
          { value: 60, color: '#309DAE' },
        ]
      }
    }
  }
}

```



```

        { value: 70, color: '#2B8C9B' },
        { value: 80, color: '#206974' },
        { value: 90, color: '#15464D' },
        { value: 100, color: '#000000' },
      ],
      type: "Fixed"
    },
    legendSettings: {
      position: 'Bottom',
      enableSmartLegend: true,
      width: '75%',
      toggleVisibility: true
    }
  },
  provide: {
    heatmap: [Tooltip, Legend]
  }
}
</script>
<style>
@import
'https://ej2.syncfusion.com/vue/documentation/node_modules/@syncfusion/ej2-
vue-heatmap/styles/material.css';
</style>

```

{% previewsample "page.domainurl/code-snippet/heatmap-chart/legend-cs8" %}

### Legend Title

The legend title displays a specific information about the legend. You can enable the legend title by setting the [title](#) property by providing the text and customizing the legend title text style using the [textStyle](#) property.

### APP.VUE

```

<template>
  <div id="app">
    <ejs-heatmap id="heatmap" :titleSettings='titleSettings'
    :xAxis='xAxis' :yAxis='yAxis' :dataSource='dataSource'
    :paletteSettings='paletteSettings' :cellSettings='cellSettings'
    :legendSettings='legendSettings'></ejs-heatmap>
  </div>
</template>
<script>
import Vue from 'vue';
import { HeatMapPlugin, Tooltip, Legend } from '@syncfusion/ej2-vue-
heatmap';
Vue.use(HeatMapPlugin);
export default {
  data: function() {
    return {
      dataSource: [
        [73, 39, 26, 39, 94, 0],
        [93, 58, 53, 38, 26, 68],
        [99, 28, 22, 4, 66, 90],
        [14, 26, 97, 69, 69, 3],

```

```

        [7, 46, 47, 47, 88, 6],
        [41, 55, 73, 23, 3, 79],
        [56, 69, 21, 86, 3, 33],
        [45, 7, 53, 81, 95, 79],
        [60, 77, 74, 68, 88, 51],
        [25, 25, 10, 12, 78, 14],
        [25, 56, 55, 58, 12, 82],
        [74, 33, 88, 23, 86, 59]
    ],
    titleSettings: {
        text: 'Sales Revenue per Employee'
    },
    xAxis: {
        labels: ['Nancy', 'Andrew', 'Janet', 'Margaret', 'Steven',
            'Michael', 'Robert', 'Laura', 'Anne', 'Paul', 'Karin',
'Mario']
    },
    yAxis: {
        labels: ['Mon', 'Tues', 'Wed', 'Thurs', 'Fri', 'Sat']
    },
    cellSettings: {
        showLabel: false,
    },
    paletteSettings: {
        palette: [
            { value: 0, color: '#6EB5D0' },
            { value: 50, color: '#7EDCA2' },
            { value: 100, color: '#DCD57E' },
        ]
    },
    legendSettings: {
        position: 'Right',
        title: {
            text: "1000 US$"
        }
    }
    },
    provide:{
        heatmap:[Tooltip, Legend]
    }
}
</script>
<style>
@import '../node_modules/@syncfusion/ej2-vue-heatmap/styles/material.css';
</style>

```

{% previewsample "page.domainurl/code-snippet/heatmap-chart/legend-cs9" %}

## Appearance in Vue HeatMap chart component

### Cell customization

You can customize the cell by using the [cellSettings](#) property.

### Border

Change the width, color, and radius of the heat map cells by using the [border](#) property.

**APP.VUE**

```

<template>
  <div id="app">
    <ejs-heatmap id="heatmap" :titleSettings='titleSettings'
:cellSettings='cellSettings' :xAxis='xAxis' :yAxis='yAxis'
:dataSource='dataSource'></ejs-heatmap>
  </div>
</template>
<script>
import Vue from 'vue';
import { HeatMapPlugin, Tooltip, Legend } from '@syncfusion/ej2-vue-heatmap';
Vue.use(HeatMapPlugin);
export default {
  data: function() {
    return {
      dataSource: [
        [73, 39, 26, 39, 94, 0],
        [93, 58, 53, 38, 26, 68],
        [99, 28, 22, 4, 66, 90],
        [14, 26, 97, 69, 69, 3],
        [7, 46, 47, 47, 88, 6],
        [41, 55, 73, 23, 3, 79],
        [56, 69, 21, 86, 3, 33],
        [45, 7, 53, 81, 95, 79],
        [60, 77, 74, 68, 88, 51],
        [25, 25, 10, 12, 78, 14],
        [25, 56, 55, 58, 12, 82],
        [74, 33, 88, 23, 86, 59]
      ],
      titleSettings: {
        text: 'Sales Revenue per Employee (in 1000 US$)',
        textStyle: {
          size: '15px',
          fontWeight: '500',
          fontStyle: 'Normal',
          fontFamily: 'Segoe UI'
        }
      },
      xAxis: {
        labels: ['Nancy', 'Andrew', 'Janet', 'Margaret', 'Steven',
'Michael', 'Robert', 'Laura', 'Anne', 'Paul', 'Karin',
'Mario']
      },
      yAxis: {
        labels: ['Mon', 'Tues', 'Wed', 'Thurs', 'Fri', 'Sat']
      },
      cellSettings: {
        border: {
          width: 1,
          radius: 4,
          color: 'white'
        }
      }
    }
  },
},

```

```

    provide:{
      heatmap:[Tooltip, Legend]
    }
  }
</script>
<style>
@import
'https://ej2.syncfusion.com/vue/documentation/node_modules/@syncfusion/ej2-
vue-heatmap/styles/material.css';
</style>

```

{% previewsample "page.domainurl/code-snippet/heatmap-chart/appearance-cs1" %}

### Cell highlighting

Enable or disable the cell highlighting while hovering over the heat map cells by using the [enableCellHighlighting](#) property.

Note: The cell highlighting only works in a SVG rendering mode.

### APP.VUE

```

<template>
  <div id="app">
    <ejs-heatmap id="heatmap" :titleSettings='titleSettings'
    :cellSettings='cellSettings' :xAxis='xAxis' :yAxis='yAxis'
    :dataSource='dataSource'></ejs-heatmap>
  </div>
</template>
<script>
import Vue from 'vue';
import { HeatMapPlugin, Tooltip, Legend } from '@syncfusion/ej2-vue-
heatmap';
Vue.use(HeatMapPlugin);
export default {
  data: function() {
    return {
      dataSource: [
        [73, 39, 26, 39, 94, 0],
        [93, 58, 53, 38, 26, 68],
        [99, 28, 22, 4, 66, 90],
        [14, 26, 97, 69, 69, 3],
        [7, 46, 47, 47, 88, 6],
        [41, 55, 73, 23, 3, 79],
        [56, 69, 21, 86, 3, 33],
        [45, 7, 53, 81, 95, 79],
        [60, 77, 74, 68, 88, 51],
        [25, 25, 10, 12, 78, 14],
        [25, 56, 55, 58, 12, 82],
        [74, 33, 88, 23, 86, 59]
      ],
      titleSettings: {
        text: 'Sales Revenue per Employee (in 1000 US$)',
        textStyle: {
          size: '15px',
          fontWeight: '500',
          fontStyle: 'Normal',

```

```

        fontFamily: 'Segoe UI'
      },
    },
    cellSettings: {
      enableCellHighlighting: true
    },
    xAxis: {
      labels: ['Nancy', 'Andrew', 'Janet', 'Margaret', 'Steven',
        'Michael', 'Robert', 'Laura', 'Anne', 'Paul', 'Karin',
'Mario']
    },
    yAxis: {
      labels: ['Mon', 'Tues', 'Wed', 'Thurs', 'Fri', 'Sat']
    }
  },
  provide: {
    heatmap: [Tooltip, Legend]
  }
}
</script>
<style>
@import
'https://ej2.syncfusion.com/vue/documentation/node_modules/@syncfusion/ej2-
vue-heatmap/styles/material.css';
</style>

```

{% previewsample "page.domainurl/code-snippet/heatmap-chart/appearance-cs2" %}

#### Color gradient mode

The [colorGradientMode](#) property can be used to set the minimum and maximum values for colors based on row and column.

Three types of color gradient modes are available.

- **Table:** The minimum and maximum value colors calculated for overall data.
- **Row:** The minimum and maximum value colors calculated for each row of data.
- **Column:** The minimum and maximum value colors calculated for each column of data.

Note: The default value of `colorGradientMode` is **Table**.

#### APP.VUE

```

<template>
  <div id="app">
    <ejs-heatmap id="heatmap" :titleSettings='titleSettings'
:cellSettings='cellSettings' :paletteSettings='paletteSettings'
:xAxis='xAxis' :yAxis='yAxis' :dataSource='dataSource'></ejs-heatmap>
  </div>
</template>
<script>
import Vue from 'vue';
import { HeatMapPlugin, Tooltip, Legend } from '@syncfusion/ej2-vue-
heatmap';
Vue.use(HeatMapPlugin);

```

```

export default {
  data: function() {
    return {
      dataSource: [
        [73, 39, 26, 39, 94, 0],
        [93, 58, 53, 38, 26, 68],
        [99, 28, 22, 4, 66, 90],
        [14, 26, 97, 69, 69, 3],
        [7, 46, 47, 47, 88, 6],
        [41, 55, 73, 23, 3, 79],
        [56, 69, 21, 86, 3, 33],
        [45, 7, 53, 81, 95, 79],
        [60, 77, 74, 68, 88, 51],
        [25, 25, 10, 12, 78, 14],
        [25, 56, 55, 58, 12, 82],
        [74, 33, 88, 23, 86, 59]
      ],
      titleSettings: {
        text: 'Sales Revenue per Employee (in 1000 US$)',
        textStyle: {
          size: '15px',
          fontWeight: '500',
          fontStyle: 'Normal',
          fontFamily: 'Segoe UI'
        }
      },
      cellSettings: {
        enableCellHighlighting: true
      },
      paletteSettings: {
        colorGradientMode: 'Column'
      },
      xAxis: {
        labels: ['Nancy', 'Andrew', 'Janet', 'Margaret', 'Steven',
          'Michael', 'Robert', 'Laura', 'Anne', 'Paul', 'Karin',
          'Mario']
      },
      yAxis: {
        labels: ['Mon', 'Tues', 'Wed', 'Thurs', 'Fri', 'Sat']
      }
    }
  },
  provide: {
    heatmap: [Tooltip, Legend]
  }
}
</script>
<style>
@import
'https://ej2.syncfusion.com/vue/documentation/node_modules/@syncfusion/ej2-
vue-heatmap/styles/material.css';
</style>

```

{% previewsample "page.domainurl/code-snippet/heatmap-chart/appearance-cs3" %}

### Background color

The background color of the HeatMap can be customized using the [backgroundColor](#) property.

#### APP.VUE

```
<template>
  <div id="app">
    <ejs-heatmap
      id="heatmap"
      :backgroundColor="backgroundColor"
      :titleSettings="titleSettings"
      :xAxis="xAxis"
      :yAxis="yAxis"
      :dataSource="dataSource"
    ></ejs-heatmap>
  </div>
</template>
<script>
import Vue from "vue";
import { HeatMapPlugin } from "@syncfusion/ej2-vue-heatmap";
Vue.use(HeatMapPlugin);
export default {
  data: function() {
    return {
      dataSource: [
        [73, 39, 26, 39, 94, 0],
        [93, 58, 53, 38, 26, 68],
        [99, 28, 22, 4, 66, 90],
        [14, 26, 97, 69, 69, 3],
        [7, 46, 47, 47, 88, 6],
        [41, 55, 73, 23, 3, 79],
        [56, 69, 21, 86, 3, 33],
        [45, 7, 53, 81, 95, 79],
        [60, 77, 74, 68, 88, 51],
        [25, 25, 10, 12, 78, 14],
        [25, 56, 55, 58, 12, 82],
        [74, 33, 88, 23, 86, 59]
      ],
      backgroundColor: "#c7afcf",
      titleSettings: {
        text: "Sales Revenue per Employee (in 1000 US$)",
        textStyle: {
          size: "15px",
          fontWeight: "500",
          fontStyle: "Normal",
          fontFamily: "Segoe UI"
        }
      },
      xAxis: {
        labels: [
          "Nancy",
          "Andrew",
          "Janet",
          "Margaret",
          "Steven",
          "Michael",
          "Robert",
        ]
      }
    }
  }
};
```

```

        "Laura",
        "Anne",
        "Paul",
        "Karin",
        "Mario"
    ]
  },
  yAxis: {
    labels: ["Mon", "Tues", "Wed", "Thurs", "Fri", "Sat"]
  }
};
}
};
</script>

```

{% previewsample "page.domainurl/code-snippet/heatmap-chart/appearance-cs4" %}

### Margin

Set the margin for the HeatMap from its container by using the [margin](#) property.

### APP.VUE

```

<template>
  <div id="app">
    <ejs-heatmap id="heatmap" :titleSettings='titleSettings'
    :margin='margin' :xAxis='xAxis' :yAxis='yAxis'
    :dataSource='dataSource'></ejs-heatmap>
  </div>
</template>
<script>
import Vue from 'vue';
import { HeatMapPlugin, Tooltip, Legend } from '@syncfusion/ej2-vue-heatmap';
Vue.use(HeatMapPlugin);
export default {
  data: function() {
    return {
      dataSource: [
        [73, 39, 26, 39, 94, 0],
        [93, 58, 53, 38, 26, 68],
        [99, 28, 22, 4, 66, 90],
        [14, 26, 97, 69, 69, 3],
        [7, 46, 47, 47, 88, 6],
        [41, 55, 73, 23, 3, 79],
        [56, 69, 21, 86, 3, 33],
        [45, 7, 53, 81, 95, 79],
        [60, 77, 74, 68, 88, 51],
        [25, 25, 10, 12, 78, 14],
        [25, 56, 55, 58, 12, 82],
        [74, 33, 88, 23, 86, 59]
      ],
      titleSettings: {
        text: 'Sales Revenue per Employee (in 1000 US$)',
        textStyle: {
          size: '15px',
          fontWeight: '500',

```



```

        fontStyle: 'Normal',
        fontFamily: 'Segoe UI'
    },
    },
    xAxis: {
        labels: ['Nancy', 'Andrew', 'Janet', 'Margaret', 'Steven',
        'Michael', 'Robert', 'Laura', 'Anne', 'Paul', 'Karin',
'Mario']
    },
    yAxis: {
        labels: ['Mon', 'Tues', 'Wed', 'Thurs', 'Fri', 'Sat']
    },
    margin: { left: 15, right: 15, top: 15, bottom: 15 }
    },
    },
    provide:{
        heatmap:[Tooltip, Legend]
    }
    }
</script>
<style>
@import
'https://ej2.syncfusion.com/vue/documentation/node_modules/@syncfusion/ej2-
vue-heatmap/styles/material.css';
</style>

```

{% previewsample "page.domainurl/code-snippet/heatmap-chart/appearance-cs5" %}

### Title

The title is used to provide a quick information about the data plotted in heatmap. The [text](#) property is used to set the title for the heatmap. The text style of the title can be customized by using the [textStyle](#) property.

### APP.VUE

```

<template>
    <div id="app">
        <ejs-heatmap id="heatmap" :titleSettings='titleSettings'
        :xAxis='xAxis' :yAxis='yAxis' :dataSource='dataSource'></ejs-heatmap>
    </div>
</template>
<script>
import Vue from 'vue';
import { HeatMapPlugin, Tooltip, Legend } from '@syncfusion/ej2-vue-
heatmap';
Vue.use(HeatMapPlugin);
export default {
    data: function() {
        return {
            dataSource: [
                [73, 39, 26, 39, 94, 0],
                [93, 58, 53, 38, 26, 68],
                [99, 28, 22, 4, 66, 90],
                [14, 26, 97, 69, 69, 3],
                [7, 46, 47, 47, 88, 6],
                [41, 55, 73, 23, 3, 79],
            ]
        }
    }
}

```

```

        [56, 69, 21, 86, 3, 33],
        [45, 7, 53, 81, 95, 79],
        [60, 77, 74, 68, 88, 51],
        [25, 25, 10, 12, 78, 14],
        [25, 56, 55, 58, 12, 82],
        [74, 33, 88, 23, 86, 59]
    ],
    titleSettings: {
        text: 'Sales Revenue per Employee (in 1000 US$)',
        textStyle: {
            size: '15px',
            fontWeight: '500',
            fontStyle: 'Italic',
            fontFamily: 'Segoe UI'
        }
    },
    xAxis: {
        labels: ['Nancy', 'Andrew', 'Janet', 'Margaret', 'Steven',
            'Michael', 'Robert', 'Laura', 'Anne', 'Paul', 'Karin',
            'Mario']
    },
    yAxis: {
        labels: ['Mon', 'Tues', 'Wed', 'Thurs', 'Fri', 'Sat']
    }
},
provide: {
    heatmap: [Tooltip, Legend]
}
</script>
<style>
@import
'https://ej2.syncfusion.com/vue/documentation/node_modules/@syncfusion/ej2-
vue-heatmap/styles/material.css';
</style>

```

{% previewsample "page.domainurl/code-snippet/heatmap-chart/appearance-cs6" %}

### Data label

The visibility of data labels can be toggled using the [showLabel](#) property. By default, the data labels will be visible.

### APP.VUE

```

<template>
    <div id="app">
        <ejs-heatmap id="heatmap" :titleSettings='titleSettings'
        :xAxis='xAxis' :yAxis='yAxis' :dataSource='dataSource'
        :cellSettings='cellSettings'></ejs-heatmap>
    </div>
</template>
<script>
import Vue from 'vue';
import { HeatMapPlugin, Tooltip, Legend } from '@syncfusion/ej2-vue-
heatmap';

```

```

Vue.use(HeatMapPlugin);
export default {
  data: function() {
    return {
      dataSource: [
        [73, 39, 26, 39, 94, 0],
        [93, 58, 53, 38, 26, 68],
        [99, 28, 22, 4, 66, 90],
        [14, 26, 97, 69, 69, 3],
        [7, 46, 47, 47, 88, 6],
        [41, 55, 73, 23, 3, 79],
        [56, 69, 21, 86, 3, 33],
        [45, 7, 53, 81, 95, 79],
        [60, 77, 74, 68, 88, 51],
        [25, 25, 10, 12, 78, 14],
        [25, 56, 55, 58, 12, 82],
        [74, 33, 88, 23, 86, 59]
      ],
      titleSettings: {
        text: 'Sales Revenue per Employee (in 1000 US$)',
        textStyle: {
          size: '15px',
          fontWeight: '500',
          fontStyle: 'Normal',
          fontFamily: 'Segoe UI'
        }
      },
      xAxis: {
        labels: ['Nancy', 'Andrew', 'Janet', 'Margaret', 'Steven',
          'Michael', 'Robert', 'Laura', 'Anne', 'Paul', 'Karin',
          'Mario']
      },
      yAxis: {
        labels: ['Mon', 'Tues', 'Wed', 'Thurs', 'Fri', 'Sat']
      },
      cellSettings: {
        showLabel: false
      }
    }
  },
  provide: {
    heatmap: [Tooltip, Legend]
  }
}
</script>
<style>
@import
'https://ej2.syncfusion.com/vue/documentation/node_modules/@syncfusion/ej2-
vue-heatmap/styles/material.css';
</style>

```

{% previewsample "page.domainurl/code-snippet/heatmap-chart/appearance-cs7" %}

#### *Customize the data label*

The label displayed in the HeatMap cell can be changed using the [cellRender](#) event.

**APP.VUE**

```

<template>
  <div id="app">
    <ejs-heatmap
      id="heatmap"
      :titleSettings="titleSettings"
      :xAxis="xAxis"
      :yAxis="yAxis"
      :dataSource="dataSource"
      :cellRender="cellRender"
    ></ejs-heatmap>
  </div>
</template>
<script>
import Vue from "vue";
import { HeatMapPlugin, Tooltip, Legend } from "@syncfusion/ej2-vue-heatmap";
Vue.use(HeatMapPlugin);
export default {
  data: function() {
    return {
      dataSource: [
        [73, 39, 26, 39, 94, 0],
        [93, 58, 53, 38, 26, 68],
        [99, 28, 22, 4, 66, 90],
        [14, 26, 97, 69, 69, 3],
        [7, 46, 47, 47, 88, 6],
        [41, 55, 73, 23, 3, 79],
        [56, 69, 21, 86, 3, 33],
        [45, 7, 53, 81, 95, 79],
        [60, 77, 74, 68, 88, 51],
        [25, 25, 10, 12, 78, 14],
        [25, 56, 55, 58, 12, 82],
        [74, 33, 88, 23, 86, 59]
      ],
      titleSettings: {
        text: "Sales Revenue per Employee (in 1000 US$)",
        textStyle: {
          size: "15px",
          fontWeight: "500",
          fontStyle: "Normal",
          fontFamily: "Segoe UI"
        }
      },
      xAxis: {
        labels: [
          "Nancy",
          "Andrew",
          "Janet",
          "Margaret",
          "Steven",
          "Michael",
          "Robert",
          "Laura",
          "Anne",
          "Paul",
        ]
      }
    }
  }
}

```

```

        "Karin",
        "Mario"
    ],
    },
    yAxis: {
        labels: ["Mon", "Tues", "Wed", "Thurs", "Fri", "Sat"]
    }
    };
},
provide: {
    heatmap: [Tooltip, Legend]
},
methods: {
    cellRender: function(args) {
        args.displayText = args.value + "$";
    }
}
};
</script>

```

{% previewsample "page.domainurl/code-snippet/heatmap-chart/appearance-cs8" %}

#### Text style

The text attributes of the data label such as font-family, font-size, and color can be customized using the [textStyle](#) in the [cellSettings](#) property.

#### APP.VUE

```

<template>
    <div id="app">
        <ejs-heatmap id="heatmap" :titleSettings='titleSettings'
        :xAxis='xAxis' :yAxis='yAxis' :dataSource='dataSource'
        :cellSettings='cellSettings'></ejs-heatmap>
    </div>
</template>
<script>
import Vue from 'vue';
import { HeatMapPlugin, Tooltip, Legend } from '@syncfusion/ej2-vue-heatmap';
Vue.use(HeatMapPlugin);
export default {
    data: function() {
        return {
            dataSource: [
                [73, 39, 26, 39, 94, 0],
                [93, 58, 53, 38, 26, 68],
                [99, 28, 22, 4, 66, 90],
                [14, 26, 97, 69, 69, 3],
                [7, 46, 47, 47, 88, 6],
                [41, 55, 73, 23, 3, 79],
                [56, 69, 21, 86, 3, 33],
                [45, 7, 53, 81, 95, 79],
                [60, 77, 74, 68, 88, 51],
                [25, 25, 10, 12, 78, 14],
                [25, 56, 55, 58, 12, 82],
                [74, 33, 88, 23, 86, 59]
            ]
        }
    }
}

```

```

    ],
    titleSettings: {
      text: 'Sales Revenue per Employee (in 1000 US$)',
      textStyle: {
        size: '15px',
        fontWeight: '500',
        fontStyle: 'Normal',
        fontFamily: 'Segoe UI'
      }
    },
    xAxis: {
      labels: ['Nancy', 'Andrew', 'Janet', 'Margaret', 'Steven',
        'Michael', 'Robert', 'Laura', 'Anne', 'Paul', 'Karin',
'Mario']
    },
    yAxis: {
      labels: ['Mon', 'Tues', 'Wed', 'Thurs', 'Fri', 'Sat']
    },
    cellSettings: {
      textStyle: {
        fontStyle: 'Italic',
        fontFamily: 'Segoe UI'
      }
    }
  }
},
provide: {
  heatmap: [Tooltip, Legend]
}
</script>
<style>
@import
'https://ej2.syncfusion.com/vue/documentation/node_modules/@syncfusion/ej2-
vue-heatmap/styles/material.css';
</style>

```

{% previewsample "page.domainurl/code-snippet/heatmap-chart/appearance-cs9" %}

#### Format

The format of the data label, such as currency, decimal, percent etc. can be changed using [format](#) property.

#### APP.VUE

```

<template>
  <div id="app">
    <ejs-heatmap id="heatmap" :titleSettings='titleSettings'
:xAxIs='xAxis' :yAxis='yAxis' :dataSource='dataSource'
:cellSettings='cellSettings'></ejs-heatmap>
  </div>
</template>
<script>
import Vue from 'vue';
import { HeatMapPlugin, Tooltip, Legend } from '@syncfusion/ej2-vue-
heatmap';

```

```

Vue.use(HeatMapPlugin);
export default {
  data: function() {
    return {
      dataSource: [
        [73, 39, 26, 39, 94, 0],
        [93, 58, 53, 38, 26, 68],
        [99, 28, 22, 4, 66, 90],
        [14, 26, 97, 69, 69, 3],
        [7, 46, 47, 47, 88, 6],
        [41, 55, 73, 23, 3, 79],
        [56, 69, 21, 86, 3, 33],
        [45, 7, 53, 81, 95, 79],
        [60, 77, 74, 68, 88, 51],
        [25, 25, 10, 12, 78, 14],
        [25, 56, 55, 58, 12, 82],
        [74, 33, 88, 23, 86, 59]
      ],
      titleSettings: {
        text: 'Sales Revenue per Employee (in 1000 US$)',
        textStyle: {
          size: '15px',
          fontWeight: '500',
          fontStyle: 'Normal',
          fontFamily: 'Segoe UI'
        }
      },
      xAxis: {
        labels: ['Nancy', 'Andrew', 'Janet', 'Margaret', 'Steven',
          'Michael', 'Robert', 'Laura', 'Anne', 'Paul', 'Karin',
          'Mario']
      },
      yAxis: {
        labels: ['Mon', 'Tues', 'Wed', 'Thurs', 'Fri', 'Sat']
      },
      cellSettings: {
        format: '${value}'
      }
    }
  },
  provide: {
    heatmap: [Tooltip, Legend]
  }
}
</script>
<style>
@import '../node_modules/@syncfusion/ej2-vue-heatmap/styles/material.css';
</style>

```

{% previewsample "page.domainurl/code-snippet/heatmap-chart/appearance-cs10" %}

### Template

Any HTML elements can be added as a template in the data labels by using the [labelTemplate](#) property of [cellSettings](#) in the HeatMap.

The following examples show various data binding methods in the HeatMap using the `labelTemplate` property.

#### Array binding

By including `${value}` in the template content, the value from the data source for the corresponding cell can be displayed in the HeatMap cell as data label template content. Additionally, the x-axis and y-axis label values can be displayed by including `${xLabel}` and `${yLabel}` in the template content.

#### Table

The following example demonstrates how to add a data label template for array table binding.

#### APP.VUE

```
<template>
  <div id="app">
    <ejs-heatmap id="heatmap" :titleSettings='titleSettings'
      :xAxis='xAxis' :yAxis='yAxis' :dataSource='dataSource'
      :cellSettings='cellSettings'></ejs-heatmap>
  </div>
</template>
<script>
import Vue from 'vue';
import { HeatMapPlugin, Tooltip, Legend } from '@syncfusion/ej2-vue-heatmap';
Vue.use(HeatMapPlugin);
export default {
  data: function () {
    return {
      dataSource: [
        [[4, 39], [3, 8], [1, 3], [1, 10], [4, 4], [2, 15]],
        [[4, 28], [5, 92], [5, 73], [3, 1], [3, 4], [4, 126]],
        [[4, 45], [5, 152], [0, 44], [4, 54], [5, 243], [2, 45]]
      ],
      titleSettings: {
        text: 'Commercial Aviation Accidents and Fatalities by year
2015 - 2017',
        textStyle: {
          size: '15px',
          fontWeight: '500',
          fontStyle: 'Normal',
          fontFamily: 'inherit'
        }
      },
      xAxis: {
        labels: ['2015', '2016', '2017']
      },
      yAxis: {
        labels: ['Jan-Feb', 'Mar-Apr', 'May-Jun', 'Jul-Aug', 'Sep-
Oct', 'Nov-Dec']
      },
      cellSettings: {
        labelTemplate:
          '<div style="width:25px;height:20px;text-align:center;padding-top:2px;background-color:#5BBB9C; border: 1px solid #000000; border-radius:50%;font-weight:bold;">${value}</div>'
      }
    }
  }
}
```



```

    }
  },
  provide: {
    heatmap: [Tooltip, Legend]
  }
}
</script>

```

{% previewsample "page.domainurl/code-snippet/heatmap-chart/appearance-cs11" %}

## Cell

The following example demonstrates how to add a data label template for array cell binding.

### APP.VUE

```

<template>
  <div id="app">
    <ejs-heatmap id="heatmap" :titleSettings='titleSettings'
      :dataSourceSettings='dataSourceSettings' :xAxis='xAxis' :yAxis='yAxis'
      :dataSource='dataSource'
      :cellSettings='cellSettings'></ejs-heatmap>
  </div>
</template>
<script>
import Vue from 'vue';
import { HeatMapPlugin, Adaptor, Tooltip, Legend } from '@syncfusion/ej2-
vue-heatmap';
Vue.use(HeatMapPlugin);
export default {
  data: function () {
    return {
      dataSource: [
        [0, 0, [4, 39]], [0, 1, [3, 8]], [0, 2, [1, 3]], [0, 3, [1,
10]], [0, 4, [4, 4]], [0, 5, [2, 15]],
        [1, 0, [4, 28]], [1, 1, [5, 92]], [1, 2, [5, 73]], [1, 3,
[3, 1]], [1, 4, [3, 4]], [1, 5, [4, 126]],
        [2, 0, [4, 45]], [2, 1, [5, 152]], [2, 2, [0, 44]], [2, 3,
[4, 54]], [2, 4, [5, 243]], [2, 5, [2, 45]]
      ],
      titleSettings: {
        text: 'Commercial Aviation Accidents and Fatalities by year
2015 - 2017',
        textStyle: {
          size: '15px',
          fontWeight: '500',
          fontStyle: 'Normal',
          fontFamily: 'inherit',
        },
      },
      xAxis: {
        labels: ['2015', '2016', '2017']
      },
      yAxis: {
        labels: ['Jan-Feb', 'Mar-Apr', 'May-Jun', 'Jul-Aug', 'Sep-
Oct', 'Nov-Dec']
      },
    },
  },

```

```

        cellSettings: {
            labelTemplate:
                '<div style="width:25px;height:20px;text-align:center;padding-top:2px;background-color:#5BBB9C; border: 1px solid #000000; border-radius:50%;font-weight:bold;">${value}</div>'
        },
        dataSourceSettings: {
            isJsonData: false,
            adaptorType: 'Cell'
        }
    },
    provide: {
        heatmap: [Tooltip, Legend, Adaptor]
    }
}
</script>

```

{% previewsample "page.domainurl/code-snippet/heatmap-chart/appearance-cs12" %}

### JSON binding

By including the desired field name in the template content, such as **\${value}**, the value from the data source for the corresponding cell can be displayed in the HeatMap cell as data label template content.

### Table

The following example demonstrates how to add a data label template for JSON table binding.

### APP.VUE

```

<template>
    <div id="app">
        <ejs-heatmap id="heatmap" :titleSettings='titleSettings'
        :dataSourceSettings='dataSourceSettings' :xAxis='xAxis'
        :yAxis='yAxis' :dataSource='dataSource'
        :cellSettings='cellSettings'></ejs-heatmap>
    </div>
</template>
<script>
import Vue from 'vue';
import { HeatMapPlugin, Adaptor, Tooltip, Legend } from '@syncfusion/ej2-vue-heatmap';
Vue.use(HeatMapPlugin);
export default {
    data: function () {
        return {
            dataSource: [
                {
                    Year: '2017',
                    image:
                        'https://ej2.syncfusion.com/vue/demos/src/circular-gauge/images/golf-ball.png',
                    'Jan-Feb': [4, 39],
                    'Mar-Apr': [3, 8],
                    'May-Jun': [1, 3],
                    'Jul-Aug': [1, 10],
                    'Sep-Oct': [4, 4],
                }
            ]
        }
    }
}

```

```

        'Nov-Dec': [2, 15]
      },
      {
        Year: '2016',
        image:
          'https://ej2.syncfusion.com/vue/demos/src/circular-
          gauge/images/basket-ball.png',
        'Jan-Feb': [4, 28],
        'Mar-Apr': [5, 92],
        'May-Jun': [5, 73],
        'Jul-Aug': [3, 1],
        'Sep-Oct': [3, 4],
        'Nov-Dec': [4, 126]
      },
      {
        Year: '2015',
        image:
          'https://ej2.syncfusion.com/vue/demos/src/circular-
          gauge/images/foot-ball.png',
        'Jan-Feb': [4, 45],
        'Mar-Apr': [5, 152],
        'May-Jun': [0, 44],
        'Jul-Aug': [4, 54],
        'Sep-Oct': [5, 243],
        'Nov-Dec': [2, 45]
      }
    ],
    titleSettings: {
      text: 'Commercial Aviation Accidents and Fatalities by year
2015 - 2017',
      textStyle: {
        size: '15px',
        fontWeight: '500',
        fontStyle: 'Normal',
        fontFamily: 'inherit',
      },
    },
    xAxis: {
      labels: ['2015', '2016', '2017']
    },
    yAxis: {
      labels: ['Jan-Feb', 'Mar-Apr', 'May-Jun', 'Jul-Aug', 'Sep-
Oct', 'Nov-Dec']
    },
    cellSettings: {
      labelTemplate:
        "<div><img style='width:20px;height:20px;'
src='${image}' /> </div>"
    },
    dataSourceSettings: {
      isJsonData: true,
      adaptorType: 'Table',
      xDataMapping: 'Year'
    }
  },
  provide: {

```

```

        heatmap: [Tooltip, Legend, Adaptor]
    }
}
</script>

```

{% previewsample "page.domainurl/code-snippet/heatmap-chart/appearance-cs13" %}

## Cell

The following example demonstrates how to add a data label template for JSON cell binding.

### APP.VUE

```

<template>
  <div id="app">
    <ejs-heatmap id="heatmap" :titleSettings='titleSettings'
:dataSourceSettings='dataSourceSettings' :xAxis='xAxis'
      :yAxis='yAxis' :dataSource='dataSource'
:cellSettings='cellSettings'></ejs-heatmap>
  </div>
</template>
<script>
import Vue from 'vue';
import { HeatMapPlugin, Adaptor, Tooltip, Legend } from '@syncfusion/ej2-
vue-heatmap';
Vue.use(HeatMapPlugin);
export default {
  data: function () {
    return {
      dataSource: [
        { Year: '2017', Months: 'Jan-Feb', Accidents: 4, Fatalities:
39 },
        { Year: '2017', Months: 'Mar-Apr', Accidents: 3, Fatalities:
8 },
        { Year: '2017', Months: 'May-Jun', Accidents: 1, Fatalities:
3 },
        { Year: '2017', Months: 'Jul-Aug', Accidents: 1, Fatalities:
10 },
        { Year: '2017', Months: 'Sep-Oct', Accidents: 4, Fatalities:
4 },
        { Year: '2017', Months: 'Nov-Dec', Accidents: 2, Fatalities:
15 },
        { Year: '2016', Months: 'Jan-Feb', Accidents: 4, Fatalities:
28 },
        { Year: '2016', Months: 'Mar-Apr', Accidents: 5, Fatalities:
92 },
        { Year: '2016', Months: 'May-Jun', Accidents: 5, Fatalities:
73 },
        { Year: '2016', Months: 'Jul-Aug', Accidents: 3, Fatalities:
1 },
        { Year: '2016', Months: 'Sep-Oct', Accidents: 3, Fatalities:
4 },
        { Year: '2016', Months: 'Nov-Dec', Accidents: 4, Fatalities:
126 },
        { Year: '2015', Months: 'Jan-Feb', Accidents: 4, Fatalities:
45 },

```

```

    { Year: '2015', Months: 'Mar-Apr', Accidents: 5, Fatalities:
152 },
    { Year: '2015', Months: 'May-Jun', Accidents: 0, Fatalities:
0 },
    { Year: '2015', Months: 'Jul-Aug', Accidents: 4, Fatalities:
54 },
    { Year: '2015', Months: 'Sep-Oct', Accidents: 5, Fatalities:
243 },
    { Year: '2015', Months: 'Nov-Dec', Accidents: 2, Fatalities:
45 },
  ],
  titleSettings: {
    text: 'Commercial Aviation Accidents and Fatalities by year
2015 - 2017',
    textStyle: {
      size: '15px',
      fontWeight: '500',
      fontStyle: 'Normal',
      fontFamily: 'inherit',
    },
  },
  xAxis: {
    labels: ['2015', '2016', '2017']
  },
  yAxis: {
    labels: ['Jan-Feb', 'Mar-Apr', 'May-Jun', 'Jul-Aug', 'Sep-
Oct', 'Nov-Dec']
  },
  cellSettings: {
    labelTemplate:
      "<div> Accidents - ${Accidents}</div>"
  },
  dataSourceSettings: {
    isJsonData: true,
    adaptorType: 'Cell',
    xDataMapping: 'Year',
    yDataMapping: 'Months',
    valueMapping: 'Fatalities',
  }
}
},
provide: {
  heatmap: [Tooltip, Legend, Adaptor]
}
}
</script>

```

{% previewsample "page.domainurl/code-snippet/heatmap-chart/appearance-cs14" %}

See Also

- [To customize the appearance of tool tip](#)

## Dimensions in Vue Heatmap chart component

### Size for container

Heat map can be rendered to its container size. You can set the size through inline or CSS.

```
`javascript
<div id='container'>

<div id='element' style="width:650px; height:350px;"></div>

</div>
`
```

### Size for heat map

You can set the size of heat map directly by using the [width](#) and [height](#) properties.

### In pixel

You can set the size for heat map in a pixel.

## APP.VUE

```
<template>
  <div id="app">
    <ejs-heatmap id="heatmap" :titleSettings='titleSettings'
      :xAxis='xAxis' :yAxis='yAxis' :dataSource='dataSource' height='350px'
      width='650px'></ejs-heatmap>
  </div>
</template>
<script>
import Vue from 'vue';
import { HeatMapPlugin, Tooltip, Legend } from '@syncfusion/ej2-vue-heatmap';
Vue.use(HeatMapPlugin);
export default {
  data: function() {
    return {
      dataSource: [
        [73, 39, 26, 39, 94, 0],
        [93, 58, 53, 38, 26, 68],
        [99, 28, 22, 4, 66, 90],
        [14, 26, 97, 69, 69, 3],
        [7, 46, 47, 47, 88, 6],
        [41, 55, 73, 23, 3, 79],
        [56, 69, 21, 86, 3, 33],
        [45, 7, 53, 81, 95, 79],
        [60, 77, 74, 68, 88, 51],
        [25, 25, 10, 12, 78, 14],
        [25, 56, 55, 58, 12, 82],
        [74, 33, 88, 23, 86, 59]
      ],
      titleSettings: {
        text: 'Sales Revenue per Employee (in 1000 US$)',
        textStyle: {
          size: '15px',
          fontWeight: '500',
          fontStyle: 'Normal',
          fontFamily: 'Segoe UI'
        }
      }
    }
  }
}
```

```

    },
    xAxis: {
      labels: ['Nancy', 'Andrew', 'Janet', 'Margaret', 'Steven',
        'Michael', 'Robert', 'Laura', 'Anne', 'Paul', 'Karin',
        'Mario'],
    },
    yAxis: {
      labels: ['Mon', 'Tues', 'Wed', 'Thurs', 'Fri', 'Sat'],
    }
  },
  provide: {
    heatmap: [Tooltip, Legend]
  }
}
</script>
<style>
@import '../node_modules/@syncfusion/ej2-vue-heatmap/styles/material.css';
</style>

```

{% previewsample "page.domainurl/code-snippet/heatmap-chart/dimensions-cs1" %}

### In percentage

By setting value in percentage, heat map gets its dimension with respect to its container. For example, when the height is '50%', heat map rendered to half of the container height.

### APP.VUE

```

<template>
  <div id="app">
    <ejs-heatmap id="heatmap" :titleSettings='titleSettings'
      :xAxis='xAxis' :yAxis='yAxis' :dataSource='dataSource' height='90%'
      width='80%'></ejs-heatmap>
    </div>
  </template>
<script>
import Vue from 'vue';
import { HeatMapPlugin, Tooltip, Legend } from '@syncfusion/ej2-vue-heatmap';
Vue.use(HeatMapPlugin);
export default {
  data: function() {
    return {
      dataSource: [
        [73, 39, 26, 39, 94, 0],
        [93, 58, 53, 38, 26, 68],
        [99, 28, 22, 4, 66, 90],
        [14, 26, 97, 69, 69, 3],
        [7, 46, 47, 47, 88, 6],
        [41, 55, 73, 23, 3, 79],
        [56, 69, 21, 86, 3, 33],
        [45, 7, 53, 81, 95, 79],
        [60, 77, 74, 68, 88, 51],
        [25, 25, 10, 12, 78, 14],
        [25, 56, 55, 58, 12, 82],
      ]
    }
  }
}

```

```

        [74, 33, 88, 23, 86, 59]
    ],
    titleSettings: {
        text: 'Sales Revenue per Employee (in 1000 US$)',
        textStyle: {
            size: '15px',
            fontWeight: '500',
            fontStyle: 'Normal',
            fontFamily: 'Segoe UI'
        }
    },
    xAxis: {
        labels: ['Nancy', 'Andrew', 'Janet', 'Margaret', 'Steven',
            'Michael', 'Robert', 'Laura', 'Anne', 'Paul', 'Karin',
            'Mario'],
    },
    yAxis: {
        labels: ['Mon', 'Tues', 'Wed', 'Thurs', 'Fri', 'Sat'],
    }
},
provide:{
    heatmap:[Tooltip, Legend]
}
}
</script>
<style>
@import '../node_modules/@syncfusion/ej2-vue-heatmap/styles/material.css';
</style>

```

{% previewsample "page.domainurl/code-snippet/heatmap-chart/dimensions-cs2" %}

### Tooltip in Vue Heatmap chart component

Tooltip is used to provide the details of the heatmap cell, and this can be displayed, while hovering the cursor over the cell or performing tap action in touch devices.

#### Default tooltip

You can enable the tooltip by setting the [showTooltip](#) property to **true** and injecting the **Tooltip** module into the **provide**.

#### APP.VUE

```

<template>
  <div id="app">
    <ejs-heatmap id="heatmap" :titleSettings='titleSettings'
    :xAxis='xAxis' :yAxis='yAxis' :dataSource='dataSource'
    :cellSettings='cellSettings' :showTooltip='showTooltip'></ejs-heatmap>
  </div>
</template>
<script>
import Vue from 'vue';
import { HeatMapPlugin, Tooltip, Legend } from '@syncfusion/ej2-vue-heatmap';
Vue.use(HeatMapPlugin);
export default {
  data: function() {

```



```

    return {
      dataSource: [
        [0.72, 0.71, 0.71, 0.67, 0.72, 0.53, 0.53, 0.56, 0.58, 0.56],
        [2.28, 2.29, 2.09, 1.84, 1.64, 1.49, 1.49, 1.39, 1.32, 1.23],
        [2.02, 2.17, 2.30, 2.39, 2.36, 2.52, 2.62, 2.57, 2.57, 2.74],
        [3.21, 3.26, 3.45, 3.47, 3.42, 3.34, 3.14, 2.83, 2.64, 2.61],
        [3.22, 3.13, 3.04, 2.95, 2.69, 2.49, 2.27, 2.18, 2.06, 1.87],
        [3.30, 3.39, 3.40, 3.48, 3.60, 3.67, 3.73, 3.79, 3.79, 4.07],
        [5.80, 5.74, 5.64, 5.44, 5.18, 5.08, 5.07, 5.00, 5.35, 5.47],
        [6.91, 7.40, 8.13, 8.80, 9.04, 9.24, 9.43, 9.35, 9.49, 9.69]
      ],
      titleSettings: {
        text: 'Crude Oil Production of Non-OPEC Countries (in Million barrels per day)',
        textStyle: {
          size: '15px',
          fontWeight: '500',
          fontStyle: 'Normal',
          fontFamily: 'Segoe UI'
        }
      },
      xAxis: {
        labels: ['Canada', 'China', 'Egypt', 'Mexico', 'Norway', 'Russia', 'UK', 'USA']
      },
      yAxis: {
        labels: ['2000', '2001', '2002', '2003', '2004', '2005', '2006', '2007', '2008', '2009', '2010']
      },
      cellSettings: {
        showLabel: false,
      },
      showTooltip: true
    },
    provide: {
      heatmap: [Tooltip, Legend]
    }
  }
</script>
<style>
@import
'https://ej2.syncfusion.com/vue/documentation/node_modules/@syncfusion/ej2-vue-heatmap/styles/material.css';
</style>

```

{% previewsample "page.domainurl/code-snippet/heatmap-chart/tooltip-cs1" %}

### Tooltip template

In heatmap, you can customize the tooltip using the [tooltipRender](#) client side event.

### APP.VUE

```

<template>
  <div id="app">

```

```

    <ejs-heatmap id="heatmap" :titleSettings='titleSettings'
    :xAxis='xAxis' :yAxis='yAxis' :dataSource='dataSource'
    :cellSettings='cellSettings' :showTooltip='showTooltip'
    :tooltipRender='tooltipRender'></ejs-heatmap>
  </div>
</template>
<script>
import Vue from 'vue';
import { HeatMapPlugin, Tooltip, Legend } from '@syncfusion/ej2-vue-heatmap';
Vue.use(HeatMapPlugin);
export default {
  data: function() {
    return {
      dataSource: [
        [0.72, 0.71, 0.71, 0.67, 0.72, 0.53, 0.53, 0.56, 0.58, 0.56],
        [2.28, 2.29, 2.09, 1.84, 1.64, 1.49, 1.49, 1.39, 1.32, 1.23],
        [2.02, 2.17, 2.30, 2.39, 2.36, 2.52, 2.62, 2.57, 2.57, 2.74],
        [3.21, 3.26, 3.45, 3.47, 3.42, 3.34, 3.14, 2.83, 2.64, 2.61],
        [3.22, 3.13, 3.04, 2.95, 2.69, 2.49, 2.27, 2.18, 2.06, 1.87],
        [3.30, 3.39, 3.40, 3.48, 3.60, 3.67, 3.73, 3.79, 3.79, 4.07],
        [5.80, 5.74, 5.64, 5.44, 5.18, 5.08, 5.07, 5.00, 5.35, 5.47],
        [6.91, 7.40, 8.13, 8.80, 9.04, 9.24, 9.43, 9.35, 9.49, 9.69]
      ],
      titleSettings: {
        text: 'Crude Oil Production of Non-OPEC Countries (in Million
barrels per day)',
        textStyle: {
          size: '15px',
          fontWeight: '500',
          fontStyle: 'Normal',
          fontFamily: 'Segoe UI'
        }
      },
      xAxis: {
        labels: ['Canada', 'China', 'Egypt', 'Mexico', 'Norway',
'Russia', 'UK', 'USA']
      },
      yAxis: {
        labels: ['2000', '2001', '2002', '2003', '2004', '2005', '2006',
'2007', '2008', '2009', '2010']
      },
      cellSettings: {
        showLabel: false,
      },
      showTooltip: true
    }
  },
  provide: {
    heatmap: [Tooltip, Legend]
  },
  methods: {
    tooltipRender: function(args)
    {
      args.content = ['In ' + args.yLabel + ', the ' + args.xLabel + '
produced ' + args.value + ' million barrels per day'];
    }
  }
}

```

```

    }
  }
</script>
<style>
@import
'https://ej2.syncfusion.com/vue/documentation/node_modules/@syncfusion/ej2-
vue-heatmap/styles/material.css';
</style>

```

{% previewsample "page.domainurl/code-snippet/heatmap-chart/tooltip-cs2" %}

Customize the appearance of Tooltip

The [fill](#) and [border](#) properties are used to customize the background color and border of the Tooltip respectively. The [textStyle](#) property in the Tooltip is used to customize the font of the tooltip text.

#### APP.VUE

```

<template>
  <div id="app">
    <ejs-heatmap id="heatmap" :titleSettings='titleSettings'
    :xAxis='xAxis' :yAxis='yAxis' :dataSource='dataSource'
    :cellSettings='cellSettings' :showTooltip='showTooltip'
    :paletteSettings='paletteSettings' :tooltipSettings='tooltipSettings'
    :tooltipRender='tooltipRender'></ejs-heatmap>
  </div>
</template>
<script>
import Vue from 'vue';
import { HeatMapPlugin, Tooltip, Legend } from '@syncfusion/ej2-vue-
heatmap';
Vue.use(HeatMapPlugin);
export default {
  data: function() {
    return {
      dataSource: [
        [0.72, 0.71, 0.71, 0.67, 0.72, 0.53, 0.53, 0.56, 0.58, 0.56],
        [2.28, 2.29, 2.09, 1.84, 1.64, 1.49, 1.49, 1.39, 1.32, 1.23],
        [2.02, 2.17, 2.30, 2.39, 2.36, 2.52, 2.62, 2.57, 2.57, 2.74],
        [3.21, 3.26, 3.45, 3.47, 3.42, 3.34, 3.14, 2.83, 2.64, 2.61],
        [3.22, 3.13, 3.04, 2.95, 2.69, 2.49, 2.27, 2.18, 2.06, 1.87],
        [3.30, 3.39, 3.40, 3.48, 3.60, 3.67, 3.73, 3.79, 3.79, 4.07],
        [5.80, 5.74, 5.64, 5.44, 5.18, 5.08, 5.07, 5.00, 5.35, 5.47],
        [6.91, 7.40, 8.13, 8.80, 9.04, 9.24, 9.43, 9.35, 9.49, 9.69]
      ],
      titleSettings: {
        text: 'Crude Oil Production of Non-OPEC Countries (in Million
        barrels per day)',
        textStyle: {
          size: '15px',
          fontWeight: '500',
          fontStyle: 'Normal',
          fontFamily: 'Segoe UI'
        }
      },
      xAxis: {

```

```

        labels: ['Canada', 'China', 'Egypt', 'Mexico', 'Norway',
        'Russia', 'UK', 'USA']
      },
      yAxis: {
        labels: ['2000', '2001', '2002', '2003', '2004', '2005', '2006',
        '2007', '2008', '2009', '2010']
      },
      cellSettings: {
        showLabel: false,
      },
      paletteSettings: {
        palette: [
          { color: '#F0ADCE' },
          { color: '#19307B' }
        ]
      },
      tooltipSettings: {
        fill: '#696295',
        textStyle: {
          color: 'FFFFFF',
          size: '12px'
        },
        border: {
          width: 2,
          color: '#F0C27B'
        }
      },
      showTooltip: true
    }
  },
  provide: {
    heatmap: [Tooltip, Legend]
  },
  methods: {
    tooltipRender: function(args)
    {
      args.content = ['In ' + args.yLabel + ', the ' + args.xLabel + '
produced ' + args.value + ' million barrels per day'];
    }
  }
}
</script>
<style>
@import '../node_modules/@syncfusion/ej2-vue-heatmap/styles/material.css';
</style>

```

{% previewsample "page.domainurl/code-snippet/heatmap-chart/tooltip-cs3" %}

Note: To use tooltip feature, we need to inject **Tooltip** module into the **provide**.

### Selection in Vue HeatMap chart component

In the HeatMap, the cell selection is used to select single or multiple HeatMap cells at runtime and get the selected cell details using the [cellSelected](#) event. You can enable the cell selection using the [allowSelection](#) property.

The HeatMap cells can be selected using the following interactions, as shown in the table below.

Modes of Interactions	Description
Mouse	HeatMap cells can be selected by clicking or dragging and dropping over them.
Touch	HeatMap cells can be selected by tapping or dragging and dropping over them.
Keyboard	The <b>Ctrl</b> key on the keyboard can be used to enable multiple cell selection with mouse and touch interaction. The <b>Ctrl</b> key can only be used if the <code>enableMultiSelect</code> property is set to <b>true</b> in order to enable multiple cell selection.

**APP.VUE**

```

<template>
  <div id="app">
    <ejs-heatmap id="heatmap" :titleSettings='titleSettings'
    :xAxis='xAxis' :yAxis='yAxis' :dataSource='dataSource'
    :allowSelection='allowSelection'></ejs-heatmap>
  </div>
</template>
<script>
import Vue from 'vue';
import { HeatMapPlugin, Tooltip, Legend } from '@syncfusion/ej2-vue-heatmap';
Vue.use(HeatMapPlugin);
export default {
  data: function() {
    return {
      dataSource: [
        [73, 39, 26, 39, 94, 0],
        [93, 58, 53, 38, 26, 68],
        [99, 28, 22, 4, 66, 90],
        [14, 26, 97, 69, 69, 3],
        [7, 46, 47, 47, 88, 6],
        [41, 55, 73, 23, 3, 79],
        [56, 69, 21, 86, 3, 33],
        [45, 7, 53, 81, 95, 79],
        [60, 77, 74, 68, 88, 51],
        [25, 25, 10, 12, 78, 14],
        [25, 56, 55, 58, 12, 82],
        [74, 33, 88, 23, 86, 59]
      ],
      titleSettings: {
        text: 'Sales Revenue per Employee (in 1000 US$)',
        textStyle: {
          size: '15px',
          fontWeight: '500',
          fontStyle: 'Normal',
          fontFamily: 'Segoe UI'
        }
      },
      xAxis: {
        labels: ['Nancy', 'Andrew', 'Janet', 'Margaret', 'Steven'],

```

```

        'Michael', 'Robert', 'Laura', 'Anne', 'Paul', 'Karin',
'Mario']
    },
    yAxis: {
        labels: ['Mon', 'Tues', 'Wed', 'Thurs', 'Fri', 'Sat']
    },
    allowSelection: true
    }
},
provide:{
    heatmap:[Tooltip, Legend]
},
}
</script>
<style>
@import
'https://ej2.syncfusion.com/vue/documentation/node_modules/@syncfusion/ej2-
vue-heatmap/styles/material.css';
</style>

```

{% previewsample "page.domainurl/code-snippet/heatmap-chart/selection-cs1" %}

### Enable single cell selection

In the HeatMap, the [enableMultiSelect](#) property is used to allow single cell selection. When you set the [enableMultiSelect](#) property to **false**, only one cell is selected. By default, [enableMultiSelect](#) property is set to **true**.

### APP.VUE

```

<template>
  <div id="app">
    <ejs-heatmap id="heatmap" :titleSettings='titleSettings'
:xAxix='xAxix' :yAxis='yAxis' :dataSource='dataSource'
:allowSelection='allowSelection'
:enableMultiSelect='enableMultiSelect'></ejs-heatmap>
  </div>
</template>
<script>
import Vue from 'vue';
import { HeatMapPlugin } from '@syncfusion/ej2-vue-heatmap';
Vue.use(HeatMapPlugin);
export default {
  data: function() {
    return {
      dataSource: [
        [73, 39, 26, 39, 94, 0],
        [93, 58, 53, 38, 26, 68],
        [99, 28, 22, 4, 66, 90],
        [14, 26, 97, 69, 69, 3],
        [7, 46, 47, 47, 88, 6],
        [41, 55, 73, 23, 3, 79],
        [56, 69, 21, 86, 3, 33],
        [45, 7, 53, 81, 95, 79],
        [60, 77, 74, 68, 88, 51],
        [25, 25, 10, 12, 78, 14],
        [25, 56, 55, 58, 12, 82],

```

```

        [74, 33, 88, 23, 86, 59]
    ],
    titleSettings: {
        text: 'Sales Revenue per Employee (in 1000 US$)',
        textStyle: {
            size: '15px',
            fontWeight: '500',
            fontStyle: 'Normal',
            fontFamily: 'Segoe UI'
        }
    },
    xAxis: {
        labels: ['Nancy', 'Andrew', 'Janet', 'Margaret', 'Steven',
            'Michael', 'Robert', 'Laura', 'Anne', 'Paul', 'Karin',
'Mario']
    },
    yAxis: {
        labels: ['Mon', 'Tues', 'Wed', 'Thurs', 'Fri', 'Sat']
    },
    allowSelection: true,
    enableMultiSelect: false
    }
}
}
</script>
<style>
@import '../node_modules/@syncfusion/ej2-vue-heatmap/styles/material.css';
</style>

```

{% previewsample "page.domainurl/code-snippet/heatmap-chart/selection-cs2" %}

#### Clearing cell selection

The [clearSelection](#) method can be used to clear all the selected cells. The below example illustrates the same.

#### APP.VUE

```

<template>
  <div id="app">
    <ejs-heatmap id="heatmap" ref="heatmap"
: titleSettings='titleSettings' :xAxis='xAxis' :yAxis='yAxis'
: dataSource='dataSource' :allowSelection='allowSelection'>
    </ejs-heatmap>
    <button id='selection' v-on:click='clearSelection'>Clear
Selection</button>
  </div>
</template>
<script>
import Vue from 'vue';
import { HeatMapPlugin, Tooltip, Legend } from '@syncfusion/ej2-vue-heatmap';
Vue.use(HeatMapPlugin);
export default {
  data: function() {
    return {
      dataSource: [

```

```

        [73, 39, 26, 39, 94, 0],
        [93, 58, 53, 38, 26, 68],
        [99, 28, 22, 4, 66, 90],
        [14, 26, 97, 69, 69, 3],
        [7, 46, 47, 47, 88, 6],
        [41, 55, 73, 23, 3, 79],
        [56, 69, 21, 86, 3, 33],
        [45, 7, 53, 81, 95, 79],
        [60, 77, 74, 68, 88, 51],
        [25, 25, 10, 12, 78, 14],
        [25, 56, 55, 58, 12, 82],
        [74, 33, 88, 23, 86, 59]
    ],
    titleSettings: {
        text: 'Sales Revenue per Employee (in 1000 US$)',
        textStyle: {
            size: '15px',
            fontWeight: '500',
            fontStyle: 'Normal',
            fontFamily: 'Segoe UI'
        }
    },
    xAxis: {
        labels: ['Nancy', 'Andrew', 'Janet', 'Margaret', 'Steven',
            'Michael', 'Robert', 'Laura', 'Anne', 'Paul', 'Karin',
'Mario']
    },
    yAxis: {
        labels: ['Mon', 'Tues', 'Wed', 'Thurs', 'Fri', 'Sat']
    },
    allowSelection: true
    },
    },
    methods: {
        clearSelection: function (args){
            this.$refs.heatmap.ej2Instances.clearSelection();
        }
    },
    provide:{
        heatmap: [Tooltip, Legend]
    }
}
</script>

```

{% previewsample "page.domainurl/code-snippet/heatmap-chart/selection-cs3" %}

### Appearance in Vue HeatMap chart component

HeatMap has built-in accessibility features like screen reading. Screen reading in the HeatMap component allows all users, regardless of ability or disability, to use the component. The following HeatMap elements will be read aloud with screen reading software like Narrator for Windows.

Elements	Description
---   ---	
Title	Reads the contents of the HeatMap chart's title.



- | Axis labels | Reads the x and y axis labels of the HeatMap chart. |
- | Multilevel labels | Reads the multilevel labels in the x and y axis of the HeatMap chart. |
- | Cell labels | Reads the labels from the cells in the Heatmap chart. |
- | Legend title | Reads the contents of the legend's title as specified in HeatMap chart. |
- | Legend item label | Reads the label of a legend item in HeatMap chart. |

### Ensuring accessibility

The HeatMap component's accessibility levels are ensured through an [accessibility-checker](#) and [axe-core](#) software tools during automated testing.

The accessibility compliance of the HeatMap component is shown in the following sample. Open the [sample](#) in a new window to evaluate the accessibility of the HeatMap component with accessibility tools.

```
{% previewsample "https://ej2.syncfusion.com/accessibility/heat-map.html" %}
```

See also

- [Accessibility in Syncfusion Vue components](#)

### Events in Vue HeatMap chart component

This section describes the HeatMap chart event, which occurs when the required actions are performed.

#### cellClick

When you click on a HeatMap cell, the [cellClick](#) event is triggered. To know more about arguments of this event, refer [here](#).

### APP.VUE

```
<template>
  <div id="app">
    <ejs-heatmap id="heatmap" :titleSettings='titleSettings'
    :xAxis='xAxis' :yAxis='yAxis' :cellClick='cellClick'
    :dataSource='dataSource'></ejs-heatmap>
  </div>
</template>
<script>
import Vue from 'vue';
import { HeatMapPlugin, Tooltip, Legend } from '@syncfusion/ej2-vue-heatmap';
Vue.use(HeatMapPlugin);
export default {
  data: function() {
    return {
      dataSource: [
        [73, 39, 26, 39, 94, 0],
        [93, 58, 53, 38, 26, 68],
        [99, 28, 22, 4, 66, 90],
        [14, 26, 97, 69, 69, 3],
        [7, 46, 47, 47, 88, 6],
        [41, 55, 73, 23, 3, 79],
        [56, 69, 21, 86, 3, 33],
        [45, 7, 53, 81, 95, 79],
      ]
    }
  }
}
```

```

        [60, 77, 74, 68, 88, 51],
        [25, 25, 10, 12, 78, 14],
        [25, 56, 55, 58, 12, 82],
        [74, 33, 88, 23, 86, 59]
    ],
    titleSettings: {
        text: 'Sales Revenue per Employee (in 1000 US$)',
        textStyle: {
            size: '15px',
            fontWeight: '500',
            fontStyle: 'Normal',
            fontFamily: 'Segoe UI'
        }
    },
    xAxis: {
        labels: ['Nancy', 'Andrew', 'Janet', 'Margaret', 'Steven',
            'Michael', 'Robert', 'Laura', 'Anne', 'Paul', 'Karin',
            'Mario']
    },
    yAxis: {
        labels: ['Mon', 'Tues', 'Wed', 'Thurs', 'Fri', 'Sat']
    }
},
methods: {
    cellClick: function (args) {
        console.log("The cell click event has been triggered!!!");
    }
},
provide: {
    heatmap: [Tooltip, Legend]
}
}
</script>

```

{% previewsample "page.domainurl/code-snippet/heatmap-chart/events-cs1" %}

### cellDoubleClick

When you double click on a HeatMap cell, the [cellDoubleClick](#) event is triggered. To know more about arguments of this event, refer [here](#).

### APP.VUE

```

<template>
    <div id="app">
        <ejs-heatmap id="heatmap" :titleSettings='titleSettings'
        :xAxis='xAxis' :yAxis='yAxis' :cellDoubleClick='cellDoubleClick'
        :dataSource='dataSource'></ejs-heatmap>
    </div>
</template>
<script>
import Vue from 'vue';
import { HeatMapPlugin, Tooltip, Legend } from '@syncfusion/ej2-vue-heatmap';
Vue.use(HeatMapPlugin);
export default {

```

```

data: function() {
  return {
    dataSource: [
      [73, 39, 26, 39, 94, 0],
      [93, 58, 53, 38, 26, 68],
      [99, 28, 22, 4, 66, 90],
      [14, 26, 97, 69, 69, 3],
      [7, 46, 47, 47, 88, 6],
      [41, 55, 73, 23, 3, 79],
      [56, 69, 21, 86, 3, 33],
      [45, 7, 53, 81, 95, 79],
      [60, 77, 74, 68, 88, 51],
      [25, 25, 10, 12, 78, 14],
      [25, 56, 55, 58, 12, 82],
      [74, 33, 88, 23, 86, 59]
    ],
    titleSettings: {
      text: 'Sales Revenue per Employee (in 1000 US$)',
      textStyle: {
        size: '15px',
        fontWeight: '500',
        fontStyle: 'Normal',
        fontFamily: 'Segoe UI'
      }
    },
    xAxis: {
      labels: ['Nancy', 'Andrew', 'Janet', 'Margaret', 'Steven',
        'Michael', 'Robert', 'Laura', 'Anne', 'Paul', 'Karin',
        'Mario']
    },
    yAxis: {
      labels: ['Mon', 'Tues', 'Wed', 'Thurs', 'Fri', 'Sat']
    }
  },
  methods: {
    cellDoubleClick: function(args) {
      console.log("The cell double click event has been triggered!!!");
    }
  },
  provide:{
    heatmap:[Tooltip, Legend]
  }
}
</script>

```

{% previewsample "page.domainurl/code-snippet/heatmap-chart/events-cs2" %}

### cellRender

The [cellRender](#) event will be triggered before each HeatMap cell is rendered. To know more about arguments of this event, refer [here](#).

### APP.VUE

```

<template>
  <div id="app">

```

```

    <ejs-heatmap id="heatmap" :titleSettings='titleSettings'
    :xAxis='xAxis' :yAxis='yAxis' :cellRender='cellRender'
    :dataSource='dataSource'></ejs-heatmap>
  </div>
</template>
<script>
import Vue from 'vue';
import { HeatMapPlugin, Tooltip, Legend } from '@syncfusion/ej2-vue-heatmap';
Vue.use(HeatMapPlugin);
export default {
  data: function() {
    return {
      dataSource: [
        [73, 39, 26, 39, 94, 0],
        [93, 58, 53, 38, 26, 68],
        [99, 28, 22, 4, 66, 90],
        [14, 26, 97, 69, 69, 3],
        [7, 46, 47, 47, 88, 6],
        [41, 55, 73, 23, 3, 79],
        [56, 69, 21, 86, 3, 33],
        [45, 7, 53, 81, 95, 79],
        [60, 77, 74, 68, 88, 51],
        [25, 25, 10, 12, 78, 14],
        [25, 56, 55, 58, 12, 82],
        [74, 33, 88, 23, 86, 59]
      ],
      titleSettings: {
        text: 'Sales Revenue per Employee (in 1000 US$)',
        textStyle: {
          size: '15px',
          fontWeight: '500',
          fontStyle: 'Normal',
          fontFamily: 'Segoe UI'
        }
      },
      xAxis: {
        labels: ['Nancy', 'Andrew', 'Janet', 'Margaret', 'Steven',
          'Michael', 'Robert', 'Laura', 'Anne', 'Paul', 'Karin',
          'Mario']
      },
      yAxis: {
        labels: ['Mon', 'Tues', 'Wed', 'Thurs', 'Fri', 'Sat']
      }
    }
  },
  methods: {
    cellRender: function (args) {
      console.log("The cell render event has been triggered!!!");
    }
  },
  provide:{
    heatmap:[Tooltip, Legend]
  }
}
</script>

```

{% previewsample "page.domainurl/code-snippet/heatmap-chart/events-cs3" %}

### cellSelected

When single or multiple cells in the HeatMap are selected, the [cellSelected](#) event is triggered. To know more about arguments of this event, refer [here](#).

### APP.VUE

```
<template>
  <div id="app">
    <ejs-heatmap id="heatmap" :titleSettings='titleSettings'
    :xAxis='xAxis' :yAxis='yAxis' :cellSelected='cellSelected'
    :dataSource='dataSource' :allowSelection='allowSelection'
    ></ejs-heatmap>
  </div>
</template>
<script>
import Vue from 'vue';
import { HeatMapPlugin, Tooltip, Legend } from '@syncfusion/ej2-vue-heatmap';
Vue.use(HeatMapPlugin);
export default {
  data: function() {
    return {
      dataSource: [
        [73, 39, 26, 39, 94, 0],
        [93, 58, 53, 38, 26, 68],
        [99, 28, 22, 4, 66, 90],
        [14, 26, 97, 69, 69, 3],
        [7, 46, 47, 47, 88, 6],
        [41, 55, 73, 23, 3, 79],
        [56, 69, 21, 86, 3, 33],
        [45, 7, 53, 81, 95, 79],
        [60, 77, 74, 68, 88, 51],
        [25, 25, 10, 12, 78, 14],
        [25, 56, 55, 58, 12, 82],
        [74, 33, 88, 23, 86, 59]
      ],
      titleSettings: {
        text: 'Sales Revenue per Employee (in 1000 US$)',
        textStyle: {
          size: '15px',
          fontWeight: '500',
          fontStyle: 'Normal',
          fontFamily: 'Segoe UI'
        }
      },
      xAxis: {
        labels: ['Nancy', 'Andrew', 'Janet', 'Margaret', 'Steven',
        'Michael', 'Robert', 'Laura', 'Anne', 'Paul', 'Karin',
        'Mario']
      },
      yAxis: {
        labels: ['Mon', 'Tues', 'Wed', 'Thurs', 'Fri', 'Sat']
      }
    }
  }
}
```

```

        allowSelection: true
      },
    },
    methods: {
      cellSelected: function(args) {
        console.log("The cell selected event has been triggered!!!");
      }
    },
    provide: {
      heatmap: [Tooltip, Legend]
    }
  }
}
</script>

```

{% previewsample "page.domainurl/code-snippet/heatmap-chart/events-cs4" %}

[created](#)

Once HeatMap has been completely rendered, the [created](#) event is triggered.

### APP.VUE

```

<template>
  <div id="app">
    <ejs-heatmap id="heatmap" :titleSettings='titleSettings'
    :xAxis='xAxis' :yAxis='yAxis' :created='created' :dataSource='dataSource'>
    </ejs-heatmap>
  </div>
</template>
<script>
import Vue from 'vue';
import { HeatMapPlugin, Tooltip, Legend } from '@syncfusion/ej2-vue-heatmap';
Vue.use(HeatMapPlugin);
export default {
  data: function() {
    return {
      dataSource: [
        [73, 39, 26, 39, 94, 0],
        [93, 58, 53, 38, 26, 68],
        [99, 28, 22, 4, 66, 90],
        [14, 26, 97, 69, 69, 3],
        [7, 46, 47, 47, 88, 6],
        [41, 55, 73, 23, 3, 79],
        [56, 69, 21, 86, 3, 33],
        [45, 7, 53, 81, 95, 79],
        [60, 77, 74, 68, 88, 51],
        [25, 25, 10, 12, 78, 14],
        [25, 56, 55, 58, 12, 82],
        [74, 33, 88, 23, 86, 59]
      ],
      titleSettings: {
        text: 'Sales Revenue per Employee (in 1000 US$)',
        textStyle: {
          size: '15px',
          fontWeight: '500',
          fontStyle: 'Normal',

```

```

        fontFamily: 'Segoe UI'
    },
    },
    xAxis: {
        labels: ['Nancy', 'Andrew', 'Janet', 'Margaret', 'Steven',
        'Michael', 'Robert', 'Laura', 'Anne', 'Paul', 'Karin',
'Mario']
    },
    yAxis: {
        labels: ['Mon', 'Tues', 'Wed', 'Thurs', 'Fri', 'Sat']
    }
    },
    },
    methods: {
        created: function (args) {
            console.log("The created event has been triggered!!!");
        }
    },
    provide:{
        heatmap:[Tooltip, Legend]
    }
}
</script>

```

{% previewsample "page.domainurl/code-snippet/heatmap-chart/events-cs5" %}

### legendRender

The [legendRender](#) event is triggered before the legend is rendered. To know more about arguments of this event, refer [here](#).

### APP.VUE

```

<template>
    <div id="app">
        <ejs-heatmap id="heatmap" :titleSettings='titleSettings'
        :xAxis='xAxis' :yAxis='yAxis' :legendRender='legendRender'
        :dataSource='dataSource'></ejs-heatmap>
    </div>
</template>
<script>
import Vue from 'vue';
import { HeatMapPlugin, Tooltip, Legend } from '@syncfusion/ej2-vue-heatmap';
Vue.use(HeatMapPlugin);
export default {
    data: function() {
        return {
            dataSource: [
                [73, 39, 26, 39, 94, 0],
                [93, 58, 53, 38, 26, 68],
                [99, 28, 22, 4, 66, 90],
                [14, 26, 97, 69, 69, 3],
                [7, 46, 47, 47, 88, 6],
                [41, 55, 73, 23, 3, 79],
                [56, 69, 21, 86, 3, 33],
                [45, 7, 53, 81, 95, 79],
            ]
        }
    }
}

```

```

        [60, 77, 74, 68, 88, 51],
        [25, 25, 10, 12, 78, 14],
        [25, 56, 55, 58, 12, 82],
        [74, 33, 88, 23, 86, 59]
    ],
    titleSettings: {
        text: 'Sales Revenue per Employee (in 1000 US$)',
        textStyle: {
            size: '15px',
            fontWeight: '500',
            fontStyle: 'Normal',
            fontFamily: 'Segoe UI'
        }
    },
    xAxis: {
        labels: ['Nancy', 'Andrew', 'Janet', 'Margaret', 'Steven',
            'Michael', 'Robert', 'Laura', 'Anne', 'Paul', 'Karin',
            'Mario']
    },
    yAxis: {
        labels: ['Mon', 'Tues', 'Wed', 'Thurs', 'Fri', 'Sat']
    }
},
methods: {
    legendRender: function(args) {
        console.log("The legend render event has been triggered!!!");
    }
},
provide:{
    heatmap:[Tooltip, Legend]
}
}
</script>

```

{% previewsample "page.domainurl/code-snippet/heatmap-chart/events-cs6" %}

### load

The [load](#) event is triggered before the HeatMap is rendered. To know more about arguments of this event, refer [here](#).

### APP.VUE

```

<template>
    <div id="app">
        <ejs-heatmap id="heatmap" :titleSettings='titleSettings'
        :xAxis='xAxis' :yAxis='yAxis' :load='load' :dataSource='dataSource'></ejs-
        heatmap>
    </div>
</template>
<script>
import Vue from 'vue';
import { HeatMapPlugin, Tooltip, Legend } from '@syncfusion/ej2-vue-
heatmap';
Vue.use(HeatMapPlugin);
export default {

```



```

data: function() {
  return {
    dataSource: [
      [73, 39, 26, 39, 94, 0],
      [93, 58, 53, 38, 26, 68],
      [99, 28, 22, 4, 66, 90],
      [14, 26, 97, 69, 69, 3],
      [7, 46, 47, 47, 88, 6],
      [41, 55, 73, 23, 3, 79],
      [56, 69, 21, 86, 3, 33],
      [45, 7, 53, 81, 95, 79],
      [60, 77, 74, 68, 88, 51],
      [25, 25, 10, 12, 78, 14],
      [25, 56, 55, 58, 12, 82],
      [74, 33, 88, 23, 86, 59]
    ],
    titleSettings: {
      text: 'Sales Revenue per Employee (in 1000 US$)',
      textStyle: {
        size: '15px',
        fontWeight: '500',
        fontStyle: 'Normal',
        fontFamily: 'Segoe UI'
      }
    },
    xAxis: {
      labels: ['Nancy', 'Andrew', 'Janet', 'Margaret', 'Steven',
        'Michael', 'Robert', 'Laura', 'Anne', 'Paul', 'Karin',
        'Mario']
    },
    yAxis: {
      labels: ['Mon', 'Tues', 'Wed', 'Thurs', 'Fri', 'Sat']
    }
  },
  methods: {
    load: function (args) {
      console.log("The load event has been triggered!!!");
    }
  },
  provide:{
    heatmap:[Tooltip, Legend]
  }
}
</script>

```

{% previewsample "page.domainurl/code-snippet/heatmap-chart/events-cs7" %}

#### loaded

Once HeatMap is loaded, the [loaded](#) event is triggered. To know more about arguments of this event, refer [here](#).

#### APP.VUE

```

<template>
  <div id="app">

```

```

    <ejs-heatmap id="heatmap" :titleSettings='titleSettings'
    :xAxis='xAxis' :yAxis='yAxis' :loaded='loaded'
    :dataSource='dataSource'></ejs-heatmap>
  </div>
</template>
<script>
import Vue from 'vue';
import { HeatMapPlugin, Tooltip, Legend } from '@syncfusion/ej2-vue-heatmap';
Vue.use(HeatMapPlugin);
export default {
  data: function() {
    return {
      dataSource: [
        [73, 39, 26, 39, 94, 0],
        [93, 58, 53, 38, 26, 68],
        [99, 28, 22, 4, 66, 90],
        [14, 26, 97, 69, 69, 3],
        [7, 46, 47, 47, 88, 6],
        [41, 55, 73, 23, 3, 79],
        [56, 69, 21, 86, 3, 33],
        [45, 7, 53, 81, 95, 79],
        [60, 77, 74, 68, 88, 51],
        [25, 25, 10, 12, 78, 14],
        [25, 56, 55, 58, 12, 82],
        [74, 33, 88, 23, 86, 59]
      ],
      titleSettings: {
        text: 'Sales Revenue per Employee (in 1000 US$)',
        textStyle: {
          size: '15px',
          fontWeight: '500',
          fontStyle: 'Normal',
          fontFamily: 'Segoe UI'
        }
      },
      xAxis: {
        labels: ['Nancy', 'Andrew', 'Janet', 'Margaret', 'Steven',
          'Michael', 'Robert', 'Laura', 'Anne', 'Paul', 'Karin',
          'Mario']
      },
      yAxis: {
        labels: ['Mon', 'Tues', 'Wed', 'Thurs', 'Fri', 'Sat']
      }
    }
  },
  methods: {
    loaded: function (args) {
      console.log("The loaded event has been triggered!!!");
    }
  },
  provide:{
    heatmap: [Tooltip, Legend]
  }
}
</script>

```

{% previewsample "page.domainurl/code-snippet/heatmap-chart/events-cs8" %}

### resized

When the window is resized, the [resized](#) event is triggered to notify the resize of the HeatMap. To know more about arguments of this event, refer [here](#).

### APP.VUE

```
<template>
  <div id="app">
    <ejs-heatmap id="heatmap" :titleSettings='titleSettings'
      :xAxis='xAxis' :yAxis='yAxis' :resized='resized'
      :dataSource='dataSource'></ejs-heatmap>
    </div>
  </template>
<script>
import Vue from 'vue';
import { HeatMapPlugin, Tooltip, Legend } from '@syncfusion/ej2-vue-heatmap';
Vue.use(HeatMapPlugin);
export default {
  data: function() {
    return {
      dataSource: [
        [73, 39, 26, 39, 94, 0],
        [93, 58, 53, 38, 26, 68],
        [99, 28, 22, 4, 66, 90],
        [14, 26, 97, 69, 69, 3],
        [7, 46, 47, 47, 88, 6],
        [41, 55, 73, 23, 3, 79],
        [56, 69, 21, 86, 3, 33],
        [45, 7, 53, 81, 95, 79],
        [60, 77, 74, 68, 88, 51],
        [25, 25, 10, 12, 78, 14],
        [25, 56, 55, 58, 12, 82],
        [74, 33, 88, 23, 86, 59]
      ],
      titleSettings: {
        text: 'Sales Revenue per Employee (in 1000 US$)',
        textStyle: {
          size: '15px',
          fontWeight: '500',
          fontStyle: 'Normal',
          fontFamily: 'Segoe UI'
        }
      },
      xAxis: {
        labels: ['Nancy', 'Andrew', 'Janet', 'Margaret', 'Steven',
          'Michael', 'Robert', 'Laura', 'Anne', 'Paul', 'Karin',
          'Mario']
      },
      yAxis: {
        labels: ['Mon', 'Tues', 'Wed', 'Thurs', 'Fri', 'Sat']
      }
    }
  }
}
```

```

    },
    methods: {
      resized: function (args) {
        console.log("The resized event has been triggered!!!");
      }
    },
    provide: {
      heatmap: [Tooltip, Legend]
    }
  }
</script>

```

{% previewsample "page.domainurl/code-snippet/heatmap-chart/events-cs9" %}

### tooltipRender

The [tooltipRender](#) event is triggered before the tooltip is rendered on the HeatMap cell. To know more about arguments of this event, refer [here](#).

### APP.VUE

```

<template>
  <div id="app">
    <ejs-heatmap id="heatmap" :titleSettings='titleSettings'
    :xAxis='xAxis' :yAxis='yAxis' :tooltipRender='tooltipRender'
    :dataSource='dataSource'></ejs-heatmap>
  </div>
</template>
<script>
import Vue from 'vue';
import { HeatMapPlugin, Tooltip, Legend } from '@syncfusion/ej2-vue-heatmap';
Vue.use(HeatMapPlugin);
export default {
  data: function() {
    return {
      dataSource: [
        [73, 39, 26, 39, 94, 0],
        [93, 58, 53, 38, 26, 68],
        [99, 28, 22, 4, 66, 90],
        [14, 26, 97, 69, 69, 3],
        [7, 46, 47, 47, 88, 6],
        [41, 55, 73, 23, 3, 79],
        [56, 69, 21, 86, 3, 33],
        [45, 7, 53, 81, 95, 79],
        [60, 77, 74, 68, 88, 51],
        [25, 25, 10, 12, 78, 14],
        [25, 56, 55, 58, 12, 82],
        [74, 33, 88, 23, 86, 59]
      ],
      titleSettings: {
        text: 'Sales Revenue per Employee (in 1000 US$)',
        textStyle: {
          size: '15px',
          fontWeight: '500',
          fontStyle: 'Normal',
          fontFamily: 'Segoe UI'
        }
      }
    }
  }
}

```

```

    }
  },
  xAxis: {
    labels: ['Nancy', 'Andrew', 'Janet', 'Margaret', 'Steven',
      'Michael', 'Robert', 'Laura', 'Anne', 'Paul', 'Karin',
'Mario']
  },
  yAxis: {
    labels: ['Mon', 'Tues', 'Wed', 'Thurs', 'Fri', 'Sat']
  }
}
},
methods: {
  tooltipRender: function (args) {
    console.log("The tooltip render event has been triggered!!!");
  }
},
provide: {
  heatmap: [Tooltip, Legend]
}
}
</script>

```

{% previewsample "page.domainurl/code-snippet/heatmap-chart/events-cs10" %}

## How To

### Tooltip template in Vue Heatmap chart component

You can show a tooltip as a table using the `template` property in `tooltipSettings`.

The following steps describe how to show the table tooltip.

#### Step 1:

Initialize the tooltip template div as shown in the following html page.

#### Step 2:

Set the element id to the `template` property in `tooltipSettings` to show the tooltip template.

#### APP.VUE

```

<template>
  <div id="app">
    <ejs-heatmap id="heatmap" :titleSettings='titleSettings'
:xAxIs='xAxIs' :yAxis='yAxis' :dataSource='dataSource'
:cellSettings='cellSettings' :showTooltip='showTooltip'
:tooltipSettings='tooltipSettings' ></ejs-heatmap>
  </div>
</template>
<script>
import Vue from 'vue';
import { HeatMapPlugin, Tooltip, Legend } from '@syncfusion/ej2-vue-heatmap';
Vue.use(HeatMapPlugin);
let template = '<div>'+
               '<table>'+

```

```

        '<tr bgcolor="#00FFFF"><td
align="right">${xValue}</td><td>${yValue}</td><td >${value}</td></tr>' +
        '</table>'+
        '</div>'
export default {
  data: function() {
    return {
      dataSource: [
        [0.72, 0.71, 0.71, 0.67, 0.72, 0.53, 0.53, 0.56, 0.58, 0.56],
        [2.28, 2.29, 2.09, 1.84, 1.64, 1.49, 1.49, 1.39, 1.32, 1.23],
        [2.02, 2.17, 2.30, 2.39, 2.36, 2.52, 2.62, 2.57, 2.57, 2.74],
        [3.21, 3.26, 3.45, 3.47, 3.42, 3.34, 3.14, 2.83, 2.64, 2.61],
        [3.22, 3.13, 3.04, 2.95, 2.69, 2.49, 2.27, 2.18, 2.06, 1.87],
        [3.30, 3.39, 3.40, 3.48, 3.60, 3.67, 3.73, 3.79, 3.79, 4.07],
        [5.80, 5.74, 5.64, 5.44, 5.18, 5.08, 5.07, 5.00, 5.35, 5.47],
        [6.91, 7.40, 8.13, 8.80, 9.04, 9.24, 9.43, 9.35, 9.49, 9.69]
      ],
      titleSettings: {
        text: 'Crude Oil Production of Non-OPEC Countries (in Million
barrels per day)',
        textStyle: {
          size: '15px',
          fontWeight: '500',
          fontStyle: 'Normal',
          fontFamily: 'Segoe UI'
        }
      },
      xAxis: {
        labels: ['Canada', 'China', 'Egypt', 'Mexico', 'Norway',
'Russia', 'UK', 'USA']
      },
      yAxis: {
        labels: ['2000', '2001', '2002', '2003', '2004', '2005', '2006',
'2007', '2008', '2009', '2010']
      },
      cellSettings: {
        showLabel: false,
      },
      tooltipSettings: {
        fill: '#696295',
        textStyle: {
          color: 'FFFFFF',
          size: '12px'
        },
        border: {
          width: 2,
          color: '#F0C27B'
        },
        template: template
      },
      showTooltip: true
    }
  },
  provide: {
    heatmap: [Tooltip, Legend]
  },
}

```

```

</script>
<style>
  table,
  th,
  td {
    border: 1px solid rgb(105, 105, 105);
    border-collapse: collapse;
  }
</style>
<style>
@import '../node_modules/@syncfusion/ej2-vue-heatmap/styles/material.css';
</style>

```

{% previewsample "page.domainurl/code-snippet/heatmap-chart/how-to-cs2" %}

### Legend label customization in Vue Heatmap chart component

You can change the legend label using the 'legendRender' client-side event. You can also hide the legend label using this client-side event.

#### APP.VUE

```

<template>
  <div id="app">
    <ejs-heatmap id="heatmap" :titleSettings='titleSettings'
    :xAxis='xAxis' :yAxis='yAxis' :dataSource='dataSource'
    :cellSettings='cellSettings' :legendRender='legendRender'
    :paletteSettings='paletteSettings'></ejs-heatmap>
  </div>
</template>
<script>
import Vue from 'vue';
import { HeatMapPlugin, Tooltip, Legend } from '@syncfusion/ej2-vue-heatmap';
Vue.use(HeatMapPlugin);
export default {
  data: function() {
    return {
      dataSource: [
        [73000, 39000, 26000, 39000, 94000, 0],
        [93000, 58000, 53000, 38000, 26000, 68000],
        [99000, 28000, 22000, 4000, 66000, 9000],
        [14000, 26000, 97000, 69000, 69000, 3000],
        [7000, 46000, 47000, 47000, 88000, 6000],
        [41000, 55000, 73000, 23000, 30000, 79000],
        [56000, 69000, 21000, 86000, 3000, 33000],
        [45000, 7000, 53000, 81000, 95000, 79000],
        [60000, 77000, 74000, 68000, 88000, 51000],
        [25000, 25000, 10000, 12000, 78000, 14000],
        [25000, 56000, 55000, 58000, 12000, 82000],
        [74000, 33000, 88000, 23000, 86000, 59000]
      ],
      titleSettings: {
        text: 'Sales Revenue per Employee (in 1000 US$)',
        textStyle: {
          size: '15px',
          fontWeight: '500',

```

```

        fontStyle: 'Normal',
        fontFamily: 'Segoe UI'
    },
    },
    xAxis: {
        labels: ['Nancy', 'Andrew', 'Janet', 'Margaret', 'Steven',
        'Michael', 'Robert', 'Laura', 'Anne', 'Paul', 'Karin',
'Mario'],
    },
    yAxis: {
        labels: ['Mon', 'Tues', 'Wed', 'Thurs', 'Fri', 'Sat']
    },
    cellSettings: {
        showLabel: false,
    },
    paletteSettings: {
        palette: [
            { value: 0, color: '#C2E7EC' },
            { value: 25000, color: '#AEDFE6' },
            { value: 50000, color: '#9AD7E0' },
            { value: 75000, color: '#72C7D4' },
            { value: 99000, color: '#5EBFCE' },
        ],
        type: "Gradient"
    },
    },
    },
    provide:{
        heatmap:[Tooltip, Legend]
    },
    methods :{
        legendRender: function(args)
        {
            if(args.text=='25,000' || args.text=='50,000' ||
args.text=='99,000'){
                args.text = args.text.replace(/,/ , "");
                args.text = ` ${parseInt(args.text/1000)} ` + "k "+"$";
            } else {
                args.cancel=true;
            }
        }
    }
}
}
</script>
<style>
@import '../node_modules/@syncfusion/ej2-vue-heatmap/styles/material.css';
</style>

```

{% previewsample "page.domainurl/code-snippet/heatmap-chart/how-to-cs1" %}



## ImageEditor

### Getting Started with the Vue Image Editor Component in Vue 2

This article provides a step-by-step guide for setting up a Vue 2 project using [Vue-CLI](#) and integrating the Syncfusion Vue Image editor component using the [Composition API](#) / [Options API](#).

#### Dependencies

The following list of dependencies are required to use the Image Editor component in your application.

```
`js
|-- @syncfusion/ej2-vue-image-editor
|-- @syncfusion/ej2-base
|-- @syncfusion/ej2-data
|-- @syncfusion/ej2-buttons
|-- @syncfusion/ej2-inputs
|-- @syncfusion/ej2-navigations
|-- @syncfusion/ej2-popups
|-- @syncfusion/ej2-splitbuttons
|-- @syncfusion/ej2-lists
|-- @syncfusion/ej2-dropdowns
`
```

#### Prerequisites

##### [System requirements for Syncfusion Vue UI components](#)

##### Setting up the Vue 2 project

To generate a Vue 2 project using Vue-CLI, use the [vue create](#) command. Follow these steps to install Vue CLI and create a new project:

```
`bash
npm install -g @vue/cli
vue create quickstart
cd quickstart
npm run serve
`
```

or

```
`bash
yarn global add @vue/cli
vue create quickstart
cd quickstart
yarn run serve
```

When creating a new project, choose the option `Default ([Vue 2] babel, eslint)` from the menu.

```
E:\samples>vue create quickstart

Vue CLI v5.0.8
? Please pick a preset:
  Default ([Vue 3] babel, eslint)
> Default ([Vue 2] babel, eslint)
  Manually select features
```

Once the `quickstart` project is set up with default settings, proceed to add Syncfusion components to the project.

### Add Syncfusion Vue packages

Syncfusion packages are available at [npmjs.com](https://www.npmjs.com). To use Vue components, install the required npm package.

This article uses the [Vue Image Editor component](#) as an example. Install the `@syncfusion/ej2-vue-image-editor` package by running the following command:

```
`bash
```

```
npm install @syncfusion/ej2-vue-image-editor --save
```

or

```
`bash
```

```
yarn add @syncfusion/ej2-vue-image-editor
```

### Import Syncfusion CSS styles

You can import themes for the Syncfusion Vue component in various ways, such as using CSS or SASS styles from npm packages, CDN, [CRG](#) and [Theme Studio](#). Refer to [themes topic](#) to know more about built-in themes and different ways to refer to themes in a Vue project.

In this article, the `Material` theme is applied using CSS styles, which are available in installed packages. The necessary Material CSS styles for the Image Editor component and its dependents were imported into the `<style>` section of `src/App.vue` file.

#### ~/SRC/APP.VUE

```
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-buttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-splitbuttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-lists/styles/material.css";
@import "../node_modules/@syncfusion/ej2-popups/styles/material.css";
@import "../node_modules/@syncfusion/ej2-inputs/styles/material.css";
@import "../node_modules/@syncfusion/ej2-navigations/styles/material.css";
@import "../node_modules/@syncfusion/ej2-dropdowns/styles/material.css";
```

```
@import "../node_modules/@syncfusion/ej2-image-editor/styles/material.css";  
</style>
```

### Add Syncfusion Vue component

Follow the below steps to add the Vue Image Editor component using **Composition API** or **Options API**:

1\ First, import and register the Image Editor component in the **script** section of the **src/App.vue** file. If you are using the **Composition API**, you should add the **setup** attribute to the **script** tag to indicate that Vue will be using the **Composition API**.

#### **COMPOSITION API (~SRC/APP.VUE)**

```
<script setup>  
import { ImageEditorComponent as EjsImageeditor } from "@syncfusion/ej2-vue-image-editor";  
</script>
```

#### **OPTIONS API (~SRC/APP.VUE)**

```
<script>  
import { ImageEditorComponent } from "@syncfusion/ej2-vue-image-editor";  
export default {  
  components: {  
    'ejs-imageeditor': ImageEditorComponent  
  }  
}  
</script>
```

2\ In the **template** section, define the Image Editor component with the [height](#) and [width](#) property.

#### **~/SRC/APP.VUE**

```
<template>  
<div>  
<ejs-imageeditor id="image-editor" height="350px" width="550px"></ejs-imageeditor>  
</div>  
</template>
```

Here is the summarized code for the above steps in the **src/App.vue** file:

#### **COMPOSITION API (~SRC/APP.VUE)**

```
<template>  
<div>  
<ejs-imageeditor id="image-editor" height="350px" width="550px"></ejs-imageeditor>  
</div>  
</template>  
<script setup>  
import { ImageEditorComponent as EjsImageeditor } from "@syncfusion/ej2-vue-image-editor";  
</script>
```

```

<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-buttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-splitbuttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-lists/styles/material.css";
@import "../node_modules/@syncfusion/ej2-popups/styles/material.css";
@import "../node_modules/@syncfusion/ej2-inputs/styles/material.css";
@import "../node_modules/@syncfusion/ej2-navigations/styles/material.css";
@import "../node_modules/@syncfusion/ej2-image-editor/styles/material.css";
#image-editor {
    width: 550px !important;
    height: 350px !important;
}
</style>

```

### OPTIONS API (~SRC/APP.VUE)

```

<template>
<div>
<ejs-imageeditor id="image-editor" height="350px" width="550px"></ejs-
imageeditor>
</div>
</template>
<script>
import { ImageEditorComponent } from "@syncfusion/ej2-vue-image-editor";
export default {
    components: {
        'ejs-imageeditor': ImageEditorComponent
    },
    data: function() {
        return {};
    }
}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-buttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-splitbuttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-lists/styles/material.css";
@import "../node_modules/@syncfusion/ej2-popups/styles/material.css";
@import "../node_modules/@syncfusion/ej2-inputs/styles/material.css";
@import "../node_modules/@syncfusion/ej2-navigations/styles/material.css";
@import "../node_modules/@syncfusion/ej2-dropdowns/styles/material.css";
@import "../node_modules/@syncfusion/ej2-image-editor/styles/material.css";
#image-editor {
    width: 550px !important;
    height: 350px !important;
}
</style>

```

### Run the project

To run the project, use the following command:

```
`bash`
```

```
npm run serve
```

```
,
```

```
or
```

```
`bash
```

```
yarn run serve
```

```
,
```

```
{% previewsample "page.domainurl/code-snippet/image-editor/getting-started-cs6" %}
```

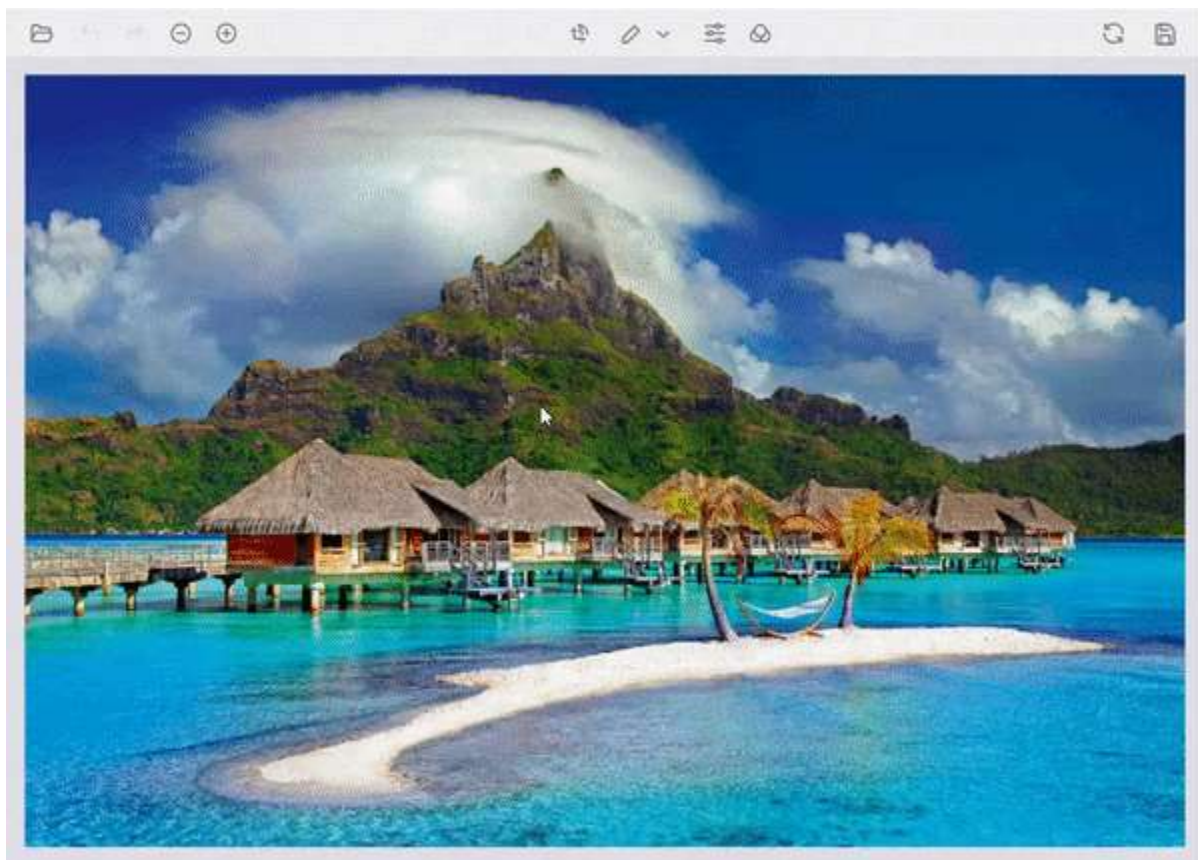
### End-user capabilities in the Image Editor component

The following operations are available for end-users and the same is explained briefly in these sections.

#### Open an image

To open an image in the image editor, do the following steps.

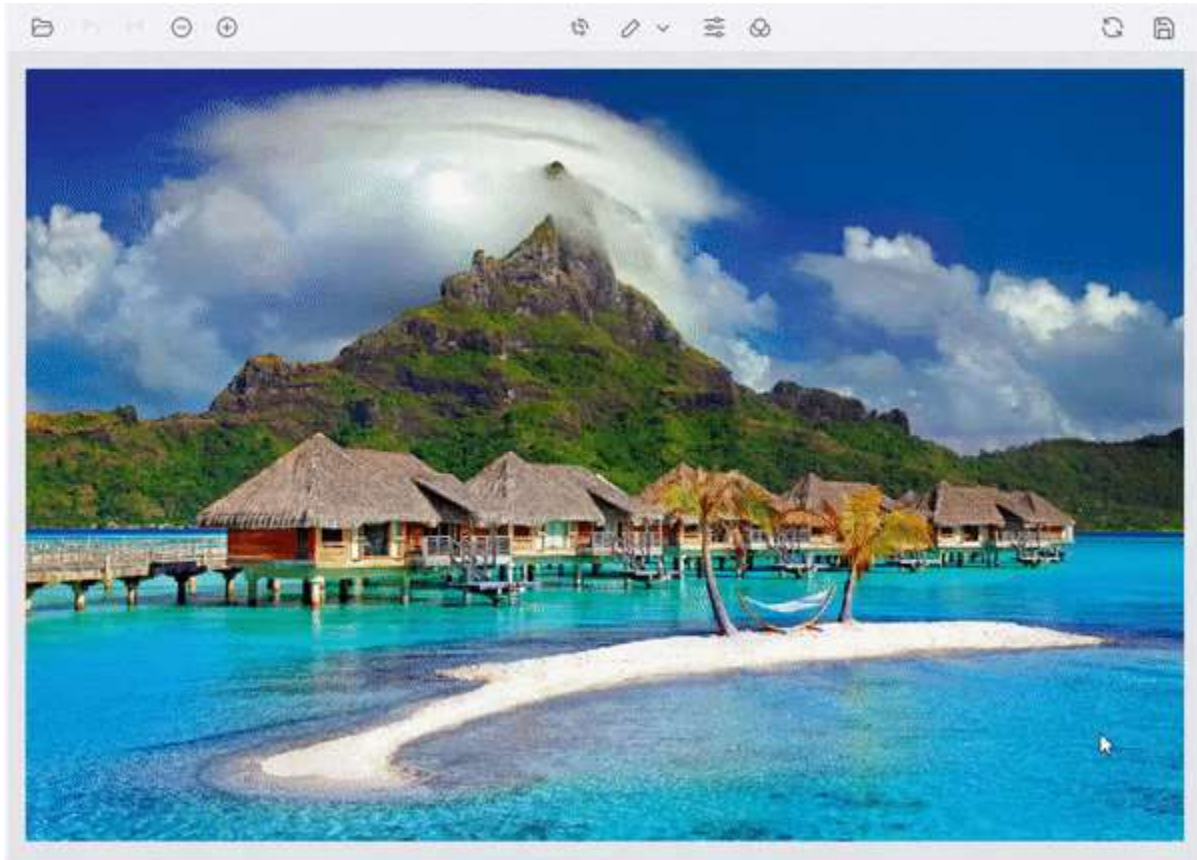
- Click the Open icon from the left side of the toolbar.
- The file explorer lists only JPEG, PNG, JPG format files.
- Select the image from the list of the images from the file explorer window.



#### Zooming

Image zooming can be performed in the following ways.

- Using toolbar.
- Using pinch zoom in touch enabled devices.
- Using mouse wheel.
- Using keyboard.



#### *Using toolbar*

To zoom in or out the image in the image editor, do the following steps.

- The Zoom In/ Out option only enabled after opening the image.

#### *Using pinch*

To zoom in or out the image in the image editor, do the following steps.

- Touch with two fingers to perform zooming.
- Zoom in and out controlled by touch gestures.

#### *Using Mouse wheel*

To zoom in or out the image in the image editor, do the following steps.

- Press the ctrl key and scroll the mouse wheel to perform zooming.
- The zoom in and out controlled by the mouse wheel.



### Using keyboard

To zoom in or out the image in the image editor, do the following steps.

- Press the ctrl key with '+' button from the keyboard to zoom in an image.
- Press the ctrl key with '-' button from the keyboard to zoom out an image.

### Panning

To pan an image in the image editor, do the following steps.

- Click on the image and do dragging to move or pan the image.
- Panning option will be enabled in the following two cases.
- If the selection is applied for cropping an image.
- If the image size exceeds the canvas size while zooming an image.

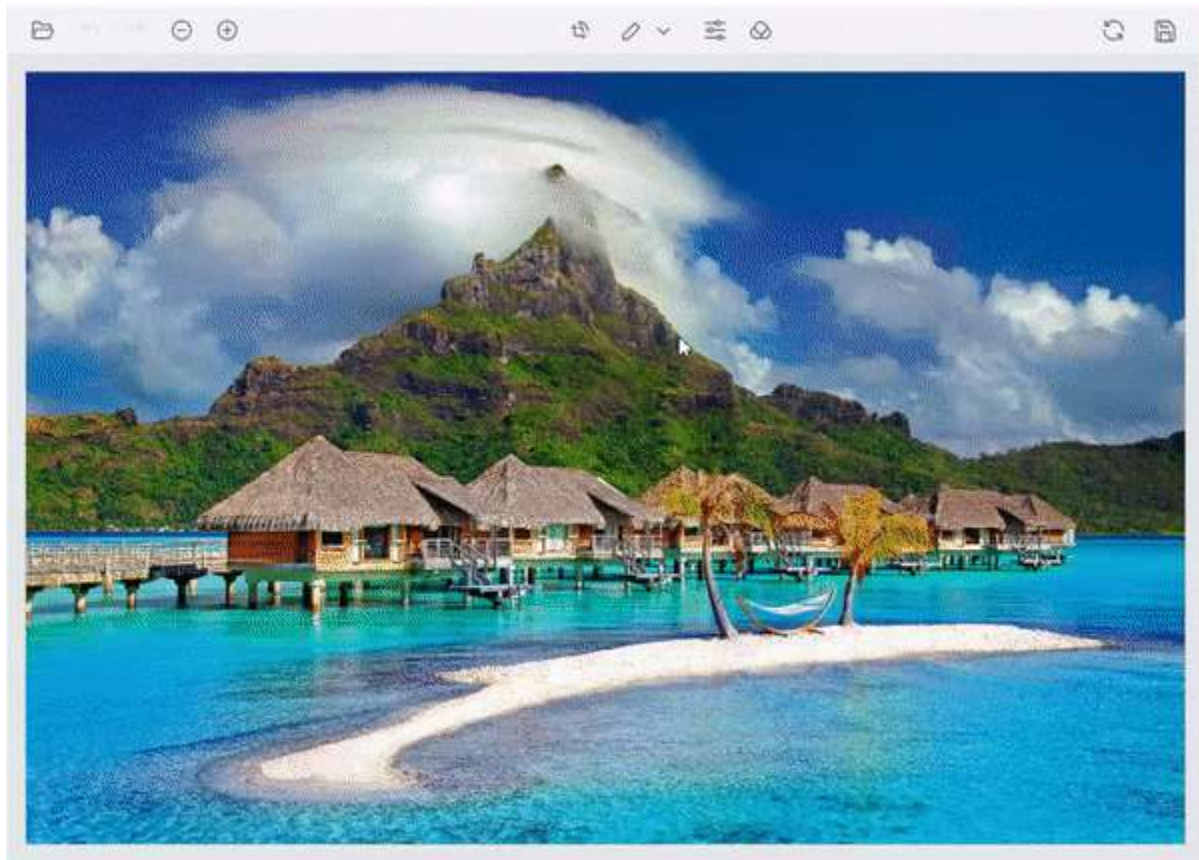


### Cropping and image transformation

To crop an image in the image editor, do the following steps.

- Cropping can be performed based on the selection in an image editor.
- To perform selection, click the crop button in the toolbar which opens the contextual toolbar that shows crop selection options, rotate options, and flip options.
- Click the crop selection button and select the type of selection such as custom, circle, square, and ratio selection from the popup.
- Once selection is completed, do panning to move the image to get the cropped region.

- Utilize the rotate or flip buttons to execute the image transformation, including any inserted annotations.
- Once the cropping region is finalized in the image click the tick icon at the top right of the toolbar to crop the image.



### Annotations

To add annotations to an image in the image editor, do the following steps.

- To add annotation, click the annotation button in the toolbar and select the type of annotations such as Line, Rectangle, Ellipse, Path, Arrow, Text, or Freehand drawing to be inserted to the image editor.
- Once the annotation is added to the image, that can be repositioned by clicking and dragging the annotations using mouse as well as resized by clicking and resizing the selection circle to be placed around the annotations.
- To rotate annotations, you can simply grab the circle located at the bottom of the annotation. The rotation can be applicable to all the annotations except text annotation.
- Customize the annotations by changing their color, stroke width, font family, and font size through the contextual toolbar. The contextual toolbar will be enabled whenever the annotations are selected.
- When annotations are selected in the Image Editor, the quick access toolbar becomes active, providing convenient access to various actions such as duplicating, deleting, or editing text associated with the selected annotation. This toolbar enables users to perform these common



operations quickly and efficiently, streamlining their workflow and enhancing the overall editing experience.



#### Filtering and fine-tune

To perform fine-tuning on an image in the image editor, do the following steps.

- Click the fine-tune button which displays the list of fine-tuning available in the image editor.
- Click one of the fine-tune options from the list of options which shows a slider to adjust the corresponding filter.
- Click on the canvas or tick icon at the right corner of the toolbar in the image editor to apply the modifications.

To apply filters on an image in the image editor, do the following steps.

- Click the filter button which displays the list of filters available in the image editor.
- Click the filter from list of options to apply the corresponding filter to an image.
- Click on the canvas or tick icon at the right corner of the toolbar in the image editor to apply the modifications.



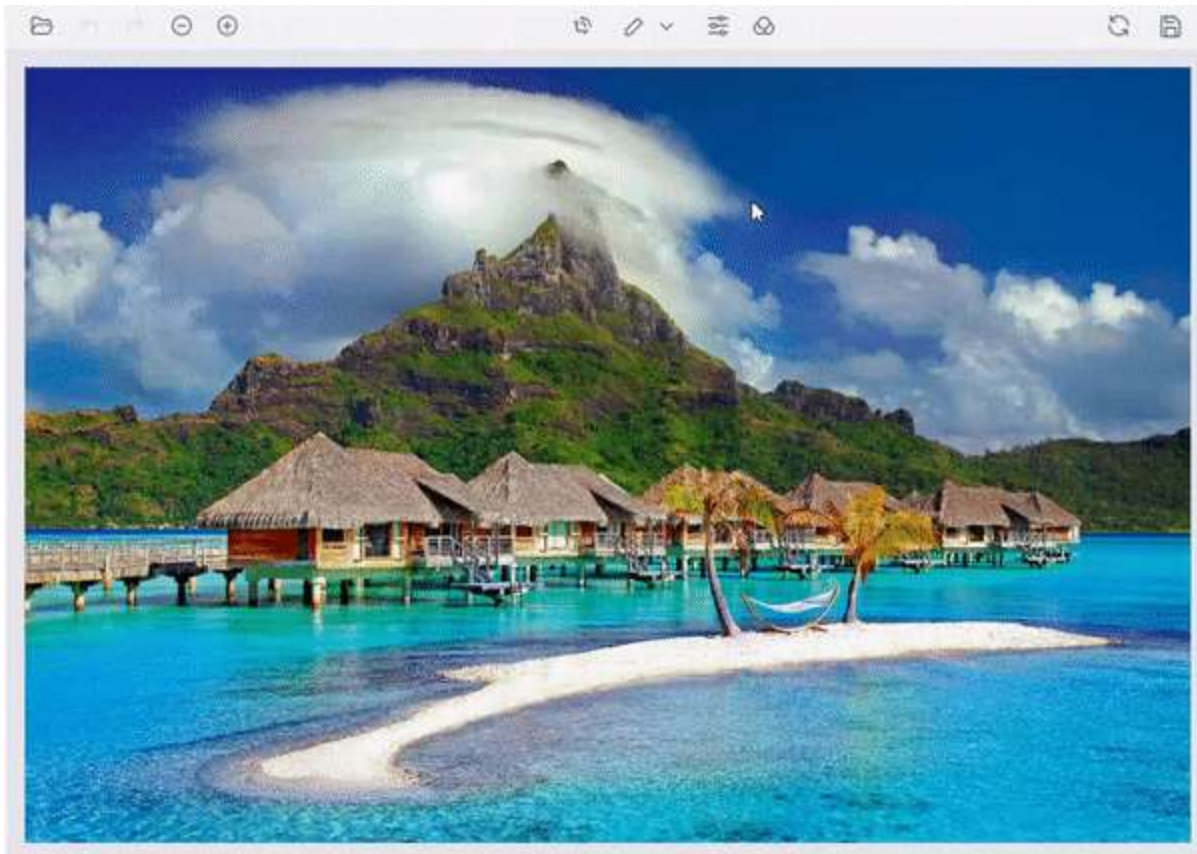
Note:

The Filters and Fine-tunes feature are not accessible within Safari due to compatibility limitations.

#### [Undo and redo the operations](#)

To undo and redo the actions performed in an image editor, do the following steps.

- The undo button will be enabled once the action is performed in an image editor.
- The redo button will be enabled once the undo action is performed in an image editor.
- Click the undo or redo button at the left side of the toolbar to perform undo and redo operation.
- Ctrl + Z and Ctrl + Y facilitates this process by allowing users to undo and redo actions, respectively.



### Reset an image

To revert all the changes done in an image editor, do the following steps.

- Click the reset button which is located on the right side of the toolbar.
- This will revert all the changes performed in the image editor.

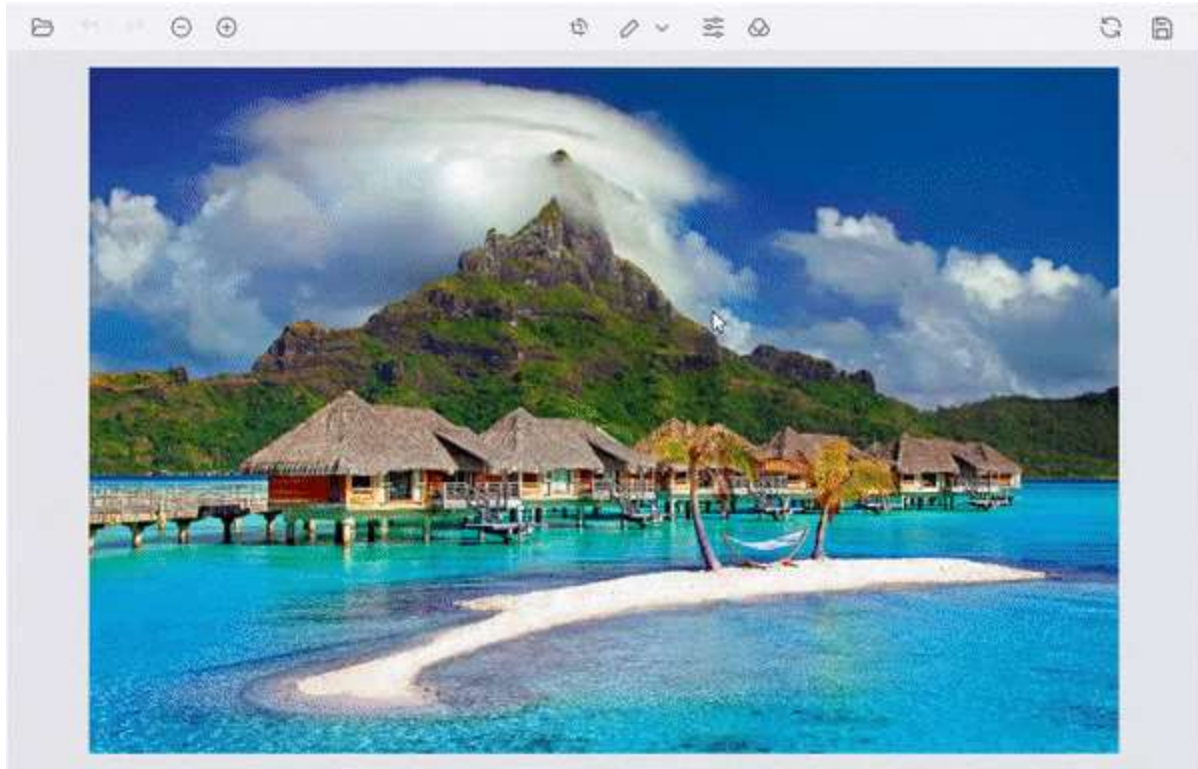




### Export an image

To save the modified image in an image editor, do the following steps.

- Click the save button which is located on the right side of the toolbar.
- Ctrl + S facilitates this process by providing users with the ability to save the image.
- Select the type of file to be saved from the popup to save with current modification done in an image.



### Open and save in the Vue Image Editor component

To import an image into the canvas, it must first be converted into a blob object. The Uploader component can be used to facilitate the process of uploading an image from the user interface. Once the image has been uploaded, it can then be converted into a blob and drawn onto the canvas.

To save an edited image in the Image Editor component, use the `toBlob` method to convert it to a blob object. This will save the image with any annotations or filters that have been applied during the editing process. The saved image can be stored as raw image data or as an image file.

### Open

The [open](#) method in the Image Editor component offers the capability to open an image by providing it in different formats. This method accepts various types of arguments, such as a base64-encoded string, raw image data, or a hosted/online URL. You can pass either the file name or the actual image data as an argument to the [open](#) method, and it will load the specified image into the image editor component. This flexibility allows you to work with images from different sources and formats, making it easier to integrate and manipulate images within the Image Editor component.

### APP.VUE

```
<template>
<div>
<ejs-imageeditor id="image-editor" ref="imageEditorObj" height="350px"
width="550px"></ejs-imageeditor>
</div>
</template>
<script>
import Vue from 'vue';
import { ImageEditorPlugin } from "@syncfusion/ej2-vue-image-editor";
import { Browser } from "@syncfusion/ej2-base";
```

```

Vue.use(ImageEditorPlugin);
export default {
  data: function() {
    return {
      toolbar: []
    };
  },
  methods: {
    created: function() {
      if (Browser.isDevice) {
        this.$refs.imageEditorObj.open('flower.png');
      } else {
        this.$refs.imageEditorObj.open('bridge.png');
      }
    }
  }
}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-buttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-splitbuttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-lists/styles/material.css";
@import "../node_modules/@syncfusion/ej2-popups/styles/material.css";
@import "../node_modules/@syncfusion/ej2-inputs/styles/material.css";
@import "../node_modules/@syncfusion/ej2-navigations/styles/material.css";
@import "../node_modules/@syncfusion/ej2-dropdowns/styles/material.css";
@import "../node_modules/@syncfusion/ej2-image-editor/styles/material.css";
#image-editor {
  width: 550px !important;
  height: 350px !important;
}
</style>

```

{% previewsample "page.domainurl/code-snippet/image-editor/getting-started-cs9" %}

### Supported image formats

The Image Editor component supports three common image formats: PNG, JPEG, and SVG. These formats allow you to work with a wide range of image files within the Image Editor.

When it comes to saving the edited image, the default file type is set as PNG. This means that when you save the edited image without specifying a different file type, it will be saved as a PNG file. However, it's important to note that the Image Editor typically provides options or methods to specify a different file type if desired. This allows you to save the edited image in formats other than the default PNG, such as JPEG or SVG, based on your specific requirements or preferences.

### Save

The Image Editor component saves the edited image as Image Data or images like PNG, JPEG, and SVG.

#### Save as ImageData

The [getImageData](#) method is used to get the image as ImageData and this can be loaded to our Image Editor component using the open method.

*Save as image*

The [export](#) method in the Image Editor component enables you to save the modified image as a file on the local device. This method accepts two parameters: the file name and the file type.

By providing a file name, you can specify the desired name for the saved image file. Additionally, you can also specify the file type to determine the format in which the image should be saved. This allows you to save the image according to your specific requirements or preferences, such as PNG, JPEG, or SVG.

By utilizing the [export](#) method with the appropriate file name and file type, you can conveniently save the modified image as a file on the local device, ensuring that it is easily accessible and shareable

In the following example, the [export](#) method is used in the button click event.

**APP.VUE**

```
<template>
<div>
<ejs-imageeditor id="image-editor" ref="imageEditorObj" height="350px"
width="550px"></ejs-imageeditor>
  <ejs-button cssClass="e-img-button" :isPrimary="true" v-
on:click.native="btnClick">Save</ejs-button>
</div>
</template>
<script>
import Vue from 'vue';
import { ImageEditorPlugin } from "@syncfusion/ej2-vue-image-editor";
import { ButtonPlugin } from '@syncfusion/ej2-vue-buttons';
import { Browser } from "@syncfusion/ej2-base";
Vue.use(ImageEditorPlugin);
Vue.use(ButtonPlugin);
export default {
  data: function() {
    return {
      toolbar: []
    };
  },
  methods: {
    created: function() {
      if (Browser.isDevice) {
        this.$refs.imageEditorObj.open('flower.png');
      } else {
        this.$refs.imageEditorObj.open('bridge.png');
      }
    },
    btnClick: function(event) {
      this.$refs.imageEditorObj.ej2Instances.export("PNG", "Syncfusion"); //
      File type, file name
    }
  }
}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-buttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-splitbuttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-lists/styles/material.css";
@import "../node_modules/@syncfusion/ej2-popups/styles/material.css";
```

```
@import "../node_modules/@syncfusion/ej2-inputs/styles/material.css";
@import "../node_modules/@syncfusion/ej2-navigations/styles/material.css";
@import "../node_modules/@syncfusion/ej2-dropdowns/styles/material.css";
@import "../node_modules/@syncfusion/ej2-image-editor/styles/material.css";
#image-editor {
    width: 550px !important;
    height: 350px !important;
}
</style>
```

{% previewsample "page.domainurl/code-snippet/image-editor/getting-started-cs10" %}

### File opened event

The [fileOpened](#) event is triggered in the Image Editor component after an image is successfully loaded. It provides the [OpenEventArgs](#) as the event argument, which contains two specific arguments:

- **FileName:** This argument is a string that contains the file name of the opened image. It represents the name of the file that was selected or provided when loading the image into the Image Editor.
- **FileType:** This argument is a string that contains the type of the opened image. It specifies the format or file type of the image that was loaded, such as PNG, JPEG, or SVG.

By accessing these arguments within the [fileOpened](#) event handler, you can retrieve information about the loaded image, such as its file name and file type. This can be useful for performing additional actions or implementing logic based on the specific image that was opened in the Image Editor component.

### Saving event

The [saving](#) event is triggered in the Image Editor component when an image is being saved to the local disk. It provides the [SaveEventArgs](#) as the event argument, which includes the following specific arguments:

- **FileName:** This argument is a string that holds the file name of the saved image. It represents the name of the file that will be used when saving the image to the local disk.
- **FileType:** This argument is a string indicating the type or format of the saved image. It specifies the desired file type in which the image will be saved, such as PNG, JPEG, or SVG.
- **Cancel:** This argument is a boolean value that can be set to true in order to cancel the saving action. By default, it is set to false, allowing the saving process to proceed. However, if you want to prevent the saving action from occurring, you can set Cancel to true within the event handler.

By accessing these arguments within the Saving event handler, you can retrieve information about the file name and file type of the image being saved. Additionally, you have the option to cancel the saving action if necessary.

### Created event

The [created](#) event is triggered once the Image Editor component is created. This event serves as a notification that the component has been fully initialized and is ready to be used. It provides a convenient opportunity to render the Image Editor with a predefined set of initial settings, including the image, annotations, and transformations.



### Destroyed event

The [destroyed](#) event is triggered once the Image Editor component is destroyed or removed from the application. This event serves as a notification that the component and its associated resources have been successfully cleaned up and are no longer active.

### Reset an image

The [reset](#) method in the Image Editor component provides the capability to undo all the changes made to an image and revert it back to its original state. This method is particularly useful when multiple adjustments, annotations, or transformations have been applied to an image and you want to start over with the original, unmodified version of the image.

By invoking the [reset](#) method, any modifications or edits made to the image will be undone, and the image will be restored to its initial state. This allows you to easily discard any changes and begin again with the fresh, unaltered image.

### Selection cropping in the Vue Image Editor component

The cropping feature in the Image Editor allows you to select and crop specific regions of an image. It offers different selection options, including custom shapes, squares, circles, and various aspect ratios such as 3:2, 4:3, 5:4, 7:5, and 16:9.

To perform a selection, you can use the [select](#) method, which allows you to define the desired selection area within the image. Once the selection is made, you can then use the [crop](#) method to crop the image based on the selected region. This enables you to extract and focus on specific parts of the image while discarding the rest.

### Insert custom / square / circle region

The [select](#) method allows to perform selection based on the type of selection. Here, the [select](#) method is used to perform the selection as custom, circle, or square. The selection region can also be customized using the select method based on the parameters below.

type - Specify the type of selection

startX - Specify the x-coordinate of the selection region's starting point

startY - Specify the y-coordinate of the selection region's starting point

width - Specify the width of the selection region

height - Specify the height of the selection region

Here is an example of square selection using the [select](#) method.

### APP.VUE

```
<template>
<div>
<ejs-imageeditor id="image-editor" ref="imageEditorObj" height="350px"
width="550px"></ejs-imageeditor>
  <ejs-button cssClass="e-img-button" :isPrimary="true" v-
on:click.native="btnClick">Square selection</ejs-button>
</div>
</template>
<script>
import Vue from 'vue';
import { ImageEditorPlugin } from "@syncfusion/ej2-vue-image-editor";
import { ButtonPlugin } from '@syncfusion/ej2-vue-buttons';
```

```

import { Browser } from "@syncfusion/ej2-base";
Vue.use(ImageEditorPlugin);
Vue.use(ButtonPlugin);
export default {
  data: function() {
    return {
      toolbar: []
    };
  },
  methods: {
    created: function() {
      if (Browser.isDevice) {
        this.$refs.imageEditorObj.open('flower.png');
      } else {
        this.$refs.imageEditorObj.open('bridge.png');
      }
    },
    btnClick: function(event) {
      this.$refs.imageEditorObj.ej2Instances.select("Square");
    }
  }
}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-buttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-splitbuttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-lists/styles/material.css";
@import "../node_modules/@syncfusion/ej2-popups/styles/material.css";
@import "../node_modules/@syncfusion/ej2-inputs/styles/material.css";
@import "../node_modules/@syncfusion/ej2-navigations/styles/material.css";
@import "../node_modules/@syncfusion/ej2-dropdowns/styles/material.css";
@import "../node_modules/@syncfusion/ej2-image-editor/styles/material.css";
#image-editor {
  width: 550px !important;
  height: 350px !important;
}
</style>

```

{% previewsample "page.domainurl/code-snippet/image-editor/getting-started-cs12" %}

#### Insert selection based on aspect ratio

The [select](#) method is used to perform the selection with the various aspect ratios such as 3:2, 4:3, 5:4, 7:5, and 16:9. The selection region can also be customized using the [select](#) method based on the parameters below.

type - Specify the type of selection

startX - Specify the x-coordinate of the selection region's starting point

startY - Specify the y-coordinate of the selection region's starting point

Here is an example of ratio selection using the [select](#) method.

#### APP.VUE

```
<template>
```

```

<div>
<ejs-imageeditor id="image-editor" ref="imageEditorObj" height="350px"
width="550px"></ejs-imageeditor>
  <ejs-button cssClass="e-img-button" :isPrimary="true" v-
on:click.native="btnClick">Ratio selection</ejs-button>
</div>
</template>
<script>
import Vue from 'vue';
import { ImageEditorPlugin } from "@syncfusion/ej2-vue-image-editor";
import { ButtonPlugin } from "@syncfusion/ej2-vue-buttons";
import { Browser } from "@syncfusion/ej2-base";
Vue.use(ImageEditorPlugin);
Vue.use(ButtonPlugin);
export default {
  data: function() {
    return {
      toolbar: []
    };
  },
  methods: {
    created: function() {
      if (Browser.isDevice) {
        this.$refs.imageEditorObj.open('flower.png');
      } else {
        this.$refs.imageEditorObj.open('bridge.png');
      }
    },
    btnClick: function(event) {
      this.$refs.imageEditorObj.ej2Instances.select("16:9");
    }
  }
}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-buttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-splitbuttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-lists/styles/material.css";
@import "../node_modules/@syncfusion/ej2-popups/styles/material.css";
@import "../node_modules/@syncfusion/ej2-inputs/styles/material.css";
@import "../node_modules/@syncfusion/ej2-navigations/styles/material.css";
@import "../node_modules/@syncfusion/ej2-dropdowns/styles/material.css";
@import "../node_modules/@syncfusion/ej2-image-editor/styles/material.css";
#image-editor {
  width: 550px !important;
  height: 350px !important;
}
</style>

```

{% previewsample "page.domainurl/code-snippet/image-editor/getting-started-cs14" %}

### Resize selections

The selection region can be changed programmatically by using [selectionChanging](#) event. This event is activated during resizing the selection using mouse, and it allows for alterations to the selection region by adjusting the specified properties.

The [SelectionChangeEventArgs](#) is used in this event to customize the selection and it has the following parameters.

SelectionChangeEventArgs.action - The type of action such as inserting or resizing

SelectionChangeEventArgs.cancel - Specifies to cancel the selection.

SelectionChangeEventArgs.currentSelectionPoint - Represents all the details of the selection including its type, position, width, and height after the current action as CropSelectionSettings.

SelectionChangeEventArgs.previousSelectionPoint - Represents all the details of the selection including its type, position, width, and height before this current action as CropSelectionSettings

Here is an example of changing the selection region using the [SelectionChangeEventArgs](#) event.

### Crop an image

The [crop](#) method allows cropping based on the selected region. Here is an example of cropping the selection region using the [crop](#) method.

Here is an example of circle cropping using the [select](#) and [crop](#) method.

### APP.VUE

```
<template>
<div>
<ejs-imageeditor id="image-editor" ref="imageEditorObj" height="350px"
width="550px"></ejs-imageeditor>
  <ejs-button cssClass="e-img-button" :isPrimary="true" v-
on:click.native="btnClick">Circle crop</ejs-button>
</div>
</template>
<script>
import Vue from 'vue';
import { ImageEditorPlugin } from "@syncfusion/ej2-vue-image-editor";
import { ButtonPlugin } from '@syncfusion/ej2-vue-buttons';
import { Browser } from "@syncfusion/ej2-base";
Vue.use(ImageEditorPlugin);
Vue.use(ButtonPlugin);
export default {
  data: function() {
    return {
      toolbar: []
    };
  },
  methods: {
    created: function() {
      if (Browser.isDevice) {
        this.$refs.imageEditorObj.open('flower.png');
      } else {
        this.$refs.imageEditorObj.open('bridge.png');
      }
    },
    btnClick: function(event) {
```

```

        this.$refs.imageEditorObj.ej2Instances.select("Circle");
        this.$refs.imageEditorObj.ej2Instances.crop();
    }
}
}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-buttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-splitbuttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-lists/styles/material.css";
@import "../node_modules/@syncfusion/ej2-popups/styles/material.css";
@import "../node_modules/@syncfusion/ej2-inputs/styles/material.css";
@import "../node_modules/@syncfusion/ej2-navigations/styles/material.css";
@import "../node_modules/@syncfusion/ej2-dropdowns/styles/material.css";
@import "../node_modules/@syncfusion/ej2-image-editor/styles/material.css";
#image-editor {
    width: 550px !important;
    height: 350px !important;
}
</style>

```

{% previewsample "page.domainurl/code-snippet/image-editor/getting-started-cs15" %}

### Cropping event

The [cropping](#) event is triggered when performing cropping on the image. This event is passed an object that contains information about the cropping event, such as the start and end point of the selection region. And this event uses [CropEventArgs](#) to handle the cropping action in the image.

The parameter available in the [cropping](#) event is,

CroppingEventArgs.startPoint – The x and y coordinates of a start point as ImageEditorPoint of the selection region.

CroppingEventArgs.endPoint - The x and y coordinates of an end point as ImageEditorPoint of the selection region.

CroppingEventArgs.cancel - To cancel the cropping action.

### Annotation in the Vue Image Editor component

The Vue Image Editor allows adding annotations to the image, including text, freehand drawings, and shapes like rectangles, ellipses, arrows, paths, and lines. This gives the flexibility to mark up the image with notes, sketches, and other visual elements as needed. These annotation tools can help to communicate and share ideas more effectively.

#### Text annotation

The text annotation feature in the Image Editor provides the capability to add and customize labels, captions, and other text elements directly onto the image. With this feature, you can easily insert text at specific locations within the image and customize various aspects of the text to meet your requirements.

You have control over the customization options including text content, font family, font style and font size for the text annotation.

### Add a text

The [drawText](#) method in the Vue Image Editor allows you to insert a text annotation into the image with specific customization options. This method accepts the following parameters:

- x: Specifies the x-coordinate of the text, determining its horizontal position within the image.
- y: Specifies the y-coordinate of the text, determining its vertical position within the image.
- text: Specifies the actual text content to be added to the image.
- fontFamily: Specifies the font family of the text, allowing you to choose a specific typeface or style for the text.
- fontSize: Specifies the font size of the text, determining its relative size within the image.
- bold: Specifies whether the text should be displayed in bold style. Set to true for bold text, and false for regular text.
- italic: Specifies whether the text should be displayed in italic style. Set to true for italic text, and false for regular text.
- color: Specifies the font color of the text, allowing you to define the desired color using appropriate color values or names.

By utilizing the [drawText](#) method with these parameters, you can precisely position and customize text annotations within the image. This provides the flexibility to add labels, captions, or other text elements with specific font styles, sizes, and colors, enhancing the visual presentation and clarity of the image.

Here is an example of adding a text in a button click using [drawText](#) method.

### APP.VUE

```
<template>
<div>
<ejs-imageeditor id="image-editor" ref="imageEditorObj" height="350px"
width="550px" :created="created" :toolbar="toolbar"></ejs-imageeditor>
<ejs-button cssClass="e-img-button" :isPrimary="true" v-
on:click.native="btnClick">Text</ejs-button>
</div>
</template>
<script>
import Vue from 'vue';
import { ImageEditorPlugin } from "@syncfusion/ej2-vue-image-editor";
import { ButtonPlugin } from '@syncfusion/ej2-vue-buttons';
import { Browser } from "@syncfusion/ej2-base";
Vue.use(ImageEditorPlugin);
Vue.use(ButtonPlugin);
export default {
  data: function() {
    return {
      toolbar: []
    };
  },
  methods: {
    created: function() {
      if (Browser.isDevice) {
        this.$refs.imageEditorObj.open('flower.jpeg');
      } else {
        this.$refs.imageEditorObj.open('bridge.jpeg');
      }
    }
  },
}
```

```

    btnClick: function(event) {
      let dimension =
this.$refs.imageEditorObj.ej2Instances.getImageDimension();

this.$refs.imageEditorObj.ej2Instances.drawText(dimension.x,dimension.y);
    }
  }
}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-buttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-splitbuttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-lists/styles/material.css";
@import "../node_modules/@syncfusion/ej2-popups/styles/material.css";
@import "../node_modules/@syncfusion/ej2-inputs/styles/material.css";
@import "../node_modules/@syncfusion/ej2-navigations/styles/material.css";
@import "../node_modules/@syncfusion/ej2-dropdowns/styles/material.css";
@import "../node_modules/@syncfusion/ej2-image-editor/styles/material.css";
#image-editor {
  width: 550px !important;
  height: 350px !important;
}
</style>

```

{% previewsample "page.domainurl/code-snippet/image-editor/getting-started-cs1" %}

#### Multiline text

The [drawText](#) method in the Vue Image Editor component is commonly used to insert text annotations into an image. If the provided text parameter contains a newline character (\n), the text will be automatically split into multiple lines, with each line appearing on a separate line in the annotation.

Here is an example of adding a multiline text in a button click using [drawText](#) method.

#### APP.VUE

```

<template>
<div>
<ejs-imageeditor id="image-editor" ref="imageEditorObj" height="350px"
width="550px"></ejs-imageeditor>
<ejs-button cssClass="e-img-button" :isPrimary="true" v-
on:click.native="btnClick">Text</ejs-button>
</div>
</template>
<script>
import Vue from 'vue';
import { ImageEditorPlugin } from "@syncfusion/ej2-vue-image-editor";
import { ButtonPlugin } from '@syncfusion/ej2-vue-buttons';
import { Browser } from "@syncfusion/ej2-base";
Vue.use(ImageEditorPlugin);
Vue.use(ButtonPlugin);
export default {
  data: function() {
    return {
      toolbar: []
    };
  };
}

```

```

    },
    methods: {
      created: function() {
        if (Browser.isDevice) {
          this.$refs.imageEditorObj.open('flower.png');
        } else {
          this.$refs.imageEditorObj.open('bridge.png');
        }
      },
      btnClick: function(event) {
        let dimension =
this.$refs.imageEditorObj.ej2Instances.getImageDimension();

this.$refs.imageEditorObj.ej2Instances.drawText(dimension.x,dimension.y, 'Enter\nText');
      }
    }
  }
</script>
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-buttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-splitbuttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-lists/styles/material.css";
@import "../node_modules/@syncfusion/ej2-popups/styles/material.css";
@import "../node_modules/@syncfusion/ej2-inputs/styles/material.css";
@import "../node_modules/@syncfusion/ej2-navigations/styles/material.css";
@import "../node_modules/@syncfusion/ej2-dropdowns/styles/material.css";
@import "../node_modules/@syncfusion/ej2-image-editor/styles/material.css";
#image-editor {
  width: 550px !important;
  height: 350px !important;
}
</style>

```

{% previewsample "page.domainurl/code-snippet/image-editor/getting-started-cs27" %}

### Delete a text

The [deleteShape](#) method in the Vue Image Editor allows you to remove a text annotation from the image editor. To use this method, you need to pass the [shapeld](#) of the annotation as a parameter.

The [shapeld](#) is a unique identifier assigned to each text annotation within the image editor. It serves as a reference to a specific annotation, enabling targeted deletion of the desired text element. By specifying the [shapeld](#) associated with the text annotation you want to remove, you can effectively delete it from the image editor.

To retrieve the inserted text annotations, you can utilize the [getShapeSetting](#) method, which provides a collection of annotations represented by [ShapeSettings](#). This method allows you to access and work with the annotations that have been inserted into the image.

Here is an example of deleting a text in a button click using [deleteShape](#) method.

### APP.VUE

```

<template>
<div>

```



```

<ejs-imageeditor id="image-editor" ref="imageEditorObj" height="350px"
width="550px"></ejs-imageeditor>
<ejs-button cssClass="e-img-button" :isPrimary="true" v-
on:click.native="drawClick">Text</ejs-button>
<ejs-button cssClass="e-img-button" :isPrimary="true" v-
on:click.native="btnClick">Delete</ejs-button>
</div>
</template>
<script>
import Vue from 'vue';
import { ImageEditorPlugin } from "@syncfusion/ej2-vue-image-editor";
import { ButtonPlugin } from '@syncfusion/ej2-vue-buttons';
import { Browser } from '@syncfusion/ej2-base';
Vue.use(ImageEditorPlugin);
Vue.use(ButtonPlugin);
export default {
  data: function() {
    return {
      toolbar: []
    };
  },
  methods: {
    created: function() {
      if (Browser.isDevice) {
        this.$refs.imageEditorObj.open('flower.png');
      } else {
        this.$refs.imageEditorObj.open('bridge.png');
      }
    },
    drawClick: function(event) {
      let dimension =
this.$refs.imageEditorObj.ej2Instances.getImageDimension();

this.$refs.imageEditorObj.ej2Instances.drawText(dimension.x,dimension.y,'Ent
er\nText');
    },
    btnClick: function(event) {
      this.$refs.imageEditorObj.ej2Instances.deleteShape('shape_1');
    }
  }
}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-buttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-splitbuttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-lists/styles/material.css";
@import "../node_modules/@syncfusion/ej2-popups/styles/material.css";
@import "../node_modules/@syncfusion/ej2-inputs/styles/material.css";
@import "../node_modules/@syncfusion/ej2-navigations/styles/material.css";
@import "../node_modules/@syncfusion/ej2-dropdowns/styles/material.css";
@import "../node_modules/@syncfusion/ej2-image-editor/styles/material.css";
#image-editor {
  width: 550px !important;
  height: 350px !important;
}
</style>

```

```
{% previewsample "page.domainurl/code-snippet/image-editor/getting-started-cs28" %}
```

### *Customize font family and text color*

The [shapeChanging](#) event in the Image Editor component is triggered when a text annotation is being modified or changed through the toolbar interaction. This event provides an opportunity to make alterations to the text's color and font family by adjusting the relevant properties.

By leveraging the [shapeChanging](#) event, you can enhance the customization options for text annotations and provide a more tailored and interactive experience within the Image Editor component.

Here is an example of changing the text's color and its font family using the [shapeChanging](#) event.

### **APP.VUE**

```
<template>
<div>
<ejs-imageeditor id="image-editor" ref="imageEditorObj" height="350px"
width="550px" :shapeChanging="shapeChanging"></ejs-imageeditor>
</div>
</template>
<script>
import Vue from 'vue';
import { ImageEditorPlugin, ImageFinetuneOption } from "@syncfusion/ej2-vue-
image-editor";
import { ButtonPlugin } from '@syncfusion/ej2-vue-buttons';
import { Browser } from "@syncfusion/ej2-base";
Vue.use(ImageEditorPlugin);
Vue.use(ButtonPlugin);
Vue.use(ImageFinetuneOption);
export default {
  data: function() {
    return {};
  },
  methods: {
    created: function() {
      if (Browser.isDevice) {
        this.$refs.imageEditorObj.open('flower.png');
      } else {
        this.$refs.imageEditorObj.open('bridge.png');
      }
    },
    shapeChanging: function(args) {
      if (args.currentShapeSettings.type === 'Text') {
        args.currentShapeSettings.color = 'red',
        args.currentShapeSettings.fontStyle = ['bold'],
        args.currentShapeSettings.fontSize = 20,
        args.currentShapeSettings.text = 'Syncfusion'
      }
    }
  }
}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-buttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-splitbuttons/styles/material.css";
```

```
@import "../node_modules/@syncfusion/ej2-lists/styles/material.css";
@import "../node_modules/@syncfusion/ej2-popups/styles/material.css";
@import "../node_modules/@syncfusion/ej2-inputs/styles/material.css";
@import "../node_modules/@syncfusion/ej2-navigations/styles/material.css";
@import "../node_modules/@syncfusion/ej2-dropdowns/styles/material.css";
@import "../node_modules/@syncfusion/ej2-image-editor/styles/material.css";
#image-editor {
    width: 550px !important;
    height: 350px !important;
}
</style>
```

{% previewsample "page.domainurl/code-snippet/image-editor/getting-started-cs29" %}

### Freehand drawing

The Freehand Draw annotation tool in the Vue Image Editor component is a versatile feature that allows users to draw and sketch directly on the image using mouse or touch input. This tool provides a flexible and creative way to add freehand drawings or annotations to the image.

The [freehandDraw](#) method is used to enable or disable the freehand drawing option in the Vue Image Editor component.

Here is an example of using the [freeHandDraw](#) method in a button click event.

### APP.VUE

```
<template>
<div>
<ejs-imageeditor id="image-editor" ref="imageEditorObj" height="350px"
width="550px"></ejs-imageeditor>
<ejs-button cssClass="e-img-button" :isPrimary="true" v-
on:click.native="btnClick">Freehand draw</ejs-button>
</div>
</template>
<script>
import Vue from 'vue';
import { ImageEditorPlugin } from "@syncfusion/ej2-vue-image-editor";
import { ButtonPlugin } from '@syncfusion/ej2-vue-buttons';
import { Browser } from "@syncfusion/ej2-base";
Vue.use(ImageEditorPlugin);
Vue.use(ButtonPlugin);
export default {
    data: function() {
        return {
            toolbar: []
        };
    },
    methods: {
        created: function() {
            if (Browser.isDevice) {
                this.$refs.imageEditorObj.open('flower.png');
            } else {
                this.$refs.imageEditorObj.open('bridge.png');
            }
        },
        btnClick: function(event) {
```

```

        this.$refs.imageEditorObj.ej2Instances.freeHandDraw(true);
    }
}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-buttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-splitbuttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-lists/styles/material.css";
@import "../node_modules/@syncfusion/ej2-popups/styles/material.css";
@import "../node_modules/@syncfusion/ej2-inputs/styles/material.css";
@import "../node_modules/@syncfusion/ej2-navigations/styles/material.css";
@import "../node_modules/@syncfusion/ej2-dropdowns/styles/material.css";
@import "../node_modules/@syncfusion/ej2-image-editor/styles/material.css";
#image-editor {
    width: 550px !important;
    height: 350px !important;
}
</style>

```

{% previewsample "page.domainurl/code-snippet/image-editor/getting-started-cs2" %}

### *Delete a freehand drawing*

The [deleteShape](#) method in the Vue Image Editor allows you to remove a freehand annotation from the image editor. To use this method, you need to pass the [shapeld](#) of the annotation as a parameter.

The [shapeld](#) is a unique identifier assigned to each freehand annotation within the image editor. It serves as a reference to a specific annotation, enabling targeted deletion of the desired annotation. By specifying the [shapeld](#) associated with the freehand annotation you want to remove, you can effectively delete it from the image editor.

To retrieve the inserted freehand annotations, you can utilize the [getShapeSetting](#) method, which provides a collection of annotations represented by [ShapeSettings](#). This method allows you to access and work with the annotations that have been inserted into the image.

Here is an example of deleting a freehand annotation in a button click using [deleteShape](#) method.

### **APP.VUE**

```

<template>
<div>
<ejs-imageeditor id="image-editor" ref="imageEditorObj" height="350px"
width="550px"></ejs-imageeditor>
<ejs-button cssClass="e-img-button" :isPrimary="true" v-
on:click.native="drawClick">Draw</ejs-button>
<ejs-button cssClass="e-img-button" :isPrimary="true" v-
on:click.native="btnClick">Delete</ejs-button>
</div>
</template>
<script>
import Vue from 'vue';
import { ImageEditorPlugin } from "@syncfusion/ej2-vue-image-editor";
import { ButtonPlugin } from "@syncfusion/ej2-vue-buttons";
import { Browser } from "@syncfusion/ej2-base";
Vue.use(ImageEditorPlugin);

```

```

Vue.use(ButtonPlugin);
export default {
  data: function() {
    return {
      toolbar: []
    };
  },
  methods: {
    created: function() {
      if (Browser.isDevice) {
        this.$refs.imageEditorObj.open('flower.png');
      } else {
        this.$refs.imageEditorObj.open('bridge.png');
      }
    },
    drawClick: function(event) {
      this.$refs.imageEditorObj.ej2Instances.freeHandDraw(true);
    },
    btnClick: function(event) {
      this.$refs.imageEditorObj.ej2Instances.deleteShape('pen_1');
    }
  }
}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-buttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-splitbuttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-lists/styles/material.css";
@import "../node_modules/@syncfusion/ej2-popups/styles/material.css";
@import "../node_modules/@syncfusion/ej2-inputs/styles/material.css";
@import "../node_modules/@syncfusion/ej2-navigations/styles/material.css";
@import "../node_modules/@syncfusion/ej2-dropdowns/styles/material.css";
@import "../node_modules/@syncfusion/ej2-image-editor/styles/material.css";
#image-editor {
  width: 550px !important;
  height: 350px !important;
}
</style>

```

{% previewsample "page.domainurl/code-snippet/image-editor/getting-started-cs33" %}

#### *Adjust the stroke width and color*

The [shapeChanging](#) event in the Vue Image Editor component is triggered when a freehand annotation is being modified or changed through the toolbar interaction. This event provides an opportunity to make alterations to the freehand annotation's color and stroke width by adjusting the relevant properties.

By leveraging the [shapeChanging](#) event, you can enhance the customization options for freehand annotations and provide a more tailored and interactive experience within the Image Editor component.

Here is an example of changing the freehand draw stroke width and color using the [shapeChanging](#) event.

#### **APP.VUE**

```

<template>
<div>
<ejs-imageeditor id="image-editor" ref="imageEditorObj" height="350px"
width="550px" :shapeChanging="shapeChanging"></ejs-imageeditor>
</div>
</template>
<script>
import Vue from 'vue';
import { ImageEditorPlugin, ImageFinetuneOption } from "@syncfusion/ej2-vue-
image-editor";
import { ButtonPlugin } from '@syncfusion/ej2-vue-buttons';
import { Browser } from "@syncfusion/ej2-base";
Vue.use(ImageEditorPlugin);
Vue.use(ButtonPlugin);
Vue.use(ImageFinetuneOption);
export default {
  data: function() {
    return {};
  },
  methods: {
    created: function() {
      if (Browser.isDevice) {
        this.$refs.imageEditorObj.open('flower.png');
      } else {
        this.$refs.imageEditorObj.open('bridge.png');
      }
    },
    shapeChanging: function(args) {
      if (args.currentShapeSettings.type === 'pen') {
        args.currentShapeSettings.color = 'red',
        args.currentShapeSettings.strokeWidth = 10
      }
    }
  }
}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-buttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-splitbuttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-lists/styles/material.css";
@import "../node_modules/@syncfusion/ej2-popups/styles/material.css";
@import "../node_modules/@syncfusion/ej2-inputs/styles/material.css";
@import "../node_modules/@syncfusion/ej2-navigations/styles/material.css";
@import "../node_modules/@syncfusion/ej2-dropdowns/styles/material.css";
@import "../node_modules/@syncfusion/ej2-image-editor/styles/material.css";
#image-editor {
  width: 550px !important;
  height: 350px !important;
}
</style>

```

{% previewsample "page.domainurl/code-snippet/image-editor/getting-started-cs34" %}

### Shape annotation

The Image Editor component provides the ability to add shape annotations to an image. These shape annotations include rectangles, ellipses, arrows, paths, and lines, allowing you to highlight, emphasize, or mark specific areas or elements within the image.

#### *Add a rectangle / ellipse / line / arrow / path*

The [drawRectangle](#) method is used to insert a rectangle to the Vue Image Editor component. Rectangle annotations are valuable tools for highlighting, emphasizing, or marking specific areas of an image to draw attention or provide additional context.

The [drawRectangle](#) method in the Vue Image Editor component takes seven parameters to define the properties of the rectangle annotation:

- x: Specifies the x-coordinate of the top-left corner of the rectangle.
- y: Specifies the y-coordinate of the top-left corner of the rectangle.
- width: Specifies the width of the rectangle.
- height: Specifies the height of the rectangle.
- strokeWidth: Specifies the stroke width of the rectangle's border.
- strokeColor: Specifies the stroke color of the rectangle's border.
- fillColor: Specifies the fill color of the rectangle.

The [drawEllipse](#) method is used to insert a ellipse to the Vue Image Editor component. Ellipse annotations are valuable for highlighting, emphasizing, or marking specific areas of an image.

The [drawEllipse](#) method in the Vue Image Editor component takes seven parameters to define the properties of the ellipse annotation:

- x: Specifies the x-coordinate of the center of the ellipse.
- y: Specifies the y-coordinate of the center of the ellipse.
- radiusX: Specifies the horizontal radius (radiusX) of the ellipse.
- radiusY: Specifies the vertical radius (radiusY) of the ellipse.
- strokeWidth: Specifies the width of the ellipse's stroke (border).
- strokeColor: Specifies the color of the ellipse's stroke (border).
- fillColor: Specifies the fill color of the ellipse.

The [drawLine](#) method is used to insert a line to the Vue Image Editor component. Line annotations are valuable for highlighting, emphasizing, or marking specific areas of an image.

The [drawLine](#) method in the Vue Image Editor component takes seven parameters to define the properties of the ellipse annotation:

- startX - Specifies the x-coordinate of the start point.
- startY - Specifies the y-coordinate of the start point.
- endX - Specifies the x-coordinate of the end point.
- endY - Specifies the y-coordinate of the end point.
- strokeWidth - Specifies the stroke width of the line.
- strokeColor - Specifies the stroke color of the line.

The [drawArrow](#) method is used to insert a arrow to the Vue Image Editor component. Arrow annotations are valuable for highlighting, emphasizing, or marking specific areas of an image.

The [drawArrow](#) method in the Vue Image Editor component takes seven parameters to define the properties of the ellipse annotation:

- startX - Specifies the x-coordinate of the start point.
- startY - Specifies the y-coordinate of the start point.
- endX - Specifies the x-coordinate of the end point.
- endY - Specifies the y-coordinate of the end point.
- strokeWidth - Specifies the stroke width of the arrow.
- strokeColor - Specifies the stroke color of the arrow.
- arrowStart - Specifies the arrowhead as ImageEditorArrowHeadType at the start of arrow.
- arrowEnd - Specifies the arrowhead as ImageEditorArrowHeadType at the end of the arrow.

The [drawPath](#) method is used to insert a path to the Vue Image Editor component. Path annotations are valuable for highlighting, emphasizing, or marking specific areas of an image.

The [drawPath](#) method in the Vue Image Editor component takes three parameters to define the properties of the ellipse annotation:

- points - Specifies collection of x and y coordinates as ImageEditorPoint to draw a path.
- strokeWidth - Specifies the stroke width of the path.
- strokeColor - Specifies the stroke color of the path.

Here is an example of inserting rectangle, ellipse, arrow, path, and line in a button click event.

#### APP.VUE

```
<template>
<div>
<ejs-imageeditor id="image-editor" ref="imageEditorObj" height="350px"
width="550px"></ejs-imageeditor>
  <ejs-button cssClass="e-img-button" :isPrimary="true" v-
on:click.native="rectangleClick">Rectangle</ejs-button>
  <ejs-button cssClass="e-img-button" :isPrimary="true" v-
on:click.native="ellipseClick">Ellipse</ejs-button>
  <ejs-button cssClass="e-img-button" :isPrimary="true" v-
on:click.native="lineClick">Line</ejs-button>
  <ejs-button cssClass="e-img-button" :isPrimary="true" v-
on:click.native="pathClick">Path</ejs-button>
  <ejs-button cssClass="e-img-button" :isPrimary="true" v-
on:click.native="arrowClick">Arrow</ejs-button>
</div>
</template>
<script>
import Vue from 'vue';
import { ImageEditorPlugin } from "@syncfusion/ej2-vue-image-editor";
import { ButtonPlugin } from "@syncfusion/ej2-vue-buttons";
import { Browser } from "@syncfusion/ej2-base";
Vue.use(ImageEditorPlugin);
Vue.use(ButtonPlugin);
export default {
  data: function() {
    return {
      toolbar: []
    };
  };
};
```



```

    },
    methods: {
      created: function() {
        if (Browser.isDevice) {
          this.$refs.imageEditorObj.open('flower.png');
        } else {
          this.$refs.imageEditorObj.open('bridge.png');
        }
      },
      rectangleClick: function(event) {
        let dimension =
this.$refs.imageEditorObj.ej2Instances.getImageDimension();

this.$refs.imageEditorObj.ej2Instances.drawRectangle(dimension.x,dimension.y
);
      },
      ellipseClick: function(event) {
        let dimension =
this.$refs.imageEditorObj.ej2Instances.getImageDimension();

this.$refs.imageEditorObj.ej2Instances.drawEllipse(dimension.x,dimension.y);
      },
      lineClick: function(event) {
        let dimension =
this.$refs.imageEditorObj.ej2Instances.getImageDimension();

this.$refs.imageEditorObj.ej2Instances.drawLine(dimension.x,dimension.y);
      },
      pathClick: function(event) {
        let dimension =
this.$refs.imageEditorObj.ej2Instances.getImageDimension();
        this.$refs.imageEditorObj.ej2Instances.drawPath([
{x: dimension.x, y: dimension.y},
{x: dimension.x+50, y: dimension.y+50},
{x: dimension.x+20, y: dimension.y+50}], 8);
      },
      arrowClick: function(event) {
        let dimension =
this.$refs.imageEditorObj.ej2Instances.getImageDimension();
        this.$refs.imageEditorObj.ej2Instances.drawArrow(dimension.x,
dimension.y+10, dimension.x+50, dimension.y+10, 10,);
      }
    }
  }
</script>
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-buttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-splitbuttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-lists/styles/material.css";
@import "../node_modules/@syncfusion/ej2-popups/styles/material.css";
@import "../node_modules/@syncfusion/ej2-inputs/styles/material.css";
@import "../node_modules/@syncfusion/ej2-navigations/styles/material.css";
@import "../node_modules/@syncfusion/ej2-dropdowns/styles/material.css";
@import "../node_modules/@syncfusion/ej2-image-editor/styles/material.css";
#image-editor {
  width: 550px !important;
  height: 350px !important;

```

```
}
</style>
```

```
{% previewsample "page.domainurl/code-snippet/image-editor/getting-started-cs3" %}
```

### Delete a shape

The [deleteShape](#) method in the Vue Image Editor allows you to remove a shape annotation from the image editor. To use this method, you need to pass the [shapeld](#) of the annotation as a parameter.

The [shapeld](#) is a unique identifier assigned to each shape annotation within the image editor. It serves as a reference to a specific annotation, enabling targeted deletion of the desired annotation. By specifying the [shapeld](#) associated with the shape annotation you want to remove, you can effectively delete it from the image editor.

To retrieve the inserted shape annotations, you can utilize the [getShapeSetting](#) method, which provides a collection of annotations represented by [ShapeSettings](#). This method allows you to access and work with the annotations that have been inserted into the image.

Here is an example of deleting rectangle, ellipse, arrow, path, and line in a button click event.

### APP.VUE

```
<template>
<div>
<ejs-imageeditor id="image-editor" ref="imageEditorObj" height="350px"
width="550px" :shapeChanging="shapeChanging"
:showQuickAccessToolbar=false></ejs-imageeditor>
<ejs-button cssClass="e-img-button" :isPrimary="true" v-
on:click.native="btnClick">Delete shape</ejs-button>
</div>
</template>
<script>
import Vue from 'vue';
import { ImageEditorPlugin } from "@syncfusion/ej2-vue-image-editor";
import { ButtonPlugin } from '@syncfusion/ej2-vue-buttons';
import { Browser } from "@syncfusion/ej2-base";
Vue.use(ImageEditorPlugin);
Vue.use(ButtonPlugin);
export default {
  data: function() {
    return {
      toolbar: ['Annotate', 'Rectangle', 'Ellipse', 'Line', 'Arrow',
'Path'],
      id: '',
    };
  },
  methods: {
    created: function() {
      if (Browser.isDevice) {
        this.$refs.imageEditorObj.open('flower.png');
      } else {
        this.$refs.imageEditorObj.open('bridge.png');
      }
    },
    shapeChanging: function(event) {
      if(event.action === 'select') {

```

```

        this.id = event.currentShapeSettings.id;
    },
    btnClick: function(event) {
        this.$refs.imageEditorObj.ej2Instances.deleteShape(this.id);
    }
}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-buttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-splitbuttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-lists/styles/material.css";
@import "../node_modules/@syncfusion/ej2-popups/styles/material.css";
@import "../node_modules/@syncfusion/ej2-inputs/styles/material.css";
@import "../node_modules/@syncfusion/ej2-navigations/styles/material.css";
@import "../node_modules/@syncfusion/ej2-dropdowns/styles/material.css";
@import "../node_modules/@syncfusion/ej2-image-editor/styles/material.css";
#image-editor {
    width: 550px !important;
    height: 350px !important;
}
</style>

```

{% previewsample "page.domainurl/code-snippet/image-editor/getting-started-cs24" %}

### Image annotation

The image annotation feature in the Image Editor provides the capability to add and customize images directly onto the image. With this feature, you can easily insert image or icons at specific locations within the image and customize various aspects of the image to meet your requirements. You have control over the customization options including rotate, flip, transparency for the image annotation.

#### Add an image annotation

The [drawImage](#) method serves the purpose of inserting an image into the Image Editor control, allowing for image annotations to be added. These image annotations can be used for various purposes, such as adding logos, watermarks, or decorative elements to the image.

The [drawImage](#) method in the Image Editor control takes six parameters to define the properties of the image annotation:

- data: Specified the image data or url of the image to be inserted.
- x: Specifies the x-coordinate of the top-left corner of the image.
- y: Specifies the y-coordinate of the top-left corner of the image.
- width: Specifies the width of the image.
- height: Specifies the height of the image.
- isAspectRatio: Specifies whether the image is rendered with aspect ratio or not.

In the following example, you can use the [drawImage](#) method in the button click event.

### APP.VUE

```

<template>
<div>

```

```

<ejs-imageeditor id="image-editor" ref="imageEditorObj" height="350px"
width="550px"></ejs-imageeditor>
<ejs-button cssClass="e-img-button" :isPrimary="true" v-
on:click.native="btnClick">Add Image</ejs-button>
</div>
</template>
<script>
import Vue from 'vue';
import { ImageEditorPlugin } from '@syncfusion/ej2-vue-image-editor';
import { ButtonPlugin } from '@syncfusion/ej2-vue-buttons';
import { Browser } from '@syncfusion/ej2-base';
Vue.use(ImageEditorPlugin);
Vue.use(ButtonPlugin);
export default {
  data: function() {
    return {
      toolbar: []
    };
  },
  methods: {
    created: function() {
      if (Browser.isDevice) {
        this.$refs.imageEditorObj.open('flower.png');
      } else {
        this.$refs.imageEditorObj.open('bridge.png');
      }
    },
    btnClick: function(event) {
      let dimension =
this.$refs.imageEditorObj.ej2Instances.getImageDimension();
      this.$refs.imageEditorObj.ej2Instances.drawImage('flower.png',
dimension.x, dimension.y, 100, 80, true, 0);
    }
  }
}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-buttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-splitbuttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-lists/styles/material.css";
@import "../node_modules/@syncfusion/ej2-popups/styles/material.css";
@import "../node_modules/@syncfusion/ej2-inputs/styles/material.css";
@import "../node_modules/@syncfusion/ej2-navigations/styles/material.css";
@import "../node_modules/@syncfusion/ej2-dropdowns/styles/material.css";
@import "../node_modules/@syncfusion/ej2-image-editor/styles/material.css";
#image-editor {
  width: 550px !important;
  height: 350px !important;
}
</style>

```

{% previewsample "page.domainurl/code-snippet/image-editor/getting-started-cs43" %}

## Transform in the vue Image Editor component

The Image Editor provides a range of transformation options for manipulating both the image and its annotations. These options include rotation, flipping, zooming, and panning. These transformations offer flexibility in adjusting the image and enhancing its visual appearance.

### Rotate an image

The Image Editor allows to rotate the image and its annotations by a specific number of degrees clockwise or anti-clockwise using [rotate](#) method. This method takes a single parameter: the angle of rotation in degrees. A positive value will rotate the image clockwise, while a negative value will rotate it anti-clockwise.

Here is an example of rotating an image in a button click event.

#### APP.VUE

```
<template>
<div>
<ejs-imageeditor id="image-editor" ref="imageEditorObj" height="350px"
width="550px"></ejs-imageeditor>
<ejs-button cssClass="e-img-button" :isPrimary="true" v-
on:click.native="btnClick">Rotate</ejs-button>
</div>
</template>
<script>
import Vue from 'vue';
import { ImageEditorPlugin } from "@syncfusion/ej2-vue-image-editor";
import { ButtonPlugin } from '@syncfusion/ej2-vue-buttons';
import { Browser } from "@syncfusion/ej2-base";
Vue.use(ImageEditorPlugin);
Vue.use(ButtonPlugin);
export default {
  data: function() {
    return {
      toolbar: []
    };
  },
  methods: {
    created: function() {
      if (Browser.isDevice) {
        this.$refs.imageEditorObj.open('flower.png');
      } else {
        this.$refs.imageEditorObj.open('bridge.png');
      }
    },
    btnClick: function(event) {
      this.$refs.imageEditorObj.ej2Instances.rotate(90);
    }
  }
}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-buttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-splitbuttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-lists/styles/material.css";
@import "../node_modules/@syncfusion/ej2-popups/styles/material.css";
```

```
@import "../node_modules/@syncfusion/ej2-inputs/styles/material.css";
@import "../node_modules/@syncfusion/ej2-navigations/styles/material.css";
@import "../node_modules/@syncfusion/ej2-dropdowns/styles/material.css";
@import "../node_modules/@syncfusion/ej2-image-editor/styles/material.css";
#image-editor {
    width: 550px !important;
    height: 350px !important;
}
</style>
```

{% previewsample "page.domainurl/code-snippet/image-editor/getting-started-cs19" %}

### Flip an image

The Image Editor provides the [flip](#) method, which allows you to flip both the image and its annotations either horizontally or vertically. This method takes a single parameter of type ImageEditorDirection, which specifies the direction in which the flip operation should be applied.

The [Direction](#) parameter accepts two values: 'Horizontal' and 'Vertical'. When you choose 'Horizontal', the image and annotations will be flipped along the horizontal axis, resulting in a mirror effect. On the other hand, selecting 'Vertical' will flip them along the vertical axis, producing a vertical mirror effect.

Here is an example of flipping an image in a button click event.

### APP.VUE

```
<template>
<div>
<ejs-imageeditor id="image-editor" ref="imageEditorObj" height="350px"
width="550px"></ejs-imageeditor>
<ejs-button cssClass="e-img-button" :isPrimary="true" v-
on:click.native="btnClick">Flip</ejs-button>
</div>
</template>
<script>
import Vue from 'vue';
import { ImageEditorPlugin } from "@syncfusion/ej2-vue-image-editor";
import { ButtonPlugin } from "@syncfusion/ej2-vue-buttons";
import { Browser } from "@syncfusion/ej2-base";
Vue.use(ImageEditorPlugin);
Vue.use(ButtonPlugin);
export default {
    data: function() {
        return {
            toolbar: []
        };
    },
    methods: {
        created: function() {
            if (Browser.isDevice) {
                this.$refs.imageEditorObj.open('flower.png');
            } else {
                this.$refs.imageEditorObj.open('bridge.png');
            }
        },
        btnClick: function(event) {
```

```

        this.$refs.imageEditorObj.ej2Instances.flip("Horizontal"); //
        Horizontal flip
    }
}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-buttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-splitbuttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-lists/styles/material.css";
@import "../node_modules/@syncfusion/ej2-popups/styles/material.css";
@import "../node_modules/@syncfusion/ej2-inputs/styles/material.css";
@import "../node_modules/@syncfusion/ej2-navigations/styles/material.css";
@import "../node_modules/@syncfusion/ej2-dropdowns/styles/material.css";
@import "../node_modules/@syncfusion/ej2-image-editor/styles/material.css";
#image-editor {
    width: 550px !important;
    height: 350px !important;
}
</style>

```

{% previewsample "page.domainurl/code-snippet/image-editor/getting-started-cs20" %}

### Zoom in or out an image

The Image Editor allows to magnify an image using the [zoom](#) method. This method allows one to zoom in and out of the image and provides a more detailed view of the image's hidden areas. This method takes two parameters to perform zooming.

- zoomFactor - Specifies a value to controlling the level of magnification applied to the image.
- zoomPoint - Specifies x and y coordinates of a point as ImageEditorPoint on image to perform zooming.

Here is an example of zooming an image in a button click event.

### APP.VUE

```

<template>
<div>
<ejs-imageeditor id="image-editor" ref="imageEditorObj" height="350px"
width="550px" :zoomSettings="zoomSettings"></ejs-imageeditor>
<ejs-button cssClass="e-img-button" :isPrimary="true" v-
on:click.native="btn1Click">Zoom in</ejs-button>
<ejs-button cssClass="e-img-button" :isPrimary="true" v-
on:click.native="btn2Click">Zoom out</ejs-button>
</div>
</template>
<script>
import Vue from 'vue';
import { ImageEditorPlugin } from "@syncfusion/ej2-vue-image-editor";
import { ButtonPlugin } from '@syncfusion/ej2-vue-buttons';
import { Browser } from "@syncfusion/ej2-base";
Vue.use(ImageEditorPlugin);
Vue.use(ButtonPlugin);
export default {

```

```

data: function() {
  return {
    toolbar: [],
    zoomSettings: {maxZoomFactor: 30, minZoomFactor: 0.1},
    zoomLevel: 1
  };
},
methods: {
  created: function() {
    if (Browser.isDevice) {
      this.$refs.imageEditorObj.open('flower.png');
    } else {
      this.$refs.imageEditorObj.open('bridge.png');
    }
  },
  btn1Click: function(event) {
    if (this.zoomLevel < 1) {
      this.zoomLevel += 0.1;
    } else {
      this.zoomLevel += 1;
    }
    if (this.zoomLevel >
this.$refs.imageEditorObj.zoomSettings.maxZoomFactor) {
      this.zoomLevel =
this.$refs.imageEditorObj.zoomSettings.maxZoomFactor;
    }
    this.$refs.imageEditorObj.ej2Instances.zoom(this.zoomLevel); // Zoom
in
  },
  btn2Click: function(event) {
    if (this.zoomLevel <= 1) {
      this.zoomLevel -= 0.1;
    } else {
      this.zoomLevel -= 1;
    }
    if (this.zoomLevel <
this.$refs.imageEditorObj.zoomSettings.minZoomFactor) {
      this.zoomLevel =
this.$refs.imageEditorObj.zoomSettings.minZoomFactor;
    }
    this.$refs.imageEditorObj.ej2Instances.zoom(this.zoomLevel); // Zoom
out
  }
}
}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-buttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-splitbuttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-lists/styles/material.css";
@import "../node_modules/@syncfusion/ej2-popups/styles/material.css";
@import "../node_modules/@syncfusion/ej2-inputs/styles/material.css";
@import "../node_modules/@syncfusion/ej2-navigations/styles/material.css";
@import "../node_modules/@syncfusion/ej2-dropdowns/styles/material.css";
@import "../node_modules/@syncfusion/ej2-image-editor/styles/material.css";
#image-editor {

```



```
width: 550px !important;
height: 350px !important;
}
</style>
```

{% previewsample "page.domainurl/code-snippet/image-editor/getting-started-cs21" %}

#### *Maximum and Minimum zoom level*

The [maxZoomFactor](#) property is a useful feature in the Image Editor that allows you to define the maximum level of zoom permitted for an image. This property sets a limit on how much the image can be magnified, preventing excessive zooming that may result in a loss of image quality or visibility.

By default, the [minZoomFactor](#) value is set to 10, meaning that the image can be zoomed in up to 10 times its original size. This ensures that the zooming functionality remains within reasonable bounds and maintains the integrity of the image.

The [maxZoomFactor](#) property allows you to specify the minimum level of zoom that is allowed for an image. By setting this property, you can prevent the image from being zoomed out beyond a certain point, ensuring that it remains visible and usable even at the smallest zoom level.

By default, the [maxZoomFactor](#) value is set to 0.1, meaning that the image can be zoomed out up to 10 times its original size.

Here is an example of specifying [minZoomFactor](#) and [maxZoomFactor](#) property in [zoomSettings](#) options in an image editor.

#### *Panning an image*

The Image Editor allows to pan an image when the image exceeds the canvas size or selection range. When zooming in on an image or applying a selection for cropping, it is common for the image to exceed the size of the canvas or exceed the selection range. So, the panning is used to view the entire image, by clicking on the canvas and dragging it in the direction they want to move.

#### *Panning event*

The [panning](#) event is activated when the user begins dragging the image within the canvas. This event provide an opportunity to perform specific actions, like adjusting the position of an image, in response to the gesture of panning. And these event uses [panEventArgs](#) to handle the panning action when the user starts dragging the image.

The parameter available in the [panEventArgs](#) events are,

- [PanEventArgs.startPoint](#) - The x and y coordinates as [ImageEditorPoint](#) for the start point.
- [PanEventArgs.endpoint](#) - The x and y coordinates as [ImageEditorPoint](#) for the end point.
- [PanEventArgs.cancel](#) – Specifies the boolean value to cancel the panning action.

#### **APP.VUE**

```
<template>
<div>
<ejs-imageeditor id="image-editor" ref="imageEditorObj" height="350px"
width="550px"></ejs-imageeditor>
<ejs-button cssClass="e-img-button" :isPrimary="true" v-
on:click.native="btnClick">Zoom & Pan</ejs-button>
</div>
</template>
```

```

<script>
import Vue from 'vue';
import { ImageEditorPlugin } from "@syncfusion/ej2-vue-image-editor";
import { ButtonPlugin } from '@syncfusion/ej2-vue-buttons';
import { Browser } from "@syncfusion/ej2-base";
Vue.use(ImageEditorPlugin);
Vue.use(ButtonPlugin);
export default {
  data: function() {
    return {
      toolbar: [],
      zoomLevel: 1
    };
  },
  methods: {
    created: function() {
      if (Browser.isDevice) {
        this.$refs.imageEditorObj.open('flower.png');
      } else {
        this.$refs.imageEditorObj.open('bridge.png');
      }
    },
    btnClick: function(event) {
      this.$refs.imageEditorObj.ej2Instances.zoom(this.zoomLevel); // Zoom
      in this.$refs.imageEditorObj.ej2Instances.pan(true);
    }
  }
}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-buttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-splitbuttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-lists/styles/material.css";
@import "../node_modules/@syncfusion/ej2-popups/styles/material.css";
@import "../node_modules/@syncfusion/ej2-inputs/styles/material.css";
@import "../node_modules/@syncfusion/ej2-navigations/styles/material.css";
@import "../node_modules/@syncfusion/ej2-dropdowns/styles/material.css";
@import "../node_modules/@syncfusion/ej2-image-editor/styles/material.css";
#image-editor {
  width: 550px !important;
  height: 350px !important;
}
</style>

```

{% previewsample "page.domainurl/code-snippet/image-editor/getting-started-cs23" %}

### Zooming event

The [zooming](#) event is triggered when performing zooming the image. This event can be used to perform certain actions, such as updating the position of the image. This event is passed an object that contains information about the zooming event, such as the amount of zooming performed. And this event uses [ZoomEventArgs](#) to handle the zooming action in the image.

The parameter available in the Zooming event is,

- `ZoomEventArgs.zoomPoint` - The x and y coordinates as `ImageEditorPoint` for the zoom point.
- `ZoomEventArgs.previousZoomFactor` - The previous zoom factor applied in the image editor.
- `ZoomEventArgs.currentZoomFactor` - The current zoom factor to be applied in the image editor.
- `ZoomEventArgs.cancel` – Specify a boolean value to cancel the zooming action.
- `ZoomEventArgs.zoomTrigger` - The type of zooming performed in the image editor.

### Rotating event

The [rotating](#) event is triggered when performing rotating the image. This event is passed an object that contains information about the rotating event, such as the amount of rotation performed. And this event uses [RotateEventArgs](#) to handle the rotating action in the image.

The parameter available in the Rotating event is,

- `RotateEventArgs.previousDegree`: The degree of rotation before the recent rotation action was applied in the Image Editor.
- `RotateEventArgs.currentDegree`: The current degree of rotation after the rotation action has been performed in the Image Editor.

`RotateEventArgs.cancel` – Specifies a boolean value to cancel the rotating action.

### Flipping event

The [flipping](#) event is triggered when performing flipping the image. This event is passed an object that contains information about the flipping event, such as the amount of flip performed. And this event uses [FlipEventArgs](#) to handle the flipping action in the image.

The parameter available in the [flipping](#) event is,

- `FlipEventArgs.direction` - The flip direction as `ImageEditorDirection` to be applied in the image editor.
- `FlipEventArgs.cancel` - Specifies a boolean value to cancel the flip action.

## Toolbar in the Vue Image Editor component

The toolbars in the Image Editor are a key component for interacting with and editing images. They provide a range of tools and options that can be customized to suit the needs and preferences. Add or remove items from the toolbar to create a personalized set of tools, or they can even create their own custom toolbar from scratch. This flexibility and customization allow them to create a unique image editing experience that is tailored to their specific needs and workflow.

In the Image Editor, the [toolbar](#) property provides the ability to customize the toolbar by adding or removing items, as well as defining a completely custom toolbar. This feature is valuable for creating a personalized image editing experience that aligns with specific requirements and workflows.

### Built-in toolbar items

Specifies the toolbar items to perform UI interactions. Refer to the built-in toolbar items for the default value.

- Crop
- Transform
- Annotate
- ZoomIn

- ZoomOut
- Open
- Reset
- Save
- Pan

### Add a custom toolbar items

Users can define their own toolbars for an image editor by customizing the items or the entire toolbar. Users can achieve this by using the [toolbar](#) property.

The built-in toolbar can be customized using the [toolbar](#) property, so the specified toolbar items can be enabled in the Image Editor toolbar. And the contextual toolbar which is enabled while inserting annotations can also be customized in the [toolbarUpdating](#) event.

### APP.VUE

```
<template>
<div>
<ejs-imageeditor id="image-editor" ref="imageEditorObj" height="350px"
width="550px"></ejs-imageeditor>
</div>
</template>
<script>
import Vue from 'vue';
import { ImageEditorPlugin } from "@syncfusion/ej2-vue-image-editor";
Vue.use(ImageEditorPlugin);
export default {
  data: function() {
    return {
      toolbar: ['Crop', 'ZoomIn', 'ZoomOut', {text: 'Custom'}]
    };
  }
}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-buttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-splitbuttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-lists/styles/material.css";
@import "../node_modules/@syncfusion/ej2-popups/styles/material.css";
@import "../node_modules/@syncfusion/ej2-inputs/styles/material.css";
@import "../node_modules/@syncfusion/ej2-navigations/styles/material.css";
@import "../node_modules/@syncfusion/ej2-dropdowns/styles/material.css";
@import "../node_modules/@syncfusion/ej2-image-editor/styles/material.css";
#image-editor {
  width: 550px !important;
  height: 350px !important;
}
</style>
```

{% previewsample "page.domainurl/code-snippet/image-editor/getting-started-cs16" %}

### Show or hide a toolbar

The [toolbar](#) property controls the visibility of the toolbar in the Image Editor. When the [toolbar](#) property is set to an empty list, the toolbar is hidden. Conversely, if the [toolbar](#) property contains a list of items, the toolbar is shown, displaying the specified items. This feature provides flexibility for users to personalize their image editing experience.

Here is an example of hiding the toolbar of the image editor using [toolbar](#) property.

#### APP.VUE

```
<template>
<div>
<ejs-imageeditor id="image-editor" ref="imageEditorObj" height="350px"
width="550px"></ejs-imageeditor>
</div>
</template>
<script>
import Vue from 'vue';
import { ImageEditorPlugin, ImageFinetuneOption } from "@syncfusion/ej2-vue-image-editor";
import { ButtonPlugin } from '@syncfusion/ej2-vue-buttons';
import { Browser } from "@syncfusion/ej2-base";
Vue.use(ImageEditorPlugin);
Vue.use(ButtonPlugin);
Vue.use(ImageFinetuneOption);
export default {
  data: function() {
    return {
      toolbar: []
    };
  },
  methods: {
    created: function() {
      if (Browser.isDevice) {
        this.$refs.imageEditorObj.open('flower.png');
      } else {
        this.$refs.imageEditorObj.open('bridge.png');
      }
    }
  }
}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-buttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-splitbuttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-lists/styles/material.css";
@import "../node_modules/@syncfusion/ej2-popups/styles/material.css";
@import "../node_modules/@syncfusion/ej2-inputs/styles/material.css";
@import "../node_modules/@syncfusion/ej2-navigations/styles/material.css";
@import "../node_modules/@syncfusion/ej2-dropdowns/styles/material.css";
@import "../node_modules/@syncfusion/ej2-image-editor/styles/material.css";
#image-editor {
  width: 550px !important;
  height: 350px !important;
}
</style>
```

```
{% previewsample "page.domainurl/code-snippet/image-editor/getting-started-cs35" %}
```

### Enable or disable a toolbar item

The [toolbar](#) property is employed to enable or disable toolbar items in the Image Editor. By default, the [toolbar](#) property includes the default toolbar items, and these items cannot be disabled. However, if you have defined custom toolbar items using the `toolbarItemModel`, you can enable or disable them by configuring their respective properties within the [toolbar](#) property. This provides the flexibility to control the availability and functionality of custom toolbar items based on your specific requirements.

Here is an example of disabling the custom toolbar item using [toolbar](#) property.

### Enable or disable a contextual toolbar item

The `toolbarItems` property in the `toolbarEventArgs` is used to enable or disable contextual toolbar items in the Image Editor. To enable or disable the default toolbar items, you can accomplish this by setting the `Disabled` property to `true` in the `ImageEditorToolbarItemModel` within the `ToolbarItems` property. This allows you to selectively enable or disable specific default toolbar items based on your requirements, providing a customized toolbar experience in the Image Editor.

### Show or hide a toolbar item

The [toolbar](#) property controls the visibility of the toolbar in the Image Editor. When the [toolbar](#) property is set to an empty list, the toolbar is hidden. Conversely, if the [toolbar](#) property contains a list of items, the toolbar is shown, displaying the specified items. This feature provides flexibility for users to personalize their image editing experience.

Here is an example of hiding the toolbar of the image editor using [toolbar](#) property.

## APP.VUE

```
<template>
<div>
<ejs-imageeditor id="image-editor" ref="imageEditorObj" height="350px"
width="550px"></ejs-imageeditor>
</div>
</template>
<script>
import Vue from 'vue';
import { ImageEditorPlugin, ImageFinetuneOption } from "@syncfusion/ej2-vue-image-editor";
import { ButtonPlugin } from '@syncfusion/ej2-vue-buttons';
import { Browser } from "@syncfusion/ej2-base";
Vue.use(ImageEditorPlugin);
Vue.use(ButtonPlugin);
Vue.use(ImageFinetuneOption);
export default {
  data: function() {
    return {
      toolbar: ['Annotate', 'Finetune', 'Filter', 'Confirm', 'Reset', 'Save', 'ZoomIn', 'ZoomOut']
    };
  },
  methods: {
    created: function() {
      if (Browser.isDevice) {
        this.$refs.imageEditorObj.open('flower.png');
      }
    }
  }
}
```

```

    } else {
      this.$refs.imageEditorObj.open('bridge.png');
    }
  }
}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-buttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-splitbuttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-lists/styles/material.css";
@import "../node_modules/@syncfusion/ej2-popups/styles/material.css";
@import "../node_modules/@syncfusion/ej2-inputs/styles/material.css";
@import "../node_modules/@syncfusion/ej2-navigations/styles/material.css";
@import "../node_modules/@syncfusion/ej2-dropdowns/styles/material.css";
@import "../node_modules/@syncfusion/ej2-image-editor/styles/material.css";
#image-editor {
  width: 550px !important;
  height: 350px !important;
}
</style>

```

{% previewsample "page.domainurl/code-snippet/image-editor/getting-started-cs36" %}

### Toolbar template

The [toolbarTemplate](#) property in the Image Editor provides the capability to fully customize the toolbar by supplying a custom template. This feature is valuable when you want to create a distinct and personalized image editing experience that goes beyond the default toolbar or the customizable toolbar options offered by the Image Editor. By defining a custom template for the toolbar, you have complete control over its layout, appearance, and functionality. This empowers you to design a unique and tailored toolbar that aligns perfectly with your specific requirements and desired user experience.

Here is an example of using [toolbarTemplate](#) to render only the button to toggle the freehand draw option.

The toolbar of the Image Editor can be replaced with the user specific UI using the [toolbarTemplate](#) property.

### APP.VUE

```

<template>
<div>
<ejs-imageeditor id="image-editor" ref="imageEditorObj" height="350px"
width="550px" :created="created" :toolbarTemplate="toolbarTemplate">
</ejs-imageeditor>
</div>
</template>
<script>
import Vue from 'vue';
import { ImageEditorPlugin } from "@syncfusion/ej2-vue-image-editor";
import { ButtonPlugin } from '@syncfusion/ej2-vue-buttons';
import { Browser, getComponent } from "@syncfusion/ej2-base";
Vue.use(ImageEditorPlugin);
Vue.use(ButtonPlugin);
export default {

```

```

    data: function() {
      return {
        toolbarTemplate: () => {
          return {
            template : Vue.component('headerTemplate', {
              template:
                `<ejs-button :isPrimary="true" v-
on:click.native='btnClick'>Custom</ejs-button>`,
              data(args) {
                return {
                  imageEditorObj:
getComponent(document.getElementById('image-editor'), 'image-editor'),
                }
              },
              methods: {
                btnClick: function(){
                  this.imageEditorObj.freeHandDraw(true);
                }
              }
            })
          }
        },
      };
    },
    methods: {
      created: function() {
        if (Browser.isDevice) {
          this.$refs.imageEditorObj.open('flower.png');
        } else {
          this.$refs.imageEditorObj.open('bridge.png');
        }
      }
    }
  }
</script>
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-buttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-splitbuttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-lists/styles/material.css";
@import "../node_modules/@syncfusion/ej2-popups/styles/material.css";
@import "../node_modules/@syncfusion/ej2-inputs/styles/material.css";
@import "../node_modules/@syncfusion/ej2-navigations/styles/material.css";
@import "../node_modules/@syncfusion/ej2-dropdowns/styles/material.css";
@import "../node_modules/@syncfusion/ej2-image-editor/styles/material.css";
#image-editor {
  width: 550px !important;
  height: 350px !important;
}
</style>

```

{% previewsample "page.domainurl/code-snippet/image-editor/getting-started-cs17" %}



### Customize contextual toolbar

The [toolbarUpdating](#) event is triggered when inserting or selecting annotations, which opens the contextual toolbar in the Image Editor. Within this event, the [toolbarItems](#) property in the [ToolbarEventArgs](#) is utilized to add or remove contextual toolbar items.

In the following example, the contextual toolbar for rectangle will be rendered with only stroke color by excluding fill color and stroke width using [toolbarUpdating](#) event.

#### APP.VUE

```
<template>
<div>
<ejs-imageeditor id="image-editor" ref="imageEditorObj" height="350px"
width="550px" :toolbarUpdating="toolbarUpdating"></ejs-imageeditor>
</div>
</template>
<script>
import Vue from 'vue';
import { ImageEditorPlugin } from "@syncfusion/ej2-vue-image-editor";
import { Browser } from "@syncfusion/ej2-base";
Vue.use(ImageEditorPlugin);
export default {
  data: function() {
    return {};
  },
  methods: {
    created: function() {
      if (Browser.isDevice) {
        this.$refs.imageEditorObj.open('flower.png');
      } else {
        this.$refs.imageEditorObj.open('bridge.png');
      }
    },
    toolbarUpdating: function(args) {
      if (args.toolbarType === 'shapes') {
        args.toolbarItems = ['strokeColor'];
      }
    }
  }
}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-buttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-splitbuttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-lists/styles/material.css";
@import "../node_modules/@syncfusion/ej2-popups/styles/material.css";
@import "../node_modules/@syncfusion/ej2-inputs/styles/material.css";
@import "../node_modules/@syncfusion/ej2-navigations/styles/material.css";
@import "../node_modules/@syncfusion/ej2-dropdowns/styles/material.css";
@import "../node_modules/@syncfusion/ej2-image-editor/styles/material.css";
#image-editor {
  width: 550px !important;
  height: 350px !important;
}
</style>
```

```
{% previewsample "page.domainurl/code-snippet/image-editor/getting-started-cs18" %}
```

### Toolbar item clicked event

The [toolbarItemClicked](#) event is triggered when a toolbar item is clicked in the Image Editor. This event is particularly useful when you have added custom options to both the main toolbar and contextual toolbar, as it allows you to capture the user's interaction with those custom options. By subscribing to the [toolbarItemClicked](#) event, you can execute specific actions or handle logic based on the toolbar item that was clicked.

### APP.VUE

```
<template>
<div>
<ejs-imageeditor id="image-editor" ref="imageEditorObj" height="350px"
width="550px" :toolbarItemClicked="toolbarItemClicked"></ejs-imageeditor>
</div>
</template>
<script>
import Vue from 'vue';
import { ImageEditorPlugin } from "@syncfusion/ej2-vue-image-editor";
import { ButtonPlugin } from '@syncfusion/ej2-vue-buttons';
import { Browser } from "@syncfusion/ej2-base";
Vue.use(ImageEditorPlugin);
export default {
  data: function() {
    return {
      toolbar: [{text: 'Custom'}]
    };
  },
  methods: {
    created: function() {
      if (Browser.isDevice) {
        this.$refs.imageEditorObj.open('flower.png');
      } else {
        this.$refs.imageEditorObj.open('bridge.png');
      }
    },
    toolbarItemClicked: function(args) {
      if(args.item.text === 'Custom') {
        this.$refs.imageEditorObj.rotate(90);
      }
    }
  }
}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-buttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-splitbuttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-lists/styles/material.css";
@import "../node_modules/@syncfusion/ej2-popups/styles/material.css";
@import "../node_modules/@syncfusion/ej2-inputs/styles/material.css";
@import "../node_modules/@syncfusion/ej2-navigations/styles/material.css";
@import "../node_modules/@syncfusion/ej2-dropdowns/styles/material.css";
@import "../node_modules/@syncfusion/ej2-image-editor/styles/material.css";
```

```
#image-editor {  
  width: 550px !important;  
  height: 350px !important;  
}  
</style>
```

{% previewsample "page.domainurl/code-snippet/image-editor/getting-started-cs40" %}

### Toolbar created event

The [toolbarCreated](#) event is triggered after the toolbar is created in the Image Editor. This event can be useful when you need to perform any actions or make modifications to the toolbar once it is fully initialized and ready for interaction. By subscribing to the [toolbarCreated](#) event, you can access the toolbar object and perform tasks such as adding event handlers, customizing the appearance, or configuring additional functionality.

### Add an additional contextual Toolbar item to text shape

The contextual toolbar that appears when inserting annotations in the Image Editor is customizable using the [toolbarUpdating](#) event. This event is triggered when the contextual toolbar is rendered, allowing you to modify its contents. To add additional toolbar items to the contextual toolbar, you can access the [toolbarItems](#) property of the object within the event handler. By adding or removing items from the [toolbarItems](#) property based on the Item property, you can customize the options available in the contextual toolbar according to your needs. This gives you the ability to extend the functionality of the contextual toolbar and provide additional tools and options for working with inserted annotations.

Here is an example of adding the custom toolbar item to the contextual toolbar.

### APP.VUE

```
<template>  
<div>  
<ejs-imageeditor id="image-editor" ref="imageEditorObj" height="350px"  
width="550px" :toolbarUpdating="toolbarUpdating"></ejs-imageeditor>  
</div>  
</template>  
<script>  
import Vue from 'vue';  
import { ImageEditorPlugin } from "@syncfusion/ej2-vue-image-editor";  
import { Browser } from "@syncfusion/ej2-base";  
Vue.use(ImageEditorPlugin);  
export default {  
  data: function() {  
    return {};  
  },  
  methods: {  
    created: function() {  
      if (Browser.isDevice) {  
        this.$refs.imageEditorObj.open('flower.png');  
      } else {  
        this.$refs.imageEditorObj.open('bridge.png');  
      }  
    },  
    toolbarUpdating: function(args) {  
      if (args.toolbarType === 'text') {  
        args.toolbarItems.push({text: 'custom'})  
      }  
    }  
  }  
}
```

```

    }
  }
}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-buttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-splitbuttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-lists/styles/material.css";
@import "../node_modules/@syncfusion/ej2-popups/styles/material.css";
@import "../node_modules/@syncfusion/ej2-inputs/styles/material.css";
@import "../node_modules/@syncfusion/ej2-navigations/styles/material.css";
@import "../node_modules/@syncfusion/ej2-dropdowns/styles/material.css";
@import "../node_modules/@syncfusion/ej2-image-editor/styles/material.css";
#image-editor {
  width: 550px !important;
  height: 350px !important;
}
</style>

```

{% previewsample "page.domainurl/code-snippet/image-editor/getting-started-cs4" %}

## Quick access toolbar in the Vue Image Editor component

The quick access toolbars in the Vue Image Editor play a vital role in facilitating interactions with annotations like Rectangle, Ellipse, Line, Arrow, and Path. These toolbars offer a diverse array of tools and options that can be tailored to match the specific requirements and preferences associated with each annotation type. The toolbar is only displayed when an annotation is selected, ensuring a focused and contextual user experience. Users have the flexibility to add or remove items from the toolbar, allowing them to create a personalized set of tools. Additionally, users can also build a completely custom toolbar from the ground up, providing them with complete control over the available options and functionality.

### Add a custom toolbar item

The quick access toolbar that appears when inserting annotations in the Image Editor is customizable using the [quickAccessToolbarOpen](#) event. This event is triggered when the quick access toolbar is opened, allowing you to modify its contents. To add additional toolbar items to the quick access toolbar, you can access the `toolbarItems` property of the `QuickAccessToolbarEventArgs` within the event handler. By adding or removing items from the `toolbarItems` property based on the item property, you can customize the options available in the quick access toolbar according to your needs. This gives you the ability to extend the functionality of the quick access toolbar and provide additional tools and options for working with inserted annotations.

Here is an example of adding the custom toolbar item to the quick access toolbar.

### APP.VUE

```

<template>
<div>
<ejs-imageeditor id="image-editor" ref="imageEditorObj" height="350px"
width="550px" :quickAccessToolbarOpen="quickAccessToolbarOpen"></ejs-
imageeditor>
</div>
</template>
<script>

```

```

import Vue from 'vue';
import { ImageEditorPlugin, ImageFinetuneOption } from "@syncfusion/ej2-vue-image-editor";
import { ButtonPlugin } from '@syncfusion/ej2-vue-buttons';
import { Browser } from "@syncfusion/ej2-base";
Vue.use(ImageEditorPlugin);
Vue.use(ButtonPlugin);
Vue.use(ImageFinetuneOption);
export default {
  data: function() {
    return {};
  },
  methods: {
    created: function() {
      if (Browser.isDevice) {
        this.$refs.imageEditorObj.open('flower.png');
      } else {
        this.$refs.imageEditorObj.open('bridge.png');
      }
    },
    quickAccessToolbarOpen: function(args) {
      args.toolbarItems = ['Clone'];
    }
  }
}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-buttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-splitbuttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-lists/styles/material.css";
@import "../node_modules/@syncfusion/ej2-popups/styles/material.css";
@import "../node_modules/@syncfusion/ej2-inputs/styles/material.css";
@import "../node_modules/@syncfusion/ej2-navigations/styles/material.css";
@import "../node_modules/@syncfusion/ej2-dropdowns/styles/material.css";
@import "../node_modules/@syncfusion/ej2-image-editor/styles/material.css";
#image-editor {
  width: 550px !important;
  height: 350px !important;
}
</style>

```

{% previewsample "page.domainurl/code-snippet/image-editor/getting-started-cs32" %}

## Undo and redo actions in the Vue Image Editor

The undo and redo functionalities provide a way to reverse and repeat editing actions performed on an image. These features are essential for maintaining control and flexibility during the editing process.

In an image editor, the undo and redo history typically have a limited capacity, and the number of steps that can be stored is 16 steps, meaning that the editor keeps track of the most recent 16 actions performed on the image. Once the history reaches its maximum capacity, any new actions beyond the 16th step will result in the removal of the oldest action from the history.

### Undo the action

The undo action in an image editor allows users to revert the most recent editing action or a series of actions back to their previous state. When the undo command is triggered, the image editor undoes the last applied modification, effectively restoring the image to its state before the action was performed. The undo action is useful for correcting mistakes, removing unwanted changes, or exploring different editing options without permanently altering the image.

### Redo the action

The Redo action in an image editor allows users to reapply previously undone actions or modifications to the image. When the redo command is triggered, the image editor reapplies the last action that was undone, bringing the image back to the state it was in after the action was initially applied. The redo is useful when users want to repeat an action that was previously undone or restore changes that were temporarily reversed.

In the following example, the [undo](#) and [redo](#) method is used in the button click event.

#### APP.VUE

```
<template>
<div>
<ejs-imageeditor id="image-editor" ref="imageEditorObj" height="350px"
width="550px"></ejs-imageeditor>
<ejs-button cssClass="e-img-button" :isPrimary="true" v-
on:click.native="btnClick">Text</ejs-button>
<ejs-button cssClass="e-img-button" :isPrimary="true" v-
on:click.native="undoClick">Undo</ejs-button>
<ejs-button cssClass="e-img-button" :isPrimary="true" v-
on:click.native="redoClick">Redo</ejs-button>
</div>
</template>
<script>
import Vue from 'vue';
import { ImageEditorPlugin } from "@syncfusion/ej2-vue-image-editor";
import { ButtonPlugin } from '@syncfusion/ej2-vue-buttons';
import { Browser } from "@syncfusion/ej2-base";
Vue.use(ImageEditorPlugin);
Vue.use(ButtonPlugin);
export default {
  data: function() {
    return {
      toolbar: []
    };
  },
  methods: {
    created: function() {
      if (Browser.isDevice) {
        this.$refs.imageEditorObj.open('flower.png');
      } else {
        this.$refs.imageEditorObj.open('bridge.png');
      }
    },
    btnClick: function(event) {
      let dimension =
this.$refs.imageEditorObj.ej2Instances.getImageDimension();
```

```

    this.$refs.imageEditorObj.ej2Instances.drawText (dimension.x,dimension.y, 'Enter\nText');
  },
  undoClick: function(event) {
    this.$refs.imageEditorObj.ej2Instances.undo();
  },
  redoClick: function(event) {
    this.$refs.imageEditorObj.ej2Instances.redo();
  }
}
}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-buttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-splitbuttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-lists/styles/material.css";
@import "../node_modules/@syncfusion/ej2-popups/styles/material.css";
@import "../node_modules/@syncfusion/ej2-inputs/styles/material.css";
@import "../node_modules/@syncfusion/ej2-navigations/styles/material.css";
@import "../node_modules/@syncfusion/ej2-dropdowns/styles/material.css";
@import "../node_modules/@syncfusion/ej2-image-editor/styles/material.css";
#image-editor {
  width: 550px !important;
  height: 350px !important;
}
</style>

```

{% previewsample "page.domainurl/code-snippet/image-editor/getting-started-cs30" %}

## Filters in the Vue Image Editor component

Filters are pre-defined effects that can be applied to an image to alter its appearance or mood. Image filters can be used to add visual interest or to enhance certain features of the image. Some common types of image filters include cold, warm, chrome, sepia, and invert. This can be done by either using the toolbar or the [applyImageFilter](#) method which takes a single parameter: the filter applied to an image.

### Apply filter effect

The [applyImageFilter](#) method is utilized to apply filters to an image. By passing the desired filter type as the first parameter of the method, specified as [ImageFilterOption](#) the method applies the corresponding filter to the image. This allows for easy and convenient application of various filters to enhance or modify the image based on the chosen filter type.

- filterOption - Specifies the filter options to the image.

In the following example, you can using the applyImageFilter method in the button click event.

### APP.VUE

```

<template>
<div>
<ejs-imageeditor id="image-editor" ref="imageEditorObj" height="350px"
width="550px"></ejs-imageeditor>

```

```

<ejs-button cssClass="e-img-button" :isPrimary="true" v-
on:click.native="chromeClick">Chrome</ejs-button>
<ejs-button cssClass="e-img-button" :isPrimary="true" v-
on:click.native="coldClick">Cold</ejs-button>
<ejs-button cssClass="e-img-button" :isPrimary="true" v-
on:click.native="warmClick">Warm</ejs-button>
<ejs-button cssClass="e-img-button" :isPrimary="true" v-
on:click.native="grayScaleClick">GrayScale</ejs-button>
<ejs-button cssClass="e-img-button" :isPrimary="true" v-
on:click.native="sepiaClick">Sepia</ejs-button>
<ejs-button cssClass="e-img-button" :isPrimary="true" v-
on:click.native="invertClick">Invert</ejs-button>
</div>
</template>
<script>
import Vue from 'vue';
import { ImageEditorPlugin, ImageFilterOption } from "@syncfusion/ej2-vue-
image-editor";
import { ButtonPlugin } from '@syncfusion/ej2-vue-buttons';
import { Browser } from "@syncfusion/ej2-base";
Vue.use(ImageEditorPlugin);
Vue.use(ButtonPlugin);
Vue.use(ImageFilterOption);
export default {
  data: function() {
    return {
      toolbar: []
    };
  },
  methods: {
    created: function() {
      if (Browser.isDevice) {
        this.$refs.imageEditorObj.open('flower.png');
      } else {
        this.$refs.imageEditorObj.open('bridge.png');
      }
    },
    chromeClick: function(event) {
      this.$refs.imageEditorObj.ej2Instances.applyImageFilter('Chrome');
    },
    coldClick: function(event) {
      this.$refs.imageEditorObj.ej2Instances.applyImageFilter('Cold');
    },
    warmClick: function(event) {
      this.$refs.imageEditorObj.ej2Instances.applyImageFilter('Warm');
    },
    grayScaleClick: function(event) {
      this.$refs.imageEditorObj.ej2Instances.applyImageFilter('GrayScale');
    },
    sepiaClick: function(event) {
      this.$refs.imageEditorObj.ej2Instances.applyImageFilter('Sepia');
    },
    invertClick: function(event) {
      this.$refs.imageEditorObj.ej2Instances.applyImageFilter('Invert');
    }
  }
}

```



```

</script>
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-buttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-splitbuttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-lists/styles/material.css";
@import "../node_modules/@syncfusion/ej2-popups/styles/material.css";
@import "../node_modules/@syncfusion/ej2-inputs/styles/material.css";
@import "../node_modules/@syncfusion/ej2-navigations/styles/material.css";
@import "../node_modules/@syncfusion/ej2-dropdowns/styles/material.css";
@import "../node_modules/@syncfusion/ej2-image-editor/styles/material.css";
#image-editor {
    width: 550px !important;
    height: 350px !important;
}
</style>

```

{% previewsample "page.domainurl/code-snippet/image-editor/getting-started-cs26" %}

### Image filtering event

The [imageFiltering](#) event is triggered when applying filtering on the image. This event is passed an object that contains information about the filtering event, such as the type of filtering.

The parameter available in the [ImageFilterEventArgs](#) event is,

[ImageFilterEventArgs.Filter](#) - The type of filtering as [ImageFilterOption](#) to be applied in the image editor.

[ImageFilterEventArgs.Cancel](#) – Specifies to cancel the filtering action.

### Finetune in the Vue Image Editor component

Fine-tuning involves making precise adjustments to the settings of an image filter in order to achieve a specific desired effect. It provides control over the intensity and specific aspects of the filter's impact on the image. For example, fine-tuning allows you to modify parameters like brightness, saturation, or other relevant properties to fine-tune the level or quality of the filter's effect. This level of control enables you to achieve the exact look or outcome you want for your image.

#### Adjust the brightness, contrast, or sharpness

The [finetuneImage](#) method is designed to facilitate fine-tuning operations on an image. It accepts two parameters: the first parameter is [ImageFinetuneOption](#) which determines the type of fine-tuning to be applied (brightness, contrast, or sharpness), and the second parameter represents the fine-tuning value, indicating the degree or intensity of the adjustment. This method allows for convenient adjustment of brightness, contrast, or sharpness by specifying the desired type and corresponding value.

The [finetuneImage](#) method is used to perform brightness, contrast, or sharpness fine-tuning by specifying this type as a first parameter and specifying the fine-tuning value as the second parameter of the method.

Here is an example of brightness, contrast, and sharpness fine-tuning using the [finetuneImage](#) method.

#### APP.VUE

```

<template>
<div>
<ejs-imageeditor id="image-editor" ref="imageEditorObj" height="350px"
width="550px"></ejs-imageeditor>

```

```

<ejs-button cssClass="e-img-button" :isPrimary="true" v-
on:click.native="brightnessClick">Brightness</ejs-button>
<ejs-button cssClass="e-img-button" :isPrimary="true" v-
on:click.native="contrastClick">Contrast</ejs-button>
</div>
</template>
<script>
import Vue from 'vue';
import { ImageEditorPlugin, ImageFinetuneOption } from "@syncfusion/ej2-vue-
image-editor";
import { ButtonPlugin } from '@syncfusion/ej2-vue-buttons';
import { Browser } from "@syncfusion/ej2-base";
Vue.use(ImageEditorPlugin);
Vue.use(ButtonPlugin);
Vue.use(ImageFinetuneOption);
export default {
  data: function() {
    return {
      toolbar: []
    };
  },
  methods: {
    created: function() {
      if (Browser.isDevice) {
        this.$refs.imageEditorObj.open('flower.png');
      } else {
        this.$refs.imageEditorObj.open('bridge.png');
      }
    },
    brightnessClick: function(event) {
      this.$refs.imageEditorObj.ej2Instances.finetuneImage(ImageFinetuneOption.Bri
ghtness, 10);
    },
    contrastClick: function(event) {
      this.$refs.imageEditorObj.ej2Instances.finetuneImage(ImageFinetuneOption.Con
trast, 10);
    }
  }
}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-buttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-splitbuttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-lists/styles/material.css";
@import "../node_modules/@syncfusion/ej2-popups/styles/material.css";
@import "../node_modules/@syncfusion/ej2-inputs/styles/material.css";
@import "../node_modules/@syncfusion/ej2-navigations/styles/material.css";
@import "../node_modules/@syncfusion/ej2-dropdowns/styles/material.css";
@import "../node_modules/@syncfusion/ej2-image-editor/styles/material.css";
#image-editor {
  width: 550px !important;
  height: 350px !important;
}
</style>

```

```
{% previewsample "page.domainurl/code-snippet/image-editor/getting-started-cs31" %}
```

Adjust the hue, exposure, blur, or opacity

The [finetuneImage](#) method is designed to facilitate fine-tuning operations on an image. It accepts two parameters: the first parameter is [ImageFinetuneOption](#) which determines the type of fine-tuning to be applied (hue, exposure, or blur), and the second parameter represents the fine-tuning value, indicating the degree or intensity of the adjustment. This method allows for convenient adjustment of hue, exposure, or blur by specifying the desired type and corresponding value.

Here is an example of hue, exposure, and blur fine-tuning using the [finetuneImage](#) method.

### APP.VUE

```
<template>
<div>
<ejs-imageeditor id="image-editor" ref="imageEditorObj" height="350px"
width="550px"></ejs-imageeditor>
<ejs-button cssClass="e-img-button" :isPrimary="true" v-
on:click.native="hueClick">Hue</ejs-button>
<ejs-button cssClass="e-img-button" :isPrimary="true" v-
on:click.native="exposureClick">Exposure</ejs-button>
<ejs-button cssClass="e-img-button" :isPrimary="true" v-
on:click.native="opacityClick">Opacity</ejs-button>
</div>
</template>
<script>
import Vue from 'vue';
import { ImageEditorPlugin } from "@syncfusion/ej2-vue-image-editor";
import { ButtonPlugin } from '@syncfusion/ej2-vue-buttons';
import { Browser } from "@syncfusion/ej2-base";
Vue.use(ImageEditorPlugin);
Vue.use(ButtonPlugin);
export default {
  data: function() {
    return {
      toolbar: []
    };
  },
  methods: {
    created: function() {
      if (Browser.isDevice) {
        this.$refs.imageEditorObj.open('flower.png');
      } else {
        this.$refs.imageEditorObj.open('bridge.png');
      }
    },
    hueClick: function(event) {
      this.$refs.imageEditorObj.ej2Instances.finetuneImage(ImageFinetuneOption.Hue
, 10);
    },
    exposureClick: function(event) {
      this.$refs.imageEditorObj.ej2Instances.finetuneImage(ImageFinetuneOption.Exp
osure, 10);
    }
  }
}
```

```

    },
    opacityClick: function(event) {

this.$refs.imageEditorObj.ej2Instances.finetuneImage(ImageFinetuneOption.Opa
city, 10);
    }
  }
}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-buttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-splitbuttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-lists/styles/material.css";
@import "../node_modules/@syncfusion/ej2-popups/styles/material.css";
@import "../node_modules/@syncfusion/ej2-inputs/styles/material.css";
@import "../node_modules/@syncfusion/ej2-navigations/styles/material.css";
@import "../node_modules/@syncfusion/ej2-dropdowns/styles/material.css";
@import "../node_modules/@syncfusion/ej2-image-editor/styles/material.css";
#image-editor {
  width: 550px !important;
  height: 350px !important;
}
</style>

```

{% previewsample "page.domainurl/code-snippet/image-editor/getting-started-cs25" %}

### Finetune value changing event

The [FinetuneValueChanging](#) event is triggered when performing fine-tuning on the image. This event is passed an object that contains information about the fine-tuning event, such as the type of fine-tuning and the value of fine-tuning performed.

The parameter available in the [FinetuneEventArgs](#) event is,

- `FinetuneEventArgs.finetune` - The type of fine-tuning as [ImageFinetuneOption](#) to be applied in the image editor.
- `FinetuneEventArgs.value` - The fine-tuning value to be applied in the image editor.
- `FinetuneEventArgs.cancel` – Specifies a boolean value to cancel the fine-tuning action.

### Frames

The frame feature in an Image Editor provides users with the capability to add decorative borders or frames around their images. Frames are a visual design element that can enhance the overall appearance and appeal of an image.

#### Apply frame to the Image

The [drawFrame](#) method is a function designed to enable the application of various frame options to an image. This method simplifies the process of adding decorative frames, such as mat, bevel, line, hook, and inset, to an image by allowing users to specify their desired frame type.

Depending on the frame type selected, users may have additional customization options, such as adjusting the frame's thickness, color, texture, or other attributes. This allows for fine-tuning the appearance of the frame to match the image's theme or the user's preferences

The [drawFrame](#) method in the Image Editor control takes nine parameters to define the properties of the frame to the image:

- `frameType` - Specified the image data or url of the image to be inserted.
- `Color` - Specifies the color for the frame.
- `gradientColor` - Specifies the gradient color for the frame.
- `size` - Specifies the size of the frame.
- `inset` - Specifies the inset value for line, hook, and inset type frames.
- `offset` - Specifies the offset value for line and inset type frames.
- `borderRadius` - Specifies the border radius for line type frame.
- `frameLineStyle` - Specifies the frame line style for line type frame.
- `lineCount` - Specifies the line count for the line type frame.

In the following example, you can use the `drawFrame` method in the button click event.

#### APP.VUE

```
<template>
<div>
<ejs-imageeditor id="image-editor" ref="imageEditorObj" height="350px"
width="550px"></ejs-imageeditor>
<ejs-button cssClass="e-img-button" :isPrimary="true" v-
on:click.native="matClick">Mat</ejs-button>
<ejs-button cssClass="e-img-button" :isPrimary="true" v-
on:click.native="bevelClick">Bevel</ejs-button>
<ejs-button cssClass="e-img-button" :isPrimary="true" v-
on:click.native="lineClick">Line</ejs-button>
<ejs-button cssClass="e-img-button" :isPrimary="true" v-
on:click.native="insetClick">Inset</ejs-button>
<ejs-button cssClass="e-img-button" :isPrimary="true" v-
on:click.native="hookClick">Hook</ejs-button>
</div>
</template>
<script>
import Vue from 'vue';
import { ImageEditorPlugin, FrameType, FrameLineStyle } from
"@syncfusion/ej2-vue-image-editor";
import { ButtonPlugin } from '@syncfusion/ej2-vue-buttons';
import { Browser } from "@syncfusion/ej2-base";
Vue.use(ImageEditorPlugin);
Vue.use(ButtonPlugin);
export default {
  data: function() {
    return {
      toolbar: []
    };
  },
  methods: {
    created: function() {
      if (Browser.isDevice) {
        this.$refs.imageEditorObj.open('flower.png');
      } else {
        this.$refs.imageEditorObj.open('bridge.png');
      }
    }
  },
}
```

```

    matClick: function(event) {
        this.$refs.imageEditorObj.ej2Instances.drawFrame(FrameType.Mat, 'red',
        'blue', 20, 20, 20, 20, FrameLineStyle.Solid, 1);
    },
    bevelClick: function(event) {
        this.$refs.imageEditorObj.ej2Instances.drawFrame(FrameType.Bevel,
        'red', 'blue', 20, 20, 20, 20, FrameLineStyle.Solid, 1);
    },
    lineClick: function(event) {
        this.$refs.imageEditorObj.ej2Instances.drawFrame(FrameType.Line,
        'red', 'blue', 20, 20, 20, 20, FrameLineStyle.Solid, 1);
    },
    insetClick: function(event) {
        this.$refs.imageEditorObj.ej2Instances.drawFrame(FrameType.Inset,
        'red', 'blue', 20, 20, 20, 20, FrameLineStyle.Solid, 1);
    },
    hookClick: function(event) {
        this.$refs.imageEditorObj.ej2Instances.drawFrame(FrameType.Hook,
        'red', 'blue', 20, 20, 20, 20, FrameLineStyle.Solid, 1);
    }
}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-buttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-splitbuttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-lists/styles/material.css";
@import "../node_modules/@syncfusion/ej2-popups/styles/material.css";
@import "../node_modules/@syncfusion/ej2-inputs/styles/material.css";
@import "../node_modules/@syncfusion/ej2-navigations/styles/material.css";
@import "../node_modules/@syncfusion/ej2-dropdowns/styles/material.css";
@import "../node_modules/@syncfusion/ej2-image-editor/styles/material.css";
#image-editor {
    width: 550px !important;
    height: 350px !important;
}
</style>

```

{% previewsample "page.domainurl/code-snippet/image-editor/getting-started-cs41" %}

### Frame changing event

The [frameChanging](#) event is triggered when applying frame on the image. This event provides information encapsulated within an object, which includes details about the frame applied in an image. This information encompasses:

**Frame Type:** This indicates the specific type of frame being applied, whether it's a mat, bevel, line, or hook.

**Customization Values:** These values contain information about any adjustments or modifications made to the frame. For instance, if the frame can be customized with attributes like color, size, or style, these details are conveyed within the event object.

The parameter available in the [FrameChangeEventArgs](#) is

- [FrameChangeEventArgs.previousFrameSetting](#) - The frame settings including size, color, inset, offset, gradient color which is applied before changing the frame.
- [FrameChangeEventArgs.currentFrameSetting](#) - The frame settings including size, color, inset, offset, gradient color which is going to apply after changing the frame.
- [FrameChangeEventArgs.cancel](#) - Specifies a boolean value to cancel the frame changing action.

## Resize

The resize feature in an Image Editor is a valuable tool that empowers users to modify the size or dimensions of an image to meet their specific requirements. Whether it's for printing, web display, or any other purpose, this feature allows users to tailor images to their desired specifications.

### Apply resize to the image

The Image Editor control includes a [resize](#) method, which allows you to adjust the size of an image. This method takes three parameters that define how the resizing should be carried out:

- width: Specifies the resizing width of the image.
- height: Specifies the resizing height of the image.
- isAspectRatio: Specifies a boolean value indicating whether the image should maintain its original aspect ratio during resizing. When set to true, the image will be resized while preserving its aspect ratio

Here is an example of resizing the image using the [resize](#) method.

### APP.VUE

```
<template>
<div>
<ejs-imageeditor id="image-editor" ref="imageEditorObj" height="350px"
width="550px"></ejs-imageeditor>
<ejs-button cssClass="e-img-button" :isPrimary="true" v-
on:click.native="aspectClick">Aspect Ratio</ejs-button>
<ejs-button cssClass="e-img-button" :isPrimary="true" v-
on:click.native="nonAspectClick">Non Aspect Ratio</ejs-button>
</div>
</template>
<script>
import Vue from 'vue';
import { ImageEditorPlugin } from "@syncfusion/ej2-vue-image-editor";
import { ButtonPlugin } from '@syncfusion/ej2-vue-buttons';
import { Browser } from "@syncfusion/ej2-base";
Vue.use(ImageEditorPlugin);
Vue.use(ButtonPlugin);
export default {
  data: function() {
    return {
      toolbar: []
    };
  },
  methods: {
    created: function() {
      if (Browser.isDevice) {
        this.$refs.imageEditorObj.open('flower.png');
      } else {
```

```

        this.$refs.imageEditorObj.open('bridge.png');
    },
    aspectClick: function(event) {
        this.$refs.imageEditorObj.ej2Instances.resize(300, 400, true);
    },
    nonAspectClick: function(event) {
        this.$refs.imageEditorObj.ej2Instances.resize(400, 100, false);
    }
}
}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-buttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-splitbuttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-lists/styles/material.css";
@import "../node_modules/@syncfusion/ej2-popups/styles/material.css";
@import "../node_modules/@syncfusion/ej2-inputs/styles/material.css";
@import "../node_modules/@syncfusion/ej2-navigations/styles/material.css";
@import "../node_modules/@syncfusion/ej2-dropdowns/styles/material.css";
@import "../node_modules/@syncfusion/ej2-image-editor/styles/material.css";
#image-editor {
    width: 550px !important;
    height: 350px !important;
}
</style>

```

{% previewsample "page.domainurl/code-snippet/image-editor/getting-started-cs42" %}

### Resizing event

The [resizing](#) event is triggered when resizing the image. This event provides information encapsulated within an object, which includes details about the previous and current height and width of an image.

The parameter available in [ResizeEventArgs](#) is,

- [ResizeEventArgs.previousWidth](#) - The width of the image before resizing is performed.
- [ResizeEventArgs.previousHeight](#) - The height of the image before resizing is performed.
- [ResizeEventArgs.width](#) - The width of the image after resizing is performed.
- [ResizeEventArgs.height](#) - The width of the image after resizing is performed.
- [ResizeEventArgs.isAspectRatio](#) - The type of resizing performed such as aspect ratio or non-aspect ratio.
- [ResizeEventArgs.cancel](#) - Specifies a boolean value to cancel the resizing action.

### Localization in the Vue Image Editor component

The **Localization** library allows you to localize the default text content of the Image Editor. The Image Editor has static text that can be changed to other cultures (Arabic, Deutsch, French, etc.) by defining the **locale** value and translation object.

The following list of properties and its values are used in the Image Editor.

| Locale key words | Text |

| ----- | ----- |



Browse	Browse
Crop	Crop
ZoomIn	Zoom In
ZoomOut	Zoom Out
Transform	Transform
Annotation	Annotation
Text	Add Text
Pen	Pen
Reset	Reset
Save	Save
Select	Select
RotateLeft	Rotate Left
RotateRight	Rotate Right
HorizontalFlip	Horizontal Flip
VerticalFlip	Vertical Flip
OK	OK
Cancel	Cancel
FillColor	Fill Color
StrokeColor	Stroke Color
StrokeWidth	StrokeWidth
FontFamily	Font Family
FontStyle	Font Style
FontSize	Font Size
FontColor	Font Color
Pan	Pan
Move	Move
Custom	Custom
Square	Square
Circle	Circle
Rectangle	Rectangle
Line	Line
Default	Default
Bold	Bold

| Italic | Italic |

| BoldItalic | Bold Italic |

| XSmall | X-Small |

| Small | Small |

| Medium | Medium |

| Large | Large |

| XLarge | X-Large |

| ABC | ABC |

**APP.VUE**

```
<template>
<div>
<ejs-imageeditor id="image-editor" height="350px" width="550px" locale="de-
DE"></ejs-imageeditor>
</div>
</template>
<script>
import Vue from 'vue';
import { ImageEditorPlugin } from "@syncfusion/ej2-vue-image-editor";
import { L10n, setCulture } from '@syncfusion/ej2-base';
Vue.use(ImageEditorPlugin);
setCulture('de-DE');
L10n.load({
  'de-DE': {
    'image-editor': {
      'Browse': 'Durchsuche',
      'Crop': 'Ernte',
      'ZoomIn': 'Hineinzoomen',
      'ZoomOut': 'Rauszoomen',
      'Transform': 'Verwandeln',
      'Annotation': 'Anmerkung',
      'Text': 'Text hinzufügen',
      'Pen': 'Stift',
      'Reset': 'Zurücksetzen',
      'Save': 'Speichern',
      'Select': 'Auswählen',
      'RotateLeft': 'Nach links drehen',
      'RotateRight': 'Drehe nach rechts',
      'HorizontalFlip': 'Horizontaler Flip',
      'VerticalFlip': 'Vertikaler Flip',
      'OK': 'OK',
      'Cancel': 'Absagen',
      'FillColor': 'Füllfarbe',
      'StrokeColor': 'Strichfarbe',
      'StrokeWidth': 'Strichbreite',
      'FontFamily': 'Schriftfamilie',
      'FontStyle': 'Schriftstil',
      'FontSize': 'Schriftgröße',
      'FontColor': 'Schriftfarbe',
      'Pan': 'Pfanne',
      'Move': 'Bewegen',
```

```

        'Custom': 'Brauch',
        'Square': 'Quadrat',
        'Circle': 'Kreis',
        'Rectangle': 'Rechteck',
        'Line': 'Linie',
        'Default': 'Standard',
        'Bold': 'Fett gedruckt',
        'Italic': 'Kursiv',
        'BoldItalic': 'Fett Kursiv',
    }
}
});
export default {
  data: function() {
    return {};
  }
}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-buttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-splitbuttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-lists/styles/material.css";
@import "../node_modules/@syncfusion/ej2-popups/styles/material.css";
@import "../node_modules/@syncfusion/ej2-inputs/styles/material.css";
@import "../node_modules/@syncfusion/ej2-navigations/styles/material.css";
@import "../node_modules/@syncfusion/ej2-dropdowns/styles/material.css";
@import "../node_modules/@syncfusion/ej2-image-editor/styles/material.css";
#image-editor {
  width: 550px !important;
  height: 350px !important;
}
</style>

```

{% previewsample "page.domainurl/code-snippet/image-editor/getting-started-cs8" %}

## Accessibility in Vue Image Editor component

The Image Editor component followed the accessibility guidelines and standards, including [ADA, Section 508](#), [WCAG 2.2](#) standards, and [WCAG roles](#) that are commonly used to evaluate accessibility.

The accessibility compliance for the Image Editor component is outlined below.

| Accessibility Criteria | Compatibility |

| -- | -- |

| [WCAG 2.2](#) Support |  |

| [Section 508](#) Support |  |

| Screen Reader Support |  |

| Right-To-Left Support |  |

| Color Contrast | `` |

| Mobile Device Support | `` |

| Keyboard Navigation Support | `` |

| [Accessibility Checker](#) Validation | `` |

| [Axe-core](#) Accessibility Validation | `` |

`<style>`

`.post .post-content img {`

`display: inline-block;`

`margin: 0.5em 0;`

`}`

`</style>`

`<div>` - All features of the component meet the requirement.`</div>`

`<div>` - Some features of the component do not meet the requirement.`</div>`

`<div>` - The component does not meet the requirement.`</div>`

### Keyboard interaction

The Image Editor component followed the keyboard interaction guideline, making it easy for people who use assistive technologies (AT) and those who completely rely on keyboard navigation. The following keyboard shortcuts are supported by the Image Editor component.

| **Press** | **To do this** |

| --- | --- |

| **Ctrl + Z** | **Undo the last user action.** |

| **Ctrl + Y** | **Redo the last user action.** |

| **Ctrl + S** | **To save the Image.** |

| **Ctrl + O** | **To open the Image.** |

| **Delete** | **To delete the shape once the shape got selected through mouse click .** |

### Ensuring accessibility

The Image Editor component's accessibility levels are ensured through an [accessibility-checker](#) and [axe-core](#) software tools during automated testing.

The accessibility compliance of the Image Editor component is shown in the following sample. Open the [sample](#) in a new window to evaluate the accessibility of the Image Editor component with accessibility tools.

```
{% previewsample "page.domainurl/code-snippet/image-editor/getting-started-cs1" %}
```

See also

- [Accessibility in Syncfusion Vue components](#)

## InplaceEditor

### Getting Started with the Vue Inplace editor Component in Vue 2

This article provides a step-by-step guide for setting up a Vue 2 project using [Vue-CLI](#) and integrating the Syncfusion Vue Inplace editor component

#### Prerequisites

[System requirements for Syncfusion Vue UI components](#)

#### Dependencies

The following list of dependencies are required to use the In-place Editor component in your application.

```
`javascript
|-- @syncfusion/ej2-vue-inplace-editor
|-- @syncfusion/ej2-base
|-- @syncfusion/ej2-buttons
|-- @syncfusion/ej2-calendars
|-- @syncfusion/ej2-data
|-- @syncfusion/ej2-dropdowns
|-- @syncfusion/ej2-inputs
|-- @syncfusion/ej2-lists
|-- @syncfusion/ej2-navigations
|-- @syncfusion/ej2-popups
|-- @syncfusion/ej2-richtexteditor
|-- @syncfusion/ej2-splitbuttons
|-- @syncfusion/ej2-vue-base
`,`
```

#### Setting up the Vue 2 project

To generate a Vue 2 project using Vue-CLI, use the [vue create](#) command. Follow these steps to install Vue CLI and create a new project:

```
`bash
npm install -g @vue/cli
```

```
vue create quickstart
```

```
cd quickstart
```

```
npm run serve
```

```
,
```

```
or
```

```
`bash
```

```
yarn global add @vue/cli
```

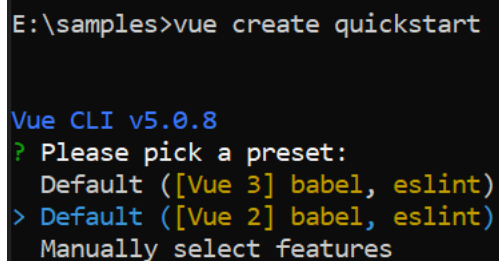
```
vue create quickstart
```

```
cd quickstart
```

```
yarn run serve
```

```
,
```

When creating a new project, choose the option `Default ([Vue 2] babel, eslint)` from the menu.



```
E:\samples>vue create quickstart

Vue CLI v5.0.8
? Please pick a preset:
  Default ([Vue 3] babel, eslint)
> Default ([Vue 2] babel, eslint)
  Manually select features
```

Once the `quickstart` project is set up with default settings, proceed to add Syncfusion components to the project.

#### Add Syncfusion Vue packages

Syncfusion packages are available at [npmjs.com](https://www.npmjs.com). To use Vue components, install the required npm package.

This article uses the [Vue Inplace editor component](#) as an example. Install the `@syncfusion/ej2-vue-inplace-editor` package by running the following command:

```
`bash
```

```
npm install @syncfusion/ej2-vue-inplace-editor --save
```

```
,
```

```
or
```

```
`bash
```

```
yarn add @syncfusion/ej2-vue-inplace-editor
```

```
,
```

### Import Syncfusion CSS styles

You can import themes for the Syncfusion Vue component in various ways, such as using CSS or SASS styles from npm packages, CDN, [CRG](#) and [Theme Studio](#). Refer to [themes topic](#) to know more about built-in themes and different ways to refer to themes in a Vue project.

In this article, the **Material** theme is applied using CSS styles, which are available in installed packages. The necessary Material CSS styles for the Inplace editor component and its dependents were imported into the `<style>` section of **src/App.vue** file.

#### ~/SRC/APP.VUE

```
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-buttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-calendars/styles/material.css";
@import "../node_modules/@syncfusion/ej2-dropdowns/styles/material.css";
@import "../node_modules/@syncfusion/ej2-inputs/styles/material.css";
@import "../node_modules/@syncfusion/ej2-lists/styles/material.css";
@import "../node_modules/@syncfusion/ej2-navigations/styles/material.css";
@import "../node_modules/@syncfusion/ej2-popups/styles/material.css";
@import "../node_modules/@syncfusion/ej2-richtexteditor/styles/material.css";
@import "../node_modules/@syncfusion/ej2-splitbuttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-inplace-editor/styles/material.css";
</style>
```

### Add Syncfusion Vue component

Follow the below steps to add the Vue Inplace editor component:

1\ First, import and register the Inplace editor component in the **script** section of the **src/App.vue** file.

#### ~/SRC/APP.VUE

```
<script>
import { InPlaceEditorComponent } from '@syncfusion/ej2-vue-inplace-editor';
export default {
  components: {
    'ejs-inplaceeditor': InPlaceEditorComponent
  }
}
</script>
```

2\ In the **template** section, define the Inplace editor component with the [model](#) property.

#### ~/SRC/APP.VUE

```
<template>
<div id="app">
<ejs-inplaceeditor id="inplace_editor" type="Text" value="Andrew"
:model="model">
</ejs-inplaceeditor>
</div>
</template>
```

3\ Declare the value for the `model` property in the `script` section.

#### ~/SRC/APP.VUE

```
<script>
data () {
  return {
    model: {
      placeholder: 'Enter employee name'
    },
  }
}
</script>
```

#### Add the In-place Editor with Textbox

By default, the TextBox component is rendered in In-place Editor with the `type` property sets as Text.

,

```
<template>
<div id="app">
  <ejs-inplaceeditor id="inplace_editor" type="Text" value="Andrew" :model="model">
  </ejs-inplaceeditor>
</div>
</template>
<script>
import { InPlaceEditorComponent } from '@syncfusion/ej2-vue-inplace-editor';
export default {
  components: {
    'ejs-inplaceeditor': InPlaceEditorComponent
  },
  name: 'app',
  data: function(){
    return {
      model: {
        placeholder: 'Enter employee name'
      },
    }
  }
}
</script>
```



,

### Configure DropDownList

You can render DropDownList by changing the [type](#) property as [DropDownList](#) and configure its properties and methods using the `model` property.

In the following sample, [type](#) and `model` values are configured to render the [DropDownList](#) component.

,

```
<template>
<div id="app">
  <ejs-inplaceeditor id="element" type="DropDownList" mode= "Inline" :model="model">
  </ejs-inplaceeditor>
</div>
</template>
<script>
import { InPlaceEditorComponent } from '@syncfusion/ej2-vue-inplace-editor';
export default {
  components: {
    'ejs-inplaceeditor': InPlaceEditorComponent
  },
  name: 'app',
  data () {
    return {
      model: {
        genderData: ['Male', 'Female'],
        placeholder: 'Select gender'
      },
    }
  }
}
</script>
```

,

### Integrate DatePicker

You can render [DatePicker](#) by changing the [type](#) property as [Date](#) and also configure its properties and methods using [model](#) property.

In the below sample, [type](#) and [model](#) values are configured to render the [DatePicker](#) component.

```
,  
<template>  
<div id="app">  
<ejs-inplaceeditor id="inplace_editor" type="Date" :model="model" :value="value">  
</ejs-inplaceeditor>  
</div>  
</template>  
<script>  
import { InPlaceEditorComponent } from '@syncfusion/ej2-vue-inplace-editor';  
export default {  
  components: {  
    'ejs-inplaceeditor': InPlaceEditorComponent  
  },  
  name: 'app',  
  data () {  
    return {  
      value: new Date('04/12/2018'),  
      model: {  
        showTodayButton: true  
      },  
    }  
  }  
}  
</script>  
,
```

Here is the summarized code for the above steps in the **src/App.vue** file:

#### ~/SRC/APP.VUE

```
<template>  
  <div id="app">  
    <div class="control-group">  
      <h3> Modify Basic Details </h3>  
      <table>  
        <tr>  
          <td>Name</td>  
          <td class='left'>  
            <ejs-inplaceeditor id="element" type="Text" mode="Inline"  
value="Andrew" :model="textModel"></ejs-inplaceeditor>  
          </td>  
        </tr>  
      </table>  
    </div>  
  </div>  
</template>
```

```

        </td>
      </tr>
      <tr>
        <td>Date of Birth</td>
        <td class='left'>
          <ejs-inplaceeditor id="dateofbirth" type="Date"
mode="Inline" :value="dateValue" :model="dateModel"></ejs-inplaceeditor>
        </td>
      </tr>
      <tr>
        <td>Gender</td>
        <td class='left'>
          <ejs-inplaceeditor id="gender" type="DropDownList" mode=
"Inline" value="Male" :model="dropdownModel"></ejs-inplaceeditor>
        </td>
      </tr>
    </table>
  </div>
</div>
</template>
<script>
import { InPlaceEditorComponent } from '@syncfusion/ej2-vue-inplace-editor';
export default {
  components: {
    'ejs-inplaceeditor': InPlaceEditorComponent
  },
  name: 'app',
  data () {
    return {
      dateValue: new Date('04/12/2018'),
      dateModel: {
        showTodayButton: true,
        placeholder: 'Select date of birth'
      },
      textModel: {
        placeholder: 'Enter your name'
      },
      dropdownModel: {
        dataSource: ['Male', 'Female'],
        placeholder: 'Select gender'
      },
    }
  }
}
</script>
<style>
  @import "../node_modules/@syncfusion/ej2-base/styles/material.css";
  @import "../node_modules/@syncfusion/ej2-buttons/styles/material.css";
  @import "../node_modules/@syncfusion/ej2-calendars/styles/material.css";
  @import "../node_modules/@syncfusion/ej2-dropdowns/styles/material.css";
  @import "../node_modules/@syncfusion/ej2-inputs/styles/material.css";
  @import "../node_modules/@syncfusion/ej2-lists/styles/material.css";
  @import "../node_modules/@syncfusion/ej2-navigations/styles/material.css";
  @import "../node_modules/@syncfusion/ej2-popups/styles/material.css";
  @import "../node_modules/@syncfusion/ej2-richtexteditor/styles/material.css";

```

```
@import "../node_modules/@syncfusion/ej2-splitbuttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-inplace-editor/styles/material.css";
</style>
<style>
#app {
    visibility: hidden;
}
#app .control-group {
    text-align: center;
    margin-top: 50px;
}
#app .control-group table {
    width: 400px;
    margin:auto;
}
#app .control-group table td {
    height: 70px;
    width: 150px;
}
#app .control-group table td.left {
    text-align: left;
}
</style>
```

### Run the project

To run the project, use the following command:

```
`bash
```

```
npm run serve
```

```
,
```

or

```
`bash
```

```
yarn run serve
```

```
,
```

```
{% previewsample "page.domainurl/code-snippet/in-place-editor/getting-started-cs11" %}
```

### Two-way binding

In In-place Editor, two-way binding support is achieved using the `v-model` directive in Vue. When you change a value in the first In-place Editor component, the changed value gets updated automatically to the second In-place Editor. The following example demonstrates, how to achieve two-way binding in In-place Editor.

#### ~/SRC/APP.VUE

```
<template>
  <div id="app">
    <div id='container'>
      <div class="control-group">
```

```

        <table>
          <tr>
            <td><b>TextBox :</b></td>
            <ejs-inplaceeditor id="textbox" type="Text" mode="Inline"
:~model="textModel" v-model="value">
            </ejs-inplaceeditor>
            <ejs-inplaceeditor id="textbox2" type="Text" mode="Inline"
:~model="textModel" v-model="value">
            </ejs-inplaceeditor>
          </tr>
          <tr>
            <td><b>Datepicker :</b></td>
            <ejs-inplaceeditor id="dateeditor1" mode="Inline"
type="Date" :model="dateModel" v-model="datePickerValue">
            </ejs-inplaceeditor>
            <ejs-inplaceeditor id="dateeditor2" mode="Inline"
type="Date" :model="dateModel" v-model="datePickerValue">
            </ejs-inplaceeditor>
          </tr>
          <tr>
            <td><b>DropDownList :</b></td>
            <ejs-inplaceeditor id="dropDowneditor1" mode="Inline"
type="DropDownList" :model="dropdownModel" v-model="dropdownValue">
            </ejs-inplaceeditor>
            <ejs-inplaceeditor id="dropDowneditor2" mode="Inline"
type="DropDownList" :model="dropdownModel" v-model="dropdownValue">
            </ejs-inplaceeditor>
          </tr>
        </table>
      </div>
    </div>
  </div>
</template>
<script>
import { InPlaceEditorComponent } from '@syncfusion/ej2-vue-inplace-editor';
export default {
  components: {
    'ejs-inplaceeditor': InPlaceEditorComponent
  },
  name: 'app',
  data () {
    let frameWorkList = ['Android', 'JavaScript', 'jQuery', 'TypeScript',
'Angular', 'React', 'Vue', 'Ionic'];
    return {
      value: 'Andrew',
      dropdownValue: 'Android',
      datePickerValue: new Date('11/23/2018'),
      textModel: {
        placeholder: 'Enter employee name'
      },
      dropdownModel: {
        placeholder: 'Select frameWorks',
        dataSource: frameWorkList
      },
      dateModel: {
        placeholder: 'Select date'
      }
    }
  }
}

```

```

    }
  }
}
</script>
<style>
  @import "../node_modules/@syncfusion/ej2-base/styles/material.css";
  @import "../node_modules/@syncfusion/ej2-buttons/styles/material.css";
  @import "../node_modules/@syncfusion/ej2-calendars/styles/material.css";
  @import "../node_modules/@syncfusion/ej2-dropdowns/styles/material.css";
  @import "../node_modules/@syncfusion/ej2-inputs/styles/material.css";
  @import "../node_modules/@syncfusion/ej2-lists/styles/material.css";
  @import "../node_modules/@syncfusion/ej2-navigations/styles/material.css";
  @import "../node_modules/@syncfusion/ej2-popups/styles/material.css";
  @import "../node_modules/@syncfusion/ej2-richtexteditor/styles/material.css";
  @import "../node_modules/@syncfusion/ej2-splitbuttons/styles/material.css";
  @import "../node_modules/@syncfusion/ej2-vue-inplace-editor/styles/material.css";
</style>
<style>
#app {
  visibility: hidden;
}
#app .control-group {
  text-align: center;
  margin-top: 50px;
}
#app .control-group table {
  width: 400px;
  margin:auto;
}
#app .control-group table td {
  height: 70px;
  width: 150px;
}
#app .control-group table td.left {
  text-align: left;
}
</style>

```

{% previewsample "page.domainurl/code-snippet/in-place-editor/getting-started-cs12" %}

### Submitting data to the server (save)

You can submit editor value to server by configuring the [url](#), [adaptor](#), and [primaryKey](#) property.

Property	Usage
<b>url</b>	Gets the url for server submit action.
<b>adaptor</b>	Specifies the adaptor type that is used by DataManager to communicate with DataSource.
<b>primaryKey</b>	Defines the unique primary key of editable field which can be used for saving data in the data-base.

The [primaryKey](#) property is mandatory. If it's not set, edited data are not sent to the server.

#### Refresh with modified value

The edited data is submitted to the server and you can see the new values getting reflected in the In-place Editor.

#### ~/SRC/APP.VUE

```
<template>
  <div id="app">
    <div class="container">
      <div class="control-group" style="text-align:center;margin: 100px auto">
        Best Employee of the year: <ejs-inplaceeditor id="element"
type="DropDownList" mode="Inline" value="Andrew Fuller" name="Employee"
:url="serviceUrl" primaryKey="Employee" adaptor="UrlAdaptor"
:model="dropdownModel" :actionSuccess= "actionSuccess"
:created='created'></ejs-inplaceeditor>
      </div>
      <table style="margin:auto;width:50%">
        <tr>
          <td style="text-align: left">
            Old Value :
          </td>
          <td id="oldValue" ref="oldValue" style="text-align: left">
          </td>
        </tr>
        <tr>
          <td style="text-align: left">
            New Value :
          </td>
          <td id="newValue" ref="newValue" style="text-align: left">
            Andrew Fuller
          </td>
        </tr>
      </table>
    </div>
  </div>
</template>
<script>
import { InPlaceEditorComponent, MultiSelect } from '@syncfusion/ej2-vue-
inplace-editor';
export default {
  components: {
    'ejs-inplaceeditor': InPlaceEditorComponent
  },
  name: 'app',
  data () {
    return {
      serviceUrl: "https://ej2services.syncfusion.com/development/web-
services/api/Editor/UpdateData",
      dropdownModel: {
        dataSource: ['Andrew Fuller', 'Janet Leverling', 'Laura
Callahan', 'Margaret Hamilt', 'Michael Suyama', 'Nancy Davloio', 'Robert
King'],
        popupHeight: '200px',
        placeholder: 'Select employee'
      }
    }
  }
}
```

```

    },
  },
},
methods: {
  created: function() {
    this.newValue = this.$refs.newValue;
    this.oldValue = this.$refs.oldValue;
  },
  actionSuccess: function(e) {
    this.oldValue.textContent = this.newValue.textContent;
    this.newValue.textContent = e.value;
  }
},
provide: {
  "inplaceeditor": [MultiSelect]
}
}
</script>
<style>
  @import "../node_modules/@syncfusion/ej2-base/styles/material.css";
  @import "../node_modules/@syncfusion/ej2-buttons/styles/material.css";
  @import "../node_modules/@syncfusion/ej2-calendars/styles/material.css";
  @import "../node_modules/@syncfusion/ej2-dropdowns/styles/material.css";
  @import "../node_modules/@syncfusion/ej2-inputs/styles/material.css";
  @import "../node_modules/@syncfusion/ej2-lists/styles/material.css";
  @import "../node_modules/@syncfusion/ej2-navigations/styles/material.css";
  @import "../node_modules/@syncfusion/ej2-popups/styles/material.css";
  @import "../node_modules/@syncfusion/ej2-richtexteditor/styles/material.css";
  @import "../node_modules/@syncfusion/ej2-splitbuttons/styles/material.css";
  @import "../node_modules/@syncfusion/ej2-vue-inplace-editor/styles/material.css";
</style>
<style>
.e-inplaceeditor {
  min-width: 200px;
  text-align: left
}
</style>

```

{% previewsample "page.domainurl/code-snippet/in-place-editor/getting-started-cs13" %}

See Also

- [Types of rendering the editor](#)

## Getting Started with the Vue In-place Editor Component in Vue 3

This article provides a step-by-step guide for setting up a [Vite](#) project with a JavaScript environment and integrating the Syncfusion Vue In-place Editor component using the [Composition API](#) / [Options API](#).

The **Composition API** is a new feature introduced in Vue.js 3 that provides an alternative way to organize and reuse component logic. It allows developers to write components as functions that use smaller, reusable functions called composition functions to manage their properties and behavior.



The **Options API** is the traditional way of writing Vue.js components, where the component logic is organized into a series of options that define the component's properties and behavior. These options include data, methods, computed properties, watchers, lifecycle hooks, and more.

#### Prerequisites

##### [System requirements for Syncfusion Vue UI components](#)

#### Set up the Vite project

A recommended approach for beginning with Vue is to scaffold a project using [Vite](#). To create a new Vite project, use one of the commands that are specific to either NPM or Yarn.

```
`bash
```

```
npm create vite@latest
```

```
`
```

or

```
`bash
```

```
yarn create vite
```

```
`
```

Using one of the above commands will lead you to set up additional configurations for the project as below:

1. Define the project name: We can specify the name of the project directly. Let's specify the name of the project as **my-project** for this article.

```
`bash
```

```
? Project name: » my-project
```

```
`
```

2. Select **Vue** as the framework. It will create a Vue 3 project.

```
`bash
```

```
? Select a framework: » - Use arrow-keys. Return to submit.
```

```
Vanilla
```

```
Vue
```

```
React
```

```
Preact
```

```
Lit
```

```
Svelte
```

```
Others
```

```
`
```

3. Choose **JavaScript** as the framework variant to build this Vite project using JavaScript and Vue.

```
`bash
```

? Select a variant: » - Use arrow-keys. Return to submit.

JavaScript

TypeScript

Customize with create-vue ↗

Nuxt ↗

,

4. Upon completing the aforementioned steps to create the `my-project`, run the following command to install its dependencies:

```
`bash
```

```
cd my-project
```

```
npm install
```

,

or

```
`bash
```

```
cd my-project
```

```
yarn install
```

,

Now that `my-project` is ready to run with default settings, let's add Syncfusion components to the project.

### Add Syncfusion Vue packages

Syncfusion Vue component packages are available at [npmjs.com](https://www.npmjs.com). To use Syncfusion Vue components in the project, install the corresponding npm package.

This article uses the [Vue In-place Editor component](#) as an example. To use the Vue In-place Editor component in the project, the `@syncfusion/ej2-vue-inplace-editor` package needs to be installed using the following command:

```
`bash
```

```
npm install @syncfusion/ej2-vue-inplace-editor --save
```

,

or

```
`bash
```

```
yarn add @syncfusion/ej2-vue-inplace-editor
```

,

### Import Syncfusion CSS styles

You can import themes for the Syncfusion Vue component in various ways, such as using CSS or SASS styles from npm packages, CDN, [CRG](#) and [Theme Studio](#). Refer to [themes topic](#) to know more about built-in themes and different ways to refer to themes in a Vue project.

In this article, `Material` theme is applied using CSS styles, which are available in installed packages. The necessary `Material` CSS styles for the In-place Editor component and its dependents were imported into the `<style>` section of `src/App.vue` file.

### ~/SRC/APP.VUE

```
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-buttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-calendars/styles/material.css";
@import "../node_modules/@syncfusion/ej2-dropdowns/styles/material.css";
@import "../node_modules/@syncfusion/ej2-inputs/styles/material.css";
@import "../node_modules/@syncfusion/ej2-lists/styles/material.css";
@import "../node_modules/@syncfusion/ej2-navigations/styles/material.css";
@import "../node_modules/@syncfusion/ej2-popups/styles/material.css";
@import "../node_modules/@syncfusion/ej2-richtexteditor/styles/material.css";
@import "../node_modules/@syncfusion/ej2-splitbuttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-inplace-editor/styles/material.css";
</style>
```

The order of importing CSS styles should be in line with its dependency graph.

### Add Syncfusion Vue component

Follow the below steps to add the Vue In-place Editor component using `Composition API` or `Options API`:

1.First, import and register the In-place Editor component in the `script` section of the `src/App.vue` file. If you are using the `Composition API`, you should add the `setup` attribute to the `script` tag to indicate that Vue will be using the `Composition API`.

### COMPOSITION API (~/SRC/APP.VUE)

```
<script setup>
import { InPlaceEditorComponent as EjsInplaceeditor } from "@syncfusion/ej2-vue-inplace-editor";
</script>
```

### OPTIONS API (~/SRC/APP.VUE)

```
<script>
import { InPlaceEditorComponent } from "@syncfusion/ej2-vue-inplace-editor";
//Component registration
export default {
  name: "App",
  components: {
    "ejs-inplaceeditor": InPlaceEditorComponent,
  }
}
</script>
```

2.In the `template` section, define the In-place Editor component with the `dataSource` property and column definitions.

**~/SRC/APP.VUE**

```
<template>
<div id="app">
<ejs-inplaceeditor id="inplace_editor" type="Text" value="Andrew"
:model="model">
</ejs-inplaceeditor>
</div>
</template>
```

Here is the summarized code for the above steps in the **src/App.vue** file:

**COMPOSITION API (~/SRC/APP.VUE)**

```
<template>
<div id="app">
<ejs-inplaceeditor id="inplace_editor" type="Text" value="Andrew"
:model="data[0].model">
</ejs-inplaceeditor>
</div>
</template>
<script setup>
import { InPlaceEditorComponent as EjsInplaceeditor } from "@syncfusion/ej2-
vue-inplace-editor";
const data = [{ model: {
placeholder: 'Enter employee name'}}]
</script>
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-buttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-calendars/styles/material.css";
@import "../node_modules/@syncfusion/ej2-dropdowns/styles/material.css";
@import "../node_modules/@syncfusion/ej2-inputs/styles/material.css";
@import "../node_modules/@syncfusion/ej2-lists/styles/material.css";
@import "../node_modules/@syncfusion/ej2-navigations/styles/material.css";
@import "../node_modules/@syncfusion/ej2-popups/styles/material.css";
@import "../node_modules/@syncfusion/ej2-
richtexteditor/styles/material.css";
@import "../node_modules/@syncfusion/ej2-splitbuttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-inplace-
editor/styles/material.css";
</style>
```

**OPTIONS API (~/SRC/APP.VUE)**

```
<template>
<div id="app">
<ejs-inplaceeditor id="inplace_editor" type="Text" value="Andrew"
:model="model">
</ejs-inplaceeditor>
</div>
</template>
<script>
import { InPlaceEditorComponent } from "@syncfusion/ej2-vue-inplace-editor";
export default {
name: "App",
```

```
components: {  
  "ejs-inplaceeditor": InPlaceEditorComponent  
},  
data: function () {  
  return {  
    model: {  
      placeholder: 'Enter employee name'  
    },  
  };  
}  
</script>  
<style>  
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";  
@import "../node_modules/@syncfusion/ej2-buttons/styles/material.css";  
@import "../node_modules/@syncfusion/ej2-calendars/styles/material.css";  
@import "../node_modules/@syncfusion/ej2-dropdowns/styles/material.css";  
@import "../node_modules/@syncfusion/ej2-inputs/styles/material.css";  
@import "../node_modules/@syncfusion/ej2-lists/styles/material.css";  
@import "../node_modules/@syncfusion/ej2-navigations/styles/material.css";  
@import "../node_modules/@syncfusion/ej2-popups/styles/material.css";  
@import "../node_modules/@syncfusion/ej2-richtexteditor/styles/material.css";  
@import "../node_modules/@syncfusion/ej2-splitbuttons/styles/material.css";  
@import "../node_modules/@syncfusion/ej2-vue-inplace-editor/styles/material.css";  
</style>
```

### Run the project

To run the project, use the following command:

```
`bash
```

```
npm run dev
```

```
,
```

or

```
`bash
```

```
yarn run dev
```

```
,
```

The output will appear as follows:



### Configure DropDownList in Vue3

You can render DropDownList by changing the type property as DropDownList and configure its properties and methods using the model property.

In the following sample, type and model values are configured to render the DropDownList component.

#### **COMPOSITION API (~SRC/APP.VUE)**

```
<template>
<div id="app">
<ejs-inplaceeditor
id="gender"
type="DropDownList"
mode="Inline"
value="Male"
:model="data[0].dropdownModel"
></ejs-inplaceeditor>
</div>
</template>
<script setup>
import { InPlaceEditorComponent as EjsInplaceeditor } from "@syncfusion/ej2-
vue-inplace-editor";
const data = [{ dropdownModel: {
dataSource: ["Male", "Female"],
placeholder: "Select gender",}}]
</script>
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-buttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-calendars/styles/material.css";
@import "../node_modules/@syncfusion/ej2-dropdowns/styles/material.css";
@import "../node_modules/@syncfusion/ej2-inputs/styles/material.css";
@import "../node_modules/@syncfusion/ej2-lists/styles/material.css";
@import "../node_modules/@syncfusion/ej2-navigations/styles/material.css";
@import "../node_modules/@syncfusion/ej2-popups/styles/material.css";
@import "../node_modules/@syncfusion/ej2-
richtexteditor/styles/material.css";
@import "../node_modules/@syncfusion/ej2-splitbuttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-inplace-
editor/styles/material.css";
</style>
```

#### **OPTIONS API (~SRC/APP.VUE)**

```
<template>
<div id="app">
<ejs-inplaceeditor
id="gender"
type="DropDownList"
mode="Inline"
value="Male"
:model="dropdownModel"
></ejs-inplaceeditor>
</div>
</template>
<script>
```

```

import { InPlaceEditorComponent } from "@syncfusion/ej2-vue-inplace-editor";
export default {
  name: "App",
  components: {
    "ejs-inplaceeditor": InPlaceEditorComponent,
  },
  data: function () {
    return {
      dropdownModel: {
        dataSource: ["Male", "Female"],
        placeholder: "Select gender",
      },
    };
  },
};
</script>
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-buttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-calendars/styles/material.css";
@import "../node_modules/@syncfusion/ej2-dropdowns/styles/material.css";
@import "../node_modules/@syncfusion/ej2-inputs/styles/material.css";
@import "../node_modules/@syncfusion/ej2-lists/styles/material.css";
@import "../node_modules/@syncfusion/ej2-navigations/styles/material.css";
@import "../node_modules/@syncfusion/ej2-popups/styles/material.css";
@import "../node_modules/@syncfusion/ej2-richtexteditor/styles/material.css";
@import "../node_modules/@syncfusion/ej2-splitbuttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-inplace-editor/styles/material.css";
</style>

```

### Integrate DatePicker in Vue3

You can render DatePicker by changing the type property as Date and also configure its properties and methods using model property.

In the below sample, type and model values are configured to render the DatePicker component.

#### COMPOSITION API (~SRC/APP.VUE)

```

<template>
<div id="app">
<ejs-inplaceeditor
id="inplace_editor"
type="Date"
:model="data[0].model"
:value="data[0].value"
>
</ejs-inplaceeditor>
</div>
</template>
<script setup>
import { InPlaceEditorComponent as EjsInplaceeditor } from "@syncfusion/ej2-vue-inplace-editor";
const data = [{ value: new Date("04/12/2018"),
model: {

```

```

showTodayButton: true}
}]
</script>
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-buttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-calendars/styles/material.css";
@import "../node_modules/@syncfusion/ej2-dropdowns/styles/material.css";
@import "../node_modules/@syncfusion/ej2-inputs/styles/material.css";
@import "../node_modules/@syncfusion/ej2-lists/styles/material.css";
@import "../node_modules/@syncfusion/ej2-navigations/styles/material.css";
@import "../node_modules/@syncfusion/ej2-popups/styles/material.css";
@import "../node_modules/@syncfusion/ej2-richtexteditor/styles/material.css";
@import "../node_modules/@syncfusion/ej2-splitbuttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-inplace-editor/styles/material.css";
</style>

```

### **OPTIONS API (~SRC/APP.VUE)**

```

<template>
<div id="app">
<ejs-inplaceeditor
id="inplace_editor"
type="Date"
:model="model"
:value="value"
>
</ejs-inplaceeditor>
</div>
</template>
<script>
import { InPlaceEditorComponent } from "@syncfusion/ej2-vue-inplace-editor";
export default {
name: "App",
components: {
"ejs-inplaceeditor": InPlaceEditorComponent,
},
data: function () {
return {
value: new Date("04/12/2018"),
model: {
showTodayButton: true,
},
};
},
};
</script>
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-buttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-calendars/styles/material.css";
@import "../node_modules/@syncfusion/ej2-dropdowns/styles/material.css";
@import "../node_modules/@syncfusion/ej2-inputs/styles/material.css";
@import "../node_modules/@syncfusion/ej2-lists/styles/material.css";

```



```
@import "../node_modules/@syncfusion/ej2-navigations/styles/material.css";
@import "../node_modules/@syncfusion/ej2-popups/styles/material.css";
@import "../node_modules/@syncfusion/ej2-richtexteditor/styles/material.css";
@import "../node_modules/@syncfusion/ej2-splitbuttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-inplace-editor/styles/material.css";
</style>
```

## Components in Vue Inplace editor component

In-place Editor renders various components based on the [type](#) property and it have built-in and injectable component. To use injectable components, inject the required modules into **In-place Editor**. By default, the [type](#) property set to **Text** and render the **TextBox**.

The following table explains Injectable components module name and built-in components and their types.

Injectable Components		Built in Components	
-----		-----	
<a href="#">AutoComplete</a>	(AutoComplete)	<a href="#">TextBox</a>	(Text)
<a href="#">ComboBox</a>	(ComboBox)	<a href="#">DatePicker</a>	(Date)
<a href="#">MultiSelect</a>	(MultiSelect)	<a href="#">DateTimePicker</a>	(DateTime)
<a href="#">TimePicker</a>	(Time)	<a href="#">DropDownList</a>	(DropDownList)
<a href="#">DateRangePicker</a>	(DateRange)	<a href="#">MaskedTextBox</a>	(Mask)
<a href="#">Slider</a>	(Slider)	<a href="#">NumericTextBox</a>	(Numeric)
<a href="#">Rte</a>	(RTE)		
<a href="#">ColorPicker</a>	(Color)		

In the following sample, built-in and injectable based In-place Editor components are rendered.

### APP.VUE

```
<template>
<div id="app">
  <div id='container'>
    <h3> Built-in Components </h3>
    <table class="table-section">
      <tr>
        <td class="col-lg-6 col-md-6 col-sm-6 col-xs-6 control-title"> DatePicker </td>
        <td class="col-lg-6 col-md-6 col-sm-6 col-xs-6">
          <ejs-inplaceeditor id="date" mode="Inline" type="Date" :model="dateModel" :value="datePickerValue">
            </ejs-inplaceeditor>
        </td>
      </tr>
      <tr>
        <td class="col-lg-6 col-md-6 col-sm-6 col-xs-6 control-title"> DateTimePicker </td>
```

```

        <td class="col-lg-6 col-md-6 col-sm-6 col-xs-6">
            <ejs-inplaceeditor id="dateTime" mode="Inline"
type="DateTime" :model="datetimeModel" :value="dateTimeValue">
                </ejs-inplaceeditor>
            </td>
        </tr>
    </tr>
    <tr>
        <td class="col-lg-6 col-md-6 col-sm-6 col-xs-6 control-
title"> DropDownList </td>
        <td class="col-lg-6 col-md-6 col-sm-6 col-xs-6">
            <ejs-inplaceeditor id="dropDowns" mode="Inline"
type="DropDownList" :model="dropdownModel" value="Android">
                </ejs-inplaceeditor>
            </td>
        </tr>
    </tr>
    <tr>
        <td class="col-lg-6 col-md-6 col-sm-6 col-xs-6 control-
title"> MaskedTextBox </td>
        <td class="col-lg-6 col-md-6 col-sm-6 col-xs-6">
            <ejs-inplaceeditor id="masked" type="Mask" mode="Inline"
:model="maskModel" value="123-345-678">
                </ejs-inplaceeditor>
            </td>
        </tr>
    </tr>
    <tr>
        <td class="col-lg-6 col-md-6 col-sm-6 col-xs-6 control-
title"> NumericTextBox </td>
        <td class="col-lg-6 col-md-6 col-sm-6 col-xs-6">
            <ejs-inplaceeditor id="numeric" type="Numeric"
mode="Inline" :model="numericModel" value=10>
                </ejs-inplaceeditor>
            </td>
        </tr>
    </tr>
    <tr>
        <td class="col-lg-6 col-md-6 col-sm-6 col-xs-6 control-
title"> TextBox </td>
        <td class="col-lg-6 col-md-6 col-sm-6 col-xs-6">
            <ejs-inplaceeditor id="textbox" type="Text"
mode="Inline" :model="textModel" value="Andrew">
                </ejs-inplaceeditor>
            </td>
        </tr>
    </table>
    <h3> Injectable Components </h3>
    <table class="table-section">
        <tr>
            <td class="col-lg-6 col-md-6 col-sm-6 col-xs-6 control-
title"> AutoComplete </td>
            <td class="col-lg-6 col-md-6 col-sm-6 col-xs-6">
                <ejs-inplaceeditor id="autoComplete" type="AutoComplete"
mode="Inline" :model="autocompleteModel" value="Android">
                    </ejs-inplaceeditor>
                </td>
            </tr>
        </tr>
        <tr>
            <td class="col-lg-6 col-md-6 col-sm-6 col-xs-6 control-
title"> ColorPicker </td>

```

```

        <td class="col-lg-6 col-md-6 col-sm-6 col-xs-6">
            <ejs-inplaceeditor id="color" type="Color" mode="Inline"
value="#81aefd">
                </ejs-inplaceeditor>
            </td>
        </tr>
        <tr>
            <td class="col-lg-6 col-md-6 col-sm-6 col-xs-6 control-
title"> ComboBox </td>
            <td class="col-lg-6 col-md-6 col-sm-6 col-xs-6">
                <ejs-inplaceeditor id="combobox" type="ComboBox"
mode="Inline" :model="textModel" value="Android">
                    </ejs-inplaceeditor>
                </td>
            </tr>
            <tr>
                <td class="col-lg-6 col-md-6 col-sm-6 col-xs-6 control-
title"> DateRangePicker </td>
                <td class="col-lg-6 col-md-6 col-sm-6 col-xs-6">
                    <ejs-inplaceeditor id="dateRange" type="DateRange"
mode="Inline" :model="dateRangeModel" :value="dateRangeValue">
                        </ejs-inplaceeditor>
                    </td>
                </tr>
                <tr>
                    <td class="col-lg-6 col-md-6 col-sm-6 col-xs-6 control-
title"> MultiSelect </td>
                    <td class="col-lg-6 col-md-6 col-sm-6 col-xs-6">
                        <ejs-inplaceeditor id="multiselect" type="MultiSelect"
mode="Inline" :model="multiselectModel" :value = "multivalue">
                            </ejs-inplaceeditor>
                        </td>
                    </tr>
                    <tr>
                        <td class="col-lg-6 col-md-6 col-sm-6 col-xs-6 control-
title"> RTE </td>
                        <td class="col-lg-6 col-md-6 col-sm-6 col-xs-6">
                            <ejs-inplaceeditor id="rte" type="RTE" :model="rteModel"
mode="Inline" value="<p>Enter your content here</p>">
                                </ejs-inplaceeditor>
                            </td>
                        </tr>
                        <tr>
                            <td class="col-lg-6 col-md-6 col-sm-6 col-xs-6 control-
title"> Slider </td>
                            <td class="col-lg-6 col-md-6 col-sm-6 col-xs-6">
                                <ejs-inplaceeditor id="slider" type="Slider"
mode="Inline" value=20>
                                    </ejs-inplaceeditor>
                                </td>
                            </tr>
                            <tr>
                                <td class="col-lg-6 col-md-6 col-sm-6 col-xs-6 control-
title"> TimePicker </td>
                                <td class="col-lg-6 col-md-6 col-sm-6 col-xs-6">
                                    <ejs-inplaceeditor id="time" type="Time" mode="Inline"
:model="timeModel" :value="timeValue">

```

```

        </ejs-inplaceeditor>
      </td>
    </tr>
  </table>
</div>
</div>
</template>
<script>
import Vue from 'vue';
import { InPlaceEditorPlugin, TimePicker, DateRangePicker, Rte, MultiSelect,
AutoComplete, ComboBox, ColorPicker, Slider } from '@syncfusion/ej2-vue-
inplace-editor';
Vue.use(InPlaceEditorPlugin);
export default {
  name: app,
  data() {
    let frameWorkList: string[] = ['Android', 'JavaScript', 'jQuery',
'TypeScript', 'Angular', 'React', 'Vue', 'Ionic'];
    return {
      dateModel: {
        placeholder: 'Select date'
      },
      datetimeModel: {
        placeholder: 'Select dateTime'
      },
      dropdownModel: {
        placeholder: 'Select frameWorks',
        dataSource: frameWorkList
      },
      maskModel: {
        mask: '000-000-000'
      },
      numericModel: {
        placeholder: 'Enter Number'
      },
      textModel: {
        placeholder: 'Enter Some Text'
      },
      dateRangeModel: {
        placeholder: 'Select date'
      },
      autocompleteModel: {
        placeholder: 'Select frameWorks',
        dataSource: frameWorkList
      },
      rteModel: {
        placeholder: 'Enter your content here'
      },
      timeModel: {
        placeholder: 'Select time'
      },
      multiselectModel: {
        placeholder: 'Select frameWorks',
        dataSource: frameWorkList
      },
      datePickerValue: new Date('11/23/2018'),
      timeValue: new Date('11/23/2018 12:00 PM'),
    };
  }
};

```

```

        dateTimeValue: new Date('11/23/2018 12:30 PM'),
        dateRangeValue: [new Date('11/12/2018'), new
Date('11/15/2018')],
        multivalue : ['Android']
    }
},
provide:{
    "inplaceeditor":[TimePicker, DateRangePicker , Rte, MultiSelect,
AutoComplete, ComboBox, ColorPicker, Slider ]
}
}
</script>
<style>
@import
"https://ej2.syncfusion.com/vue/documentation/node\_modules/@syncfusion/ej2-
base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-buttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-calendars/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-dropdowns/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-inputs/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-lists/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-
navigations/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-popups/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-
richtexteditor/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-
splitbuttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-inplace-
editor/styles/material.css";
#loader {
color: #008cff;
height: 40px;
left: 45%;
position: absolute;
top: 45%;
width: 30%;
}
body {
padding: 20px 0
}
.control-title {
font-weight: 600;
padding-right: 20px;
}
td {
height: 80px;
}
trtd:first-child {
text-align: right;
}
trtd:last-child {
text-align: left;
}
.table-section {
margin: 0 auto;
}

```

```
h3 {  
  text-align: center;  
  font-size: 24px;  
}  
</style>
```

{% previewsample "page.domainurl/code-snippet/in-place-editor/getting-started-cs2" %}

### Model configuration

Components properties and events can be customized using the In-place Editor [model](#) property.

In the following code, the [type](#) defined as the `Date` and `DatePicker` properties are configured through [model](#) property to customize the [DatePicker](#) component at In-place Editor.

```
`ts
```

```
model: {  
  showTodayButton: true,  
  placeholder: 'Select Date'  
}  
,
```

```
[src/app/app.vue]
```

```
,
```

```
<template>
```

```
<div id="app">
```

```
<ejs-inplaceeditor id="inplace_editor" type="Date" :value="datepickerValue" :model="model">
```

```
</ejs-inplaceeditor>
```

```
</div>
```

```
</template>
```

```
<script>
```

```
import Vue from 'vue';
```

```
import { InPlaceEditorPlugin } from '@syncfusion/ej2-vue-inplace-editor';
```

```
Vue.use(InPlaceEditorPlugin);
```

```
export default {
```

```
  name: 'app',
```

```
  data () {
```

```
    return {
```

```
      datepickerValue: new Date('04/12/2018');
```

```
      dateModel: {
```

```
        showTodayButton: true,
```

```
placeholder: 'Select Date'
```

```
},
```

```
}
```

```
}
```

```
}
```

```
</script>
```

```
,
```

See Also

- [HTML5 components](#)

## Configuration in Vue Inplace editor component

### Rendering modes

This section explains the supported rendering modes of the In-place Editor. Possible Rendering modes are as follows.

- Popup
- Inline

By default, **Popup** mode will be rendered, when opening an editor.

- For **Popup** mode, editable container displays as like tooltip or popover above the element.
- For **Inline** mode, editable container displays as instead of the element. To render **Inline** mode while opening the editor, specify **mode** as **Inline**.

In the following sample, the In-place Editor renders with **Inline** mode. You can dynamically switch into another mode by changing the drop-down item value.

### **APP.VUE**

```
<template>
  <div id="app">
    <table class="table-section">
      <tr>
        <td>Mode</td>
        <td>
          <ejs-dropdownlist ref="editorMode" id="editorMode"
:dataSource='dataPlace' :change='changeEditorMode' :value='dataValue'
:fields='placeFields'>
          </ejs-dropdownlist>
        </td>
      </tr>
      <tr>
        <td class="sample-td">Enter your Name</td>
        <td class="sample-td">
```

```

        <ejs-inplaceeditor ref="editObj" id="inplace_editor"
mode="Inline" type="Text" value="Andrew" submitOnEnter= "true"
:model="textModel">
        </ejs-inplaceeditor>
    </td>
</tr>
</table>
</div>
</template>
<script>
import Vue from 'vue';
import { InPlaceEditorPlugin } from '@syncfusion/ej2-vue-inplace-editor';
import { DropDownListPlugin } from "@syncfusion/ej2-vue-dropdowns";
Vue.use(InPlaceEditorPlugin);
Vue.use(DropDownListPlugin);
export default {
    name: 'app',
    data () {
        return {
            textModel: {
                placeholder: 'Enter Some Name'
            },
            dataValue: 'inline',
            placeFields: { text: 'mode', value: 'id' },
            dataPlace: [{ id: 'inline', mode: 'Inline' }, { id: 'popup',
mode: 'Popup' } ]},
        },
        mounted: function() {
            this.editObj = this.$refs.editObj.ej2Instances;
        },
        methods: {
            changeEditorMode: function(args) {
                var editMode = this.$refs.editorMode.$el.value;
                this.editObj.mode = editMode;
                this.editObj.dataBind();
            }
        }
    }
}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-buttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-calendars/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-dropdowns/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-inputs/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-lists/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-
navigations/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-popups/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-
richtexteditor/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-
splitbuttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-inplace-
editor/styles/material.css";
.table-section {

```



```

        margin: 0 auto;
    }
    tr td:first-child {
        text-align: right;
        padding-right: 20px;
    }
    .sample-td {
        padding-top: 10px;
        min-width: 230px;
        height: 100px;
    }
}
</style>

```

{% previewsample "page.domainurl/code-snippet/in-place-editor/getting-started-cs3" %}

### *Pop-up customization*

In-place Editor popup mode can be customized by using the [title](#) and [model](#) properties in [popupSettings](#) API.

Popup mode rendered by using the Essential JS 2 Tooltip component, so you can use tooltip properties and events to customize the behavior of popup via the [model](#) property of [popupSettings](#) API.

For more details, refer the tooltip documentation [section](#).

In the following sample, popup [title](#) and [position](#) customized using the [popupSettings](#) property. All possible tooltip position data configured in the drop-down, if we change drop down item, selected value bound to [model](#) property and applied it to [Tooltip](#) component. **Tooltip** have following position options.

- TopLeft
- TopCenter
- TopRight
- BottomLeft
- BottomCenter
- BottomRight
- LeftTop
- LeftCenter
- LeftBottom
- RightTop
- RightCenter
- RightBottom

### **APP.VUE**

```

<template>
<div id="app">
    <table class="table-section">
        <tr>
            <td>Position</td>
            <td>
                <ejs-dropdownlist ref="editorMode" id="editorMode"
                :dataSource='dataPlace' :change='changeEditorMode' :value='dataValue'
                :fields='placeFields'>
            </ejs-dropdownlist>

```

```

        </td>
      </tr>
      <tr>
        <td class="edit-heading sample-td">Enter your Name</td>
        <td class="sample-td">
          <ejs-inplaceeditor ref="editObj" id="inplace_editor"
mode="Popup" type="Text" value="Andrew" submitOnEnter= "true"
:model="textModel" :popupSettings="textpopupsettings">
            </ejs-inplaceeditor>
          </td>
        </tr>
      </table>
    </div>
  </template>
  <script>
import Vue from 'vue';
import { InPlaceEditorPlugin } from '@syncfusion/ej2-vue-inplace-editor';
import { DropDownListPlugin } from "@syncfusion/ej2-vue-dropdowns";
Vue.use(InPlaceEditorPlugin);
Vue.use(DropDownListPlugin);
export default {
  name: 'app',
  data () {
    return {
      textModel: {
        placeholder: 'Enter Some Name',
      },
      textpopupsettings: {
        model: {
          position: 'BottomCenter'
        }
      },
      dataValue: 'BottomCenter',
      placeFields: { text: 'mode', value: 'id' },
      dataPlace: [{ id: 'TopLeft', mode: 'TopLeft' }, { id:
'TopCenter', mode: 'TopCenter' }, { id: 'TopRight', mode: 'TopRight' }, { id:
'BottomLeft', mode: 'BottomLeft' }, { id: 'BottomCenter', mode:
'BottomCenter' }, { id: 'BottomRight', mode: 'BottomRight' }, { id:
'LeftTop', mode: 'LeftTop' }, { id: 'LeftCenter', mode: 'LeftCenter' }, { id:
'LeftBottom', mode: 'LeftBottom' }, { id: 'RightTop', mode: 'RightTop' }, {
id: 'RightCenter', mode: 'RightCenter' }, { id: 'RightBottom', mode:
'RightBottom' } ],
    }
  },
  mounted: function() {
    this.editObj = this.$refs.editObj.ej2Instances;
  },
  methods: {
    changeEditorMode: function(args) {
      var positions = this.$refs.editorMode.$el.value;
      this.$refs.editObj.ej2Instances.popupSettings.model.position =
positions;
      this.editObj.dataBind();
    }
  }
}
  </script>

```

```

<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-buttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-calendars/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-dropdowns/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-inputs/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-lists/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-
navigations/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-popups/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-
richtexteditor/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-
splitbuttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-inplace-
editor/styles/material.css";
.table-section {
    margin: 0 auto;
}
.sample-td {
    padding-top: 150px;
}
.edit-heading {
    padding-right: 20px;
}
</style>

```

{% previewsample "page.domainurl/code-snippet/in-place-editor/getting-started-cs4" %}

### Event actions for editing

The event action of the editor that enable in the edit mode based on the [editableOn](#) property, by default **Click** is assigned, the following options are also supported.

- **Click:** The editor will be opened as single click actions.
- **DbClick:** The editor will be opened as double-click actions and it is not applicable for edit icon.
- **EditIconClick:** Disables the editing of event action of input and allows user to edit only through edit icon.

In-place Editor get focus by pressing the **tab** key from previous focusable DOM element and then by pressing **enter** key, the editor will be opened.

In the following sample, when switching drop-down item, the selected value assigned to the [editableOn](#) property. If you changed to **DbClick**, the editor will open when making a double click on the input.

### APP.VUE

```

<template>
<div id="app">
    <table class="table-section">
        <tr>
            <td>EditableOn</td>
            <td>

```

```

        <ejs-dropdownlist ref="editableOn" id="editableon"
:dataSource='editableData' :change='onEditableOn' :value='editableValue'
:fields='editableFields'>
        </ejs-dropdownlist>
    </td>
</tr>
<tr>
    <td class="sample-td">Enter your Name</td>
    <td class="sample-td">
        <ejs-inplaceeditor ref="editObj" id="inplace_editor"
mode="Inline" type="Text" value="Andrew" submitOnEnter= "true"
:model="textModel">
        </ejs-inplaceeditor>
    </td>
</tr>
</table>
</div>
</template>
<script>
import Vue from 'vue';
import { InPlaceEditorPlugin } from '@syncfusion/ej2-vue-inplace-editor';
import { DropDownListPlugin } from "@syncfusion/ej2-vue-dropdowns";
Vue.use(InPlaceEditorPlugin);
Vue.use(DropDownListPlugin);
export default {
    name: 'app',
    data () {
        return {
            textModel: {
                placeholder: 'Enter Some Text'
            },
            editableFields: { text: 'editable', value: 'id' },
            editableData: [{ id: 'Click', editable: 'Click' }, { id:
'DblClick', editable: 'Double Click' }, { id: 'EditIconClick', editable:
'Edit Icon Click' }],
            editableValue: 'Click',
        }
    },
    mounted: function(){
        this.editObj = this.$refs.editObj.ej2Instances;
    },
    methods: {
        onEditableOn: function() {
            var editableOn = this.$refs.editableOn.ej2Instances.value;
            this.$refs.editObj.ej2Instances.editableOn = editableOn;
            this.editObj.dataBind();
        },
    }
}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-buttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-calendars/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-dropdowns/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-inputs/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-lists/styles/material.css";

```

```

@import "../node_modules/@syncfusion/ej2-vue-
navigations/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-popups/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-
richtexteditor/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-
splitbuttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-inplace-
editor/styles/material.css";
.table-section {
    margin: 0 auto;
}
tr td:first-child {
    text-align: right;
    padding-right: 20px;
}
.sample-td {
    padding-top: 10px;
    min-width: 230px;
    height: 100px;
}
</style>

```

{% previewsample "page.domainurl/code-snippet/in-place-editor/getting-started-cs5" %}

#### Action on focus out

Action to be performed when the user clicks outside the container, that means focusing out of editable content and it can be handled by the [actionOnBlur](#) property, by default **Submit** assigned. It also has the following options.

- **Cancel:** Cancels the editing and resets the old content.
- **Submit:** Submits the edited content to the server.
- **Ignore:** No action is performed with this type and allows to edit multiple editors.

In the following sample, when switching drop-down item, the selected value assigned to the **actionOnBlur** property.

#### APP.VUE

```

<template>
<div id="app">
  <table class="table-section">
    <tr>
      <td>ActionOnBlur</td>
      <td>
        <ejs-dropdownlist ref="actions" id="actions"
        :dataSource='blurActionData' :change='onEditableOn' :value='actionValue'
        :fields='actionFields'>
        </ejs-dropdownlist>
      </td>
    </tr>
    <tr>
      <td class="sample-td">Enter your Name</td>
      <td class="sample-td">

```

```

        <ejs-inplaceeditor ref="editObj" id="inplace_editor"
mode="Inline" type="Text" value="Andrew" submitOnEnter= "true"
:model="textModel" >
        </ejs-inplaceeditor>
    </td>
</tr>
</table>
</div>
</template>
<script>
import Vue from 'vue';
import { InPlaceEditorPlugin } from '@syncfusion/ej2-vue-inplace-editor';
import { DropDownListPlugin } from "@syncfusion/ej2-vue-dropdowns";
Vue.use(InPlaceEditorPlugin);
Vue.use(DropDownListPlugin);
export default {
  name: 'app',
  data () {
    return {
      textModel: {
        placeholder: 'Enter Some Text'
      },
      actionFields: { text: 'editable', value: 'id' },
      blurActionData: [{ id: 'Submit', editable: 'Submit' }, {id:
'Cancel', editable: 'Cancel'}, { id:'Ignore', editable: 'Ignore'}],
      actionValue: 'Submit',
    };
  },
  mounted: function(){
    this.editObj = this.$refs.editObj.ej2Instances;
  },
  methods: {
    onEditableOn: function() {
      var actionType = this.$refs.actions.ej2Instances.value;
      this.$refs.editObj.ej2Instances.actionOnBlur = actionType;
      this.editObj.dataBind();
    },
  }
}
</script>

```

{% previewsample "page.domainurl/code-snippet/in-place-editor/getting-started-cs6" %}

### Display modes

By default, In-place Editor input element highlighted with a dotted underline. To remove dotted underline from input element, add `data-underline="false"` attribute at In-place Editor root element.

In the following sample, denotes to indicate intractable and normal display modes with different samples.

### APP.VUE

```

<template>
<div id="app">
<h4>Example of data-underline attribute</h4>
    <table class="table-section">

```

```

        <tr>
        <td>Intractable UI</td>
        <td>
        <ejs-inplaceeditor mode="Inline" :model="textModel" type="Text"
value="Andrew">
        </ejs-inplaceeditor>
        </td>
        </tr>
        <tr>
        <td class="sample-td">Normal UI </td>
        <td class="sample-td">
        <ejs-inplaceeditor mode="Inline" :model="textModel" type="Text"
data-underline="false" value="Andrew">
        </td>
        </tr>
    </table>
</div>
</template>
<script>
import Vue from 'vue';
import { InPlaceEditorPlugin } from '@syncfusion/ej2-vue-inplace-editor';
Vue.use(InPlaceEditorPlugin);
export default {
    name: 'app',
    data () {
        return {
            textModel: {
                placeholder: 'Enter Some Text'
            },
        };
    }
}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-buttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-calendars/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-dropdowns/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-inputs/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-lists/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-
navigations/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-popups/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-
richtexteditor/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-
splitbuttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-inplace-
editor/styles/material.css";
.table-section {
    margin: 0 auto;
}
tr td:first-child {
    text-align: right;
    padding-right: 20px;
}
.sample-td {

```

```
padding-top: 10px;
min-width: 230px;
height: 100px;
}
h4{
  text-align: center;
  font-size: 18px;
}
</style>
```

{% previewsample "page.domainurl/code-snippet/in-place-editor/getting-started-cs7" %}

See Also

- [Disable the editor](#)
- [Animate the editor during popup mode](#)

### Buttons in Vue Inplace editor component

The In-place Editor had an action for save and cancel using buttons. The [saveButton](#) and [cancelButton](#) properties accept the [ButtonModel](#) objects for customizing the save and cancel button properties.

Buttons can be show or hide by sets a Boolean value to the [showButtons](#) property.

Without buttons value will be processed via the following ways.

- **[actionOnBlur](#)**: By clicking out side the editor component get focus out and do action based on this property value.
- **[submitOnEnter](#)**: Pressing **Enter** key it performs the submit action, if this property set to **true**.

In the following sample, the [content](#) and [cssClass](#) properties of **Button** value assigned to the [saveButton](#) and [cancelButton](#) properties to customize its appearance. Also check or uncheck a checkbox buttons render or removed from the editor.

To restrict either save or cancel button rendering into a DOM, simply pass empty object **{}** in the [saveButton](#) or [cancelButton](#) properties.

For more details about buttons, refer this documentation [section](#).

### APP.VUE

```
<template>
<div id="app">
  <table class="table-section">
    <tr>
      <td>ShowButtons:</td>
      <td>
        <ejs-checkbox ref="checkObj" id="showbuttons" checked=true
:change="onChange" :labelPosition="labelPosition" :label="label"></ejs-
checkbox>
      </td>
    </tr>
    <tr>
      <td class="sample-td">Enter your name:</td>
      <td class="sample-td">
```



```

    <ejs-inplaceeditor ref="editObj" mode="Inline" :model="textModel"
    type="Text" value="Andrew" data-underline="false">
    </ejs-inplaceeditor>
  </td>
</tr>
</div>
</template>
<script>
import Vue from 'vue';
import { InPlaceEditorPlugin } from '@syncfusion/ej2-vue-inplace-editor';
import { CheckBoxPlugin } from "@syncfusion/ej2-vue-buttons";
Vue.use(InPlaceEditorPlugin);
Vue.use(CheckBoxPlugin);
export default {
  name: 'app',
  data () {
    return {
      textModel: {
        placeholder: 'Enter Some Text'
      },
      labelPosition: 'Before',
      label: 'show',
    };
  },
  mounted: function() {
    this.editObj = this.$refs.editObj.ej2Instances;
  },
  methods: {
    onChange: function(args) {
      args.checked ? this.editObj.showButtons = true :
this.editObj.showButtons = false
    }
  }
}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-buttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-calendars/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-dropdowns/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-inputs/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-lists/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-
navigations/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-popups/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-
richtexteditor/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-
splitbuttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-inplace-
editor/styles/material.css";
.table-section {
  margin: 0 auto;
}
tr td:first-child {
  text-align: right;
  padding-right: 20px;

```

```

}
.sample-td {
    padding-top: 10px;
    min-width: 230px;
    height: 100px;
}
</style>

```

{% previewsample "page.domainurl/code-snippet/in-place-editor/getting-started-cs1" %}

See Also

- [In-place editor buttons](#)

### Server actions in Vue Inplace editor component

By passing In-place Editor component value to the server, the [primaryKey](#) property value must require, otherwise action not performed for remote data.

If the [URL](#) property value is empty, data passing will handled at local and also the [actionSuccess](#) event will trigger with `null` as argument value.

The following arguments are passed to the server when submit actions perform.

Arguments	Explanations
value	For processing edited value, like DB value updating.
primaryKey	For value mapping to the server, like selecting DB.
name	For field mapping to the server, like DB column field mapping.

Find the following sample server codes for defining models and controller functions to configure processing data.

`C#

```

public class SubmitModel
{
    public string Name { get; set; }
    public string PrimaryKey { get; set; }
    public string Value { get; set; }
}

```

`C#

```

public IEnumerable<SubmitModel> UpdateData([FromBody]SubmitModel value)
{
    // User can process data
}

```

```
return value;
```

```
}
```

```
,
```

- Server actions successfully done, the [actionSuccess](#) event will be fired with returned server data.
- If the server is not responding, the [actionFailure](#) event will be fired with data, but value not updated in the Editor.

In the following sample, the `actionSuccess` event will trigger once the value submitted successfully into the server. In this sample, both `actionSuccess` and `actionFailure` were configured and resulted value will be converted to chips.

### APP.VUE

```
<template>
<div class="app">
  <table class="table-section">
    <tr>
      <td>Select frameworks </td>
      <td>
        <ejs-inplaceeditor ref="tagObj" id="inplace_tag_editor"
data-underline='false' mode="Inline" type="MultiSelect" emptyText="Enter
your tags" :url="serviceUrl" name='skill'
          :value="multiValue" :model="selectModel" :actionSuccess=
"onActionSuccess" :actionFailure = "onActionFailure"
:popupSettings="tagPopupSettings" adaptor='UrlAdaptor' primaryKey =
"FrameWork">
          </ejs-inplaceeditor>
        </td>
      </tr>
    </table>
  </div>
</template>
<script>
import Vue from 'vue';
import { InPlaceEditorPlugin, MultiSelect, ActionEventArgs } from
"@syncfusion/ej2-vue-inplace-editor";
import { DropDownListPlugin } from "@syncfusion/ej2-vue-dropdowns";
Vue.use(InPlaceEditorPlugin);
Vue.use(DropDownListPlugin);
export default {
  name: 'app',
  data () {
    let multiData = ['Android', 'JavaScript', 'jQuery',
'TypeScript', 'Angular', 'React', 'Vue', 'Ionic'];
    return {
      selectModel: {
        mode: 'Box',
        dataSource: multiData,
        placeholder: 'Enter your tags'
      },
      serviceUrl: 'https://ej2services.syncfusion.com/development/web-
services/api/Editor/UpdateData',
      multiValue: ['TypeScript', 'JavaScript'],
```

```

    popupSettings: {
      model: { width: 300 }
    },
    tagPopupSettings: {
      model: { width: 'auto' }
    },
  },
  mounted: function() {
    this.tagObj = this.$refs.tagObj.ej2Instances;
    this.chipOnCreate(this.tagObj.value);
  },
  methods: {
    onActionSuccess: function(e) {
      e.value = this.chipCreation(e.value.split(','), true);
    },
    onActionFailure: function(e) {
      e.value = this.chipCreation(e.value.split(','), false);
    },
    chipOnCreate: function() {
      this.tagObj.element.querySelector('.e-editable-value').innerHTML
= this.chipCreation(this.tagObj.value);
    },
    chipCreation: function(data) {
      let value = '<div class="e-chip-list">';
      [].slice.call(data).forEach((val) => {
        value += '<div class="e-chip"> <span class="e-chip-text"> '
+ val + '</span></div>';
      });
      value += '</div>';
      return value;
    },
  },
  provide: {
    "inplaceeditor": [MultiSelect]
  }
}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-buttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-calendars/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-dropdowns/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-inputs/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-lists/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-
navigations/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-popups/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-
richtexteditor/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-
splitbuttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-inplace-
editor/styles/material.css";
.table-section {
  margin: 0 auto;
}

```

```
tr td:first-child {
    text-align: right;
    padding-right: 20px;
}
</style>
```

{% previewsample "page.domainurl/code-snippet/in-place-editor/getting-started-cs22" %}

See Also

- [Indicate the server actions in the editor](#)

### Data binding in Vue Inplace editor component

The Essential JS 2 components load the data either from local data sources or remote data services using the `dataSource` property and it supports the data type of an array or `DataManager`. Also supports different kind of data services such as OData, OData V4, Web API, and data formats such as XML, JSON, JSONP with the help of `DataManager` adaptors.

#### Local

To bind local data to the Essential JS 2 components, you can assign a JavaScript array of object or string to the `dataSource` property. The local data source can also be provided as an instance of the `DataManager`.

#### APP.VUE

```
<template>
<div class="app">
  <table class="table-section">
    <tr>
      <td>Select customer name: </td>
      <td>
        <ejs-inplaceeditor ref="dropObj" id="dropdownEle" mode="Inline"
type="DropDownList" :model="dropdownModel" value="Maria Anders">
        </ejs-inplaceeditor>
      </td>
    </tr>
  </table>
</div>
</template>
<script>
import Vue from 'vue';
import { InPlaceEditorPlugin, MultiSelect, ActionEventArgs } from
"@syncfusion/ej2-vue-inplace-editor";
Vue.use(InPlaceEditorPlugin);
export default {
  name: 'app',
  data () {
    let dropdownData = [
      { Id: '1', Name: 'Maria Anders' },
      { Id: '2', Name: 'Ana Trujillo' },
      { Id: '3', Name: 'Antonio Moreno' },
      { Id: '4', Name: 'Thomas Hardy' },
      { Id: '5', Name: 'Chiristina Berglund' },
      { Id: '6', Name: 'Hanna Moos' }
    ]
  }
}
```

```

    ];
    return {
      dropdownModel: {
        dataSource: dropdownData,
        placeholder: 'Select a customer',
        fields: { text: 'Name' },
      },
    },
  };
},
mounted: function() {
  this.dropObj = this.$refs.dropObj.ej2Instances;
},
}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-buttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-calendars/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-dropdowns/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-inputs/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-lists/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-
navigations/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-popups/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-
richtexteditor/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-
splitbuttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-inplace-
editor/styles/material.css";
.table-section {
  margin: 0 auto;
  margin-top: 45px;
}
tr td:first-child {
  text-align: right;
  padding-right: 20px;
}
</style>

```

{% previewsample "page.domainurl/code-snippet/in-place-editor/getting-started-cs8" %}

### Remote

To bind remote data to the Essential JS 2 Vue components, assign service data as an instance of **DataManager** to the **dataSource** property. To interact with remote data source, provide the endpoint URL.

### OData V4

The OData V4 is an improved version of OData protocols, and the [DataManager] can also retrieve and consume OData V4 services. To fetch data from the server by using **DataManager** with the adaptor property configure as [ODataV4Adaptor].

In the following sample, In-place Editor renders a **DropDownList** component and its **dataSource** property configured for fetching a data from the server by using **ODataV4Adaptor** with **DataManager**.

**APP.VUE**

```

<template>
<div id="app">
  <table class="table-section">
    <tr>
      <td>Select customer name:</td>
      <td>
        <ejs-inplaceeditor ref="dropObj" id="dropdownEle" mode="Inline"
type="DropDownList" value='Maria Anders' :model="dropdownModel">
        </ejs-inplaceeditor>
      </td>
    </tr>
  </table>
</div>
</template>
<script>
import Vue from 'vue';
import { InPlaceEditorPlugin } from "@syncfusion/ej2-vue-inplace-editor";
Vue.use(InPlaceEditorPlugin);
import { DataManager, Query, ODataV4Adaptor } from "@syncfusion/ej2-data";
export default {
  data () {
    return {
      dropdownModel: {
        dataSource: new DataManager({
          url: 'https://services.odata.org/V4/Northwind/Northwind.svc/',
          adaptor: new ODataV4Adaptor,
          crossDomain: true
        }),
        placeholder: "Select a customer",
        query: new Query().from('Customers').select(['ContactName',
'CustomerID']).take(6),
        fields: { text: 'ContactName', value: 'CustomerID' }
      }
    }
  }
}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-buttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-calendars/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-dropdowns/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-inputs/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-lists/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-
navigations/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-popups/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-
richtexteditor/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-
splitbuttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-inplace-
editor/styles/material.css";
.table-section {
  margin: 0 auto;

```

```

        margin-top: 45px;
    }
    tr td:first-child {
        text-align: right;
        padding-right: 20px;
    }
}
</style>

```

{% previewsample "page.domainurl/code-snippet/in-place-editor/getting-started-cs9" %}

#### Web API

Data can fetch from the server by using [DataManager] with the adaptor property configure as [WebApiAdaptor].

In the following sample, In-place Editor render a DropDownList component and its dataSource property configured for fetching a data from the server by using WebApiAdaptor with DataManager.

#### APP.VUE

```

<template>
<div id="app">
  <table class="table-section">
    <tr>
      <td>select customer name:</td>
      <td>
        <ejs-inplaceeditor ref="dropObj" id="dropdownEle" mode="Inline"
type="DropDownList" value='Maria Anders' :model="dropdownModel">
        </ejs-inplaceeditor>
      </td>
    </tr>
  </table>
</div>
</template>
<script>
import Vue from 'vue';
import { InPlaceEditorPlugin } from "@syncfusion/ej2-vue-inplace-editor";
import { DataManager, ODataV4Adaptor, Query } from '@syncfusion/ej2-data';
Vue.use(InPlaceEditorPlugin);
var dm= new DataManager({
  url:
'https://services.odata.org/V4/Northwind/Northwind.svc/Customers',
  adaptor: new ODataV4Adaptor,
  crossDomain: true
});
export default {
  data () {
    return {
      dropdownModel: {
        dataSource: dm,
        placeholder:"Select a customer",
        fields : { text: 'ContactName', value: 'CustomerID' },
        query : new Query().select(['ContactName',
'CustomerID']).take(6),
      }
    }
  },

```



```

    mounted: function() {
      this.dropObj = this.$refs.dropObj.ej2Instances;
    }
  }
</script>
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-buttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-calendars/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-dropdowns/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-inputs/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-lists/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-navigations/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-popups/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-richtexteditor/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-splitbuttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-inplace-editor/styles/material.css";
.table-section {
  margin: 0 auto;
  margin-top: 45px;
}
tr td:first-child {
  text-align: right;
  padding-right: 20px;
}
</style>

```

```
{% previewsample "page.domainurl/code-snippet/in-place-editor/getting-started-cs10" %}
```

### Integration in Vue Inplace editor component

The In-place Editor supports adding HTML5 input components using the [template](#) property. The Template property can be given as either a `string` or a `query selector`.

#### As a string

The HTML element tag can be given as a string for the template property. Here, the input is rendered as an HTML template.

```
`ts
```

```
template: "<div><input type='text' id='name'></input></div>"
```

```
`
```

#### As a selector

The template property also allows getting template content through `query selector`. Here, the input wrapper element 'ID' attribute is specified in the template.

```
`ts
```

```
template: "#date"
```

```
`
```

### As a template

You can render other components inside In-place Editor using Vue template . Follow the below guidelines for using other the components as template in In-place Editor.

Declare a template in the template section of the “.vue” file. An empty object “data” needs to be initialized in the data option of the default export object in script section.

The template function needs to be assigned to the template property of the EJ2 Vue In-place Editor Component.

Template mode, the `value` property not handled by the In-place Editor component. So, before sending a value to the server, you need to modify at [actionBegin](#) event, otherwise, an empty string will pass. In the following template sample, before submitting a data to the server, event argument and `value` property content updated in the `actionBegin` event handler.

### APP.VUE

```
<template>
<div class="app">
  <table class="table-section">
    <tr>
      <td>Select date:</td>
      <td>
        <ejs-inplaceeditor ref="dropObj" id="dropdownEle" mode="Inline"
:template = "Template1" value="1/2/2018" :actionBegin="actionBegin"
actionOnBlur="ignore">
        </ejs-inplaceeditor>
      </td>
    </tr>
  </table>
</div>
</template>
<script>
import Vue from 'vue';
import { InPlaceEditorPlugin } from "@syncfusion/ej2-vue-inplace-editor";
import { DatePickerComponent, DatePickerPlugin } from '@syncfusion/ej2-vue-
calendars';
import { isNullOrUndefined as isNOU } from '@syncfusion/ej2-base';
Vue.component(DatePickerPlugin.name, DatePickerComponent);
Vue.use(InPlaceEditorPlugin);
export default {
  name: 'app',
  data () {
    return {
      Template1: function () {
        return {
          template: Vue.component('DatePickerComponent', {
            template: ' <ejs-datepicker value="1/2/2018"></ejs-
datepicker>',
            data() { return { }; }
          })
        }
      }
    },
    mounted: function() {
```

```

    this.dropObj = this.$refs.dropObj.ej2Instances;
  },
  methods: {
    actionBegin: function(args) {
      var date = (document.getElementsByClassName('e-datepicker')[0]).ej2_instances[0].value;
      if(!(isNOU(date))) {
        this.$refs.dropObj.ej2Instances.value =
date.toLocaleDateString();
        args.data.value = date;
      }
    }
  }
}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-buttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-calendars/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-dropdowns/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-inputs/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-lists/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-
navigations/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-popups/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-
richtexteditor/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-
splitbuttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-inplace-
editor/styles/material.css";
.table-section {
  margin: 0 auto;
  margin-top: 45px;
}
tr td:first-child {
  text-align: right;
  padding-right: 20px;
}
</style>

```

{% previewsample "page.domainurl/code-snippet/in-place-editor/getting-started-cs18" %}

See Also

- [Built-in Components](#)

## Validation in Vue Inplace editor component

In-place Editor component supports validation and it can be achieved by adding rules to the [validationRules](#) property, its child property **key** must be same as [name](#) property, otherwise validation not performed. Submitting data to the server or calling the [validate](#) method validation executed.

### Validation Rules

In-place Editor has following validation rules, which are used to perform validation.

Rules	Description	Example
required	The input element must have any input values	a or 1 or -
email	The input element must have valid email format values	<inplace@syncfusion.com>
url	The input element must have valid url format values	<a href="http://syncfusion.com/">http://syncfusion.com/</a>
date	The input element must have valid date format values	12/25/2019
dateIso	The input element must have valid dateIso format values	2019-12-25
number	The input element must have valid number format values	1.0 or 1
maxLength	Input value must have less than or equal to maxLength character length	if maxLength: 5, [check] is valid and [checking] is invalid
minLength	Input value must have less than or equal to minLength character length	if minLength: 5, [testing] is valid and [test] is invalid
rangeLength	Input value must have value between rangeLength character length	if rangeLength: [4,5], [test] is valid and [key] is invalid
range	Input value must have value between range number	if range: [4,5], [4] is valid and [6] is invalid
max	Input value must have less than or equal to max number	if max: 3, [3] is valid and [4] is invalid
min	Input value must have less than or equal to min number	if min: 4, [5] is valid and [2] is invalid

### Style in Vue Inplace editor component

The following content provides the exact CSS structure that can be used to modify the control's appearance based on the user preference.

#### Customizing the In-place Editor text

Use the following CSS to customize the default In-place Editor's text content properties like font-family, font-size, color and border bottom.

```
`css
/ To change color, font family and font size /
.e-inplaceeditor .e-editable-value-wrapper .e-editable-value {
border-bottom: 2px dotted green;
color: red;
font-size: 12px;
font-family: Segoe UI
}
```

### Customizing the In-place Editor action buttons

Use the following CSS to customize the default In-place Editor's action buttons.

```
`css
/ To change icon color for save button /
.e-inplaceeditor .e-editable-action-buttons .e-btn-save.e-icon-btn .e-btn-icon.e-icons,
.e-inplaceeditor-tip .e-editable-action-buttons .e-btn-save.e-icon-btn .e-btn-icon.e-icons{
color: green;
}
/ To change icon color for cancel button /
.e-inplaceeditor .e-editable-action-buttons .e-btn-cancel.e-icon-btn .e-btn-icon.e-icons, .e-
inplaceeditor-tip .e-editable-action-buttons .e-btn-cancel.e-icon-btn .e-btn-icon.e-icons {
color: red;
}
/ To change background color for save button /
.e-inplaceeditor .e-editable-action-buttons .e-btn-save.e-icon-btn,
.e-inplaceeditor-tip .e-editable-action-buttons .e-btn-save.e-icon-btn {
background-color: antiquewhite;
}
/ To change background color for cancel button /
.e-inplaceeditor .e-editable-action-buttons .e-btn-cancel.e-icon-btn,
.e-inplaceeditor-tip .e-editable-action-buttons .e-btn-cancel.e-icon-btn {
background-color: antiquewhite;
}
`
```

### Accessibility in Vue Inplace editor component

The Inplace editor component followed the accessibility guidelines and standards, including [ADA](#), [Section 508](#), [WCAG 2.2](#) standards, and [WCAG roles](#) that are commonly used to evaluate accessibility.

The accessibility compliance for the Inplace editor component is outlined below.

| Accessibility Criteria | Compatibility |

| -- | -- |

| [WCAG 2.2 Support](#) |  |

| [Section 508 Support](#) |  |

| [Screen Reader Support](#) |  |

| [Right-To-Left Support](#) |  |

| [Color Contrast](#) |  |

| [Mobile Device Support](#) |  |

| [Keyboard Navigation Support](#) |  |

| [Accessibility Checker Validation](#) |  |

| [Axe-core Accessibility Validation](#) |  |

```
<style>
.post .post-content img {
display: inline-block;
margin: 0.5em 0;
}
</style>

<div> - All
features of the component meet the requirement.</div>

<div> - Some features of the component do not meet the requirement.</div>

<div> - The
component does not meet the requirement.</div>
```

### Keyboard interaction

You can use the following key shortcuts to access the Inplace editor without interruptions:

| **Keyboard shortcuts** | **Actions** |

| --- | --- |

| Tab | Helps in focusing the Inplace editor on the page and switching between the consecutive Inplace editor bars. |

| Shift + Tab | Helps in focusing the previous Inplace editor bar element on the Inplace editor. |

### Ensuring accessibility

The Inplace editor component's accessibility levels are ensured through an [accessibility-checker](#) and [axe-core](#) software tools during automated testing.

The accessibility compliance of the Inplace editor component is shown in the following sample. Open the [sample](#) in a new window to evaluate the accessibility of the Inplace editor component with accessibility tools.

```
{% previewsample "https://ej2.syncfusion.com/accessibility/inplace-editor.html" %}
```

See also

- [Accessibility in Syncfusion Vue components](#)

## Localization in Vue Inplace editor component

### Localization

Localization library allows you to localize the default text content of the In-place Editor to different cultures using the [locale](#) property. In-place Editor following keys will be localize based on culture.

| Locale key | en-US (default) |

|-----|-----|

| save | Close |

| cancel | Cancel |

| loadingText | Loading... |

| editIcon | Click to edit |

| editAreaClick | Click to edit |

| editAreaDoubleClick | Double click to edit |

To load translation object in an application use `load` function of `L10n` class. In the following sample, French culture is set to In-place Editor and change the tooltip text.

### APP.VUE

```
<template>
  <div id="app">
    <table class="table-section">
      <tr>
        <td>EditableOn:</td>
        <td>
          <ejs-dropdownlist ref="editableOn" id="editableon"
width='90%' :dataSource='editableData' :change='onEditableOn'
:value='editableValue' :fields='editableFields'>
            </ejs-dropdownlist>
        </td>
      </tr>
      <tr>
        <td class="sample-td">Enter your name:</td>
        <td class="sample-td">
          <ejs-inplaceeditor ref="editObj" id="inplace_editor"
mode="Inline" type="Text" value="Andrew" submitOnEnter= "true"
:model="textModel" locale='fr-BE'>
            </ejs-inplaceeditor>
        </td>
      </tr>
    </table>
```

```

    </div>
</template>
<script>
import Vue from 'vue';
import { InPlaceEditorPlugin } from '@syncfusion/ej2-vue-inplace-editor';
import { DropDownListPlugin } from "@syncfusion/ej2-vue-dropdowns";
import { L10n } from '@syncfusion/ej2-base';
Vue.use(InPlaceEditorPlugin);
Vue.use(DropDownListPlugin);
L10n.load({
  'fr-BE': {
    'inplace-editor': {
      'save': 'enregistrer',
      'cancel': 'Annuler',
      'loadingText': 'Chargement...',
      'editIcon': 'Cliquez pour éditer',
      'editAreaClick': 'Cliquez pour éditer',
      'editAreaDoubleClick': 'Double-cliquez pour éditer'
    }
  }
});
export default {
  name: 'app',
  data () {
    return {
      textModel: {
        placeholder: 'Enter employee name'
      },
      editableFields: { text: 'editable', value: 'id' },
      editableData: [{ id: 'Click', editable: 'Click' }, { id:
'DblClick', editable: 'Double Click' }, { id: 'EditIconClick', editable:
'Edit Icon Click' }],
      editableValue: 'Click',
    };
  },
  mounted: function() {
    this.editObj = this.$refs.editObj.ej2Instances;
  },
  methods: {
    onEditableOn: function(args) {
      var editableOn = this.$refs.editableOn.ej2Instances.value;
      this.editObj.editableOn = editableOn;
      this.editObj.dataBind();
    },
  }
}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-buttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-calendars/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-dropdowns/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-inputs/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-lists/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-
navigations/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-popups/styles/material.css";

```



```

@import "../node_modules/@syncfusion/ej2-vue-richtexteditor/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-splitbuttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-inplace-editor/styles/material.css";
.table-section {
    margin: 0 auto;
}
tr td:first-child {
    text-align: right;
    padding-right: 20px;
}
.sample-td {
    padding-top: 10px;
    min-width: 230px;
    height: 100px;
}
</style>

```

{% previewsample "page.domainurl/code-snippet/in-place-editor/getting-started-cs19" %}

### Right to left

Specifies the direction of the In-place Editor component using the [enableRtl](#) property. For writing systems that require it like Arabic, Hebrew, etc., the direction can be switched to right-to-left.

It will not change based on the locale property.

### APP.VUE

```

<template>
  <div id="app">
    <table class="table-section">
      <tr>
        <td>Enter your name:</td>
        <td>
          <ejs-inplaceeditor ref="editObj" id="editor"
mode="Inline" value= 'Andrew' enableRtl=true>
          </ejs-inplaceeditor>
        </td>
      </tr>
    </table>
  </div>
</template>
<script>
import Vue from 'vue';
import { InPlaceEditorPlugin } from '@syncfusion/ej2-vue-inplace-editor';
Vue.use(InPlaceEditorPlugin);
export default {
  name: 'app',
  data () {
    return {};
  },
  mounted: function() {
    this.editObj = this.$refs.editObj.ej2Instances;
  },
}

```

```

}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-buttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-calendars/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-dropdowns/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-inputs/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-lists/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-
navigations/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-popups/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-
richtexteditor/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-
splitbuttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-inplace-
editor/styles/material.css";
.table-section {
    margin: 0 auto;
}
tr td:first-child {
    text-align: right;
    padding-right: 20px;
}
</style>

```

{% previewsample "page.domainurl/code-snippet/in-place-editor/getting-started-cs20" %}

### Format

Formatting is a way of representing the value in different format. You can format the following mentioned components with its `format` property, when it passed through the In-place Editor [model](#) property.

- [DatePicker](#)
- [DateRangePicker](#)
- [DateTimePicker](#)
- [NumericTextBox](#)
- [Slider](#)
- [TimePicker](#)

### APP.VUE

```

<template>
<div id="app">
    <table class="table-section">
        <tr>
            <td>select date: </td>
            <td>
                <ejs-inplaceeditor ref="dateObj" id="datePickerEle"
mode="Inline" type="Date" :value="datePickerValue" :model="datePickerModel">
                    </ejs-inplaceeditor>
                </td>
            </tr>
        </table>
    </div>

```

```

        </table>
    </div>
</template>
<script>
import Vue from 'vue';
import { InPlaceEditorPlugin } from '@syncfusion/ej2-vue-inplace-editor';
Vue.use(InPlaceEditorPlugin);
export default {
  name: 'app',
  data () {
    return {
      datePickerModel: {
        placeholder: 'Select date',
        format: 'yyyy-MM-dd'
      },
      datePickerValue: new Date('11/23/2018'),
    };
  },
  mounted: function() {
    this.dateObj = this.$refs.dateObj.ej2Instances;
  },
}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-buttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-calendars/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-dropdowns/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-inputs/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-lists/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-navigations/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-popups/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-richtexteditor/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-splitbuttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-inplace-editor/styles/material.css";
.table-section {
  margin: 0 auto;
}
tr td:first-child {
  text-align: right;
  padding-right: 20px;
}
</style>

```

{% previewsample "page.domainurl/code-snippet/in-place-editor/getting-started-cs21" %}

## How To

### Dynamic edit mode in Vue Inplace editor component

At component initial load, if you want to open editor state without interacting In-place Editor input element, it can be achieved by configuring the [enableEditMode](#) property to `true`.

In the following sample, editor opened at initial load and when toggling a checkbox, it will remove or open the editor.

### APP.VUE

```
<template>
<div id="app">
<table class="table-section">
  <tr>
    <td>
      <div>EnableEditMode</div>
    </td>
    <td>
      <div>
        <ejs-checkbox ref="checkObj2" id="editorEnable"
label='Enable' checked= true :change="onChange"></ejs-checkbox>
      </div>
    </td>
  </tr>
  <tr>
    <td class="sample-td">
      <div>Enter your name: </div>
    </td class="sample-td">
    <td>
      <ejs-inplaceeditor ref="editObj" id="inplace_editor"
mode="Inline" type="Text" actionOnBlur='Ignore' enableEditMode= true
value="Andrew" submitOnEnter= "true">
      </ejs-inplaceeditor>
    </td>
  </tr>
</table>
</div>
</template>
<script>
import Vue from 'vue';
import { InPlaceEditorPlugin } from "@syncfusion/ej2-vue-inplace-editor";
import { CheckBoxPlugin } from "@syncfusion/ej2-vue-buttons";
Vue.use(InPlaceEditorPlugin);
Vue.use(CheckBoxPlugin);
export default {
  data () {
    return {
      labelPosition: 'Before',
    };
  },
  mounted: function() {
    this.editObj = this.$refs.editObj.ej2Instances;
  },
  methods: {
    onChange: function(e) {
      this.$refs.editObj.ej2Instances.enableEditMode = e.checked;
    },
  }
}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
```

```

@import "../node_modules/@syncfusion/ej2-vue-buttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-calendars/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-dropdowns/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-inputs/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-lists/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-
navigations/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-popups/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-
richtexteditor/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-
splitbuttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-inplace-
editor/styles/material.css";
.table-section {
    margin: 0 auto;
}
tr td:first-child {
    text-align: right;
    padding-right: 20px;
}
.sample-td {
    padding-top: 10px;
    min-width: 230px;
    height: 100px;
}
</style>

```

```
{% previewsample "page.domainurl/code-snippet/in-place-editor/getting-started-cs17" %}
```

### Disable edit mode in Vue Inplace editor component

The edit mode of In-place Editor can be disabled by setting the [disabled](#) property value to `true`. In the following sample, when check or uncheck the checkbox, In-place Editor component will disable or enable the edit mode.

### APP.VUE

```

<template>
<div id="app">
<table class="table-section">
  <tr>
    <td>
      <div>Disabled</div>
    </td>
    <td>
      <div>
        <ejs-checkbox ref="checkObj2" id="editorEnable"
:change="onChangeEnable" label='Disable'></ejs-checkbox>
      </div>
    </td>
  </tr>
  <tr>
    <td class="sample-td">
      <div>Enter your name: </div>
    </td>
    <td class="sample-td">

```

```

        <ejs-inplaceeditor ref="editObj" id="inplace_editor"
mode="Inline" type="Text" actionOnBlur='Ignore' value="Andrew"
submitOnEnter= "true">
        </ejs-inplaceeditor>
    </td>
</tr>
</table>
</div>
</template>
<script>
import Vue from 'vue';
import { InPlaceEditorPlugin } from "@syncfusion/ej2-vue-inplace-editor";
import { CheckBoxPlugin } from "@syncfusion/ej2-vue-buttons";
Vue.use(InPlaceEditorPlugin);
Vue.use(CheckBoxPlugin);
export default {
    data () {
        return {
            labelPosition: 'Before',
        };
    },
    mounted: function() {
        this.editObj = this.$refs.editObj.ej2Instances;
    },
    methods: {
        onChangeEnable: function(args) {
            var checkObj1 = this.$refs.checkObj2.ej2Instances;
            checkObj1.checked ? this.editObj.disabled = true :
this.editObj.disabled = false;
        },
    }
}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-buttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-calendars/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-dropdowns/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-inputs/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-lists/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-
navigations/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-popups/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-
richtexteditor/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-
splitbuttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-inplace-
editor/styles/material.css";
.table-section {
    margin: 0 auto;
}
tr td:first-child {
    text-align: right;
    padding-right: 20px;
}
.sample-td {

```

```
padding-top: 10px;
min-width: 230px;
height: 100px;
}
</style>
```

```
{% previewsample "page.domainurl/code-snippet/in-place-editor/getting-started-cs16" %}
```

### Custom indication in Vue Inplace editor component

You can add custom indication to unsaved input value by using the [actionSuccess](#) event, when data not submitted to the server.

In this sample, the `actionSuccess` event configured and the `URL` property not included. Then submit button clicked, the current editor value saved into input and data sending to server action prevented due to the `URL` property not configured.

But [actionSuccess](#) event will trigger the handler function with `null` argument values. In handler function data property `primaryKey` value checked, whether it empty or not. If it is empty custom class, added in the `e-value-wrapper` element to customize its styles.

To send input value to local, set the `URL` property as empty.

### APP.VUE

```
<template>
<div id="app">
  <table class="table-section">
    <tr>
      <td>Enter your name: </td>
      <td>
        <ejs-inplaceeditor ref="editObj" id="inplace_editor" mode="Inline"
type="Text" value="Andrew" :actionSuccess="actionSuccess" submitOnEnter=
"true">
          </ejs-inplaceeditor>
        </td>
      </tr>
    </table>
  </div>
</template>
<script>
import Vue from 'vue';
import { InPlaceEditorPlugin } from "@syncfusion/ej2-vue-inplace-editor";
import { isNullOrUndefined as isNOU } from '@syncfusion/ej2-base';
Vue.use(InPlaceEditorPlugin);
export default {
  data () {
    return {
    };
  },
  mounted: function() {
    this.editObj = this.$refs.editObj.ej2Instances;
  },
  methods: {
    actionSuccess: function(e) {
      let pk = e.data['PrimaryKey'];
      if (isNOU(pk) || pk === '') {
```

```

        document.querySelector('.e-editable-value').classList.add('e-
send-error');
    }
}
}
</script>
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-buttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-calendars/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-dropdowns/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-inputs/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-lists/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-
navigations/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-popups/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-
richtexteditor/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-
splitbuttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-inplace-
editor/styles/material.css";
.table-section {
    margin: 0 auto;
}
tr td:first-child {
    text-align: right;
    padding-right: 20px;
}
.e-inplaceeditor .e-editable-value-wrapper .e-editable-value.e-send-error {
    color: red;
}
</style>

```

{% previewsample "page.domainurl/code-snippet/in-place-editor/getting-started-cs15" %}

### Custom animation in Vue Inplace editor component

In popup mode, the In-place Editor rendered with the Essential JS 2 **Tooltip** component. You can use tooltip properties and events to customize the popup by configure properties into the [model](#) property inside the [popupSettings](#) API.

In the following sample, popup animation can be customized by passing animation effect using the [model](#) property and the dynamic animation effect changes configured from the Essential JS 2 Vue **DropDownList** component **change** event.

### APP.VUE

```

<template>
<div id="app">
  <table class="table-section">
    <tr>
      <td>Open Animation:</td>
      <td>

```



```

        <ejs-dropdownlist ref="editorMode" id="editorMode" width='90%'
:dataSource='dataPlace' :change='valueChange' :value='dataValue'
:fields='placeFields'>
        </ejs-dropdownlist>
    </td>
</tr>
<tr>
    <td class="sample-td">Enter your name:</td>
    <td class="sample-td">
        <ejs-inplaceeditor ref="editObj" id="inplace_editor"
mode="Popup" type="Text" value="Andrew" submitOnEnter= "true"
:model="textModel" :popupSettings=textPopupSettings>
        </ejs-inplaceeditor>
    </td>
</tr>
</table>
</div>
</template>
<script>
import Vue from 'vue';
import { InPlaceEditorPlugin } from "@syncfusion/ej2-vue-inplace-editor";
import { DropDownListPlugin } from "@syncfusion/ej2-vue-dropdowns";
Vue.use(InPlaceEditorPlugin);
Vue.use(DropDownListPlugin);
export default {
    data () {
        return {
            placeFields: { text: 'data', value: 'id' },
            dataPlace: [{ id: 'None', data: 'None' }, { id: 'FadeIn', data:
'FadeIn' }, { id: 'FadeZoomIn', data: 'FadeZoomIn' }, { id: 'ZoomIn', data:
'ZoomIn' } ],
            dataValue: 'ZoomIn',
            textModel: {
                placeholder: 'Enter some text'
            },
            textPopupSettings: {
                title: 'Enter Employee Name',
                model: {
                    animation: {
                        open: { effect: 'ZoomIn', duration: 1000, delay: 0 }
                    }
                }
            },
        },
    },
    mounted() {
        this.editObj = this.$refs.editObj.ej2Instances;
    },
    methods: {
        valueChange: function(e) {
            this.$refs.editObj.popupSettings.model.animation.open.effect =
e.value;
            this.$refs.editObj.dataBind();
        }
    }
}
</script>

```

```
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-buttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-calendars/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-dropdowns/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-inputs/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-lists/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-
navigations/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-popups/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-
richtexteditor/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-
splitbuttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-inplace-
editor/styles/material.css";
.table-section {
    margin: 0 auto;
}
tr td:first-child {
    text-align: right;
    padding-right: 20px;
}
.sample-td {
    padding-top: 10px;
    min-width: 230px;
    height: 100px;
}
</style>
```

{% previewsample "page.domainurl/code-snippet/in-place-editor/getting-started-cs14" %}

## Kanban

### Getting Started with the Vue Kanban Component in Vue 2

This article provides a step-by-step guide for setting up a Vue 2 project using [Vue-CLI](#) and integrating the Syncfusion Vue Kanban component

#### Prerequisites

##### [System requirements for Syncfusion Vue UI components](#)

#### Setting up the Vue 2 project

To generate a Vue 2 project using Vue-CLI, use the [vue create](#) command. Follow these steps to install Vue CLI and create a new project:

```
`bash
```

```
npm install -g @vue/cli
```

```
vue create quickstart
```

```
cd quickstart
```

```
npm run serve
```

```
`
```

or

```
`bash
```

```
yarn global add @vue/cli
```

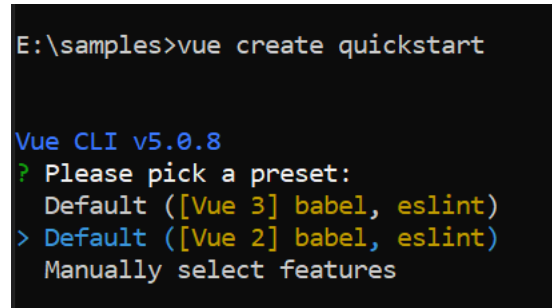
```
vue create quickstart
```

```
cd quickstart
```

```
yarn run serve
```

```
`
```

When creating a new project, choose the option `Default ([Vue 2] babel, eslint)` from the menu.



```
E:\samples>vue create quickstart

Vue CLI v5.0.8
? Please pick a preset:
  Default ([Vue 3] babel, eslint)
> Default ([Vue 2] babel, eslint)
  Manually select features
```

Once the `quickstart` project is set up with default settings, proceed to add Syncfusion components to the project.

#### Add Syncfusion Vue packages

Syncfusion packages are available at [npmjs.com](https://www.npmjs.com). To use Vue components, install the required npm package.

This article uses the [Vue Kanban component](#) as an example. Install the `@syncfusion/ej2-vue-kanban` package by running the following command:

```
`bash
```

```
npm install @syncfusion/ej2-vue-kanban --save
```

```
`
```

or

```
`bash
```

```
yarn add @syncfusion/ej2-vue-kanban
```

```
`
```

The `--save` will instruct NPM to include the Kanban package inside of the `dependencies` section of the `package.json`.

#### Import Syncfusion CSS styles

You can import themes for the Syncfusion Vue component in various ways, such as using CSS or SASS styles from npm packages, CDN, [CRG](#) and [Theme Studio](#). Refer to [themes topic](#) to know more about built-in themes and different ways to refer to themes in a Vue project.

In this article, the **Material** theme is applied using CSS styles, which are available in installed packages. The necessary **Material** CSS styles for the Kanban component and its dependents were imported into the `<style>` section of `src/App.vue` file.

#### ~/SRC/APP.VUE

```
<style>
@import '../node_modules/@syncfusion/ej2-base/styles/material.css';
@import '../node_modules/@syncfusion/ej2-buttons/styles/material.css';
@import '../node_modules/@syncfusion/ej2-layouts/styles/material.css';
@import '../node_modules/@syncfusion/ej2-dropdowns/styles/material.css';
@import '../node_modules/@syncfusion/ej2-inputs/styles/material.css';
@import '../node_modules/@syncfusion/ej2-navigations/styles/material.css';
@import '../node_modules/@syncfusion/ej2-popups/styles/material.css';
@import '../node_modules/@syncfusion/ej2-vue-kanban/styles/material.css';
</style>
```

#### Add Syncfusion Vue component

Follow the below steps to add the Vue Kanban component using **Composition API** or **Options API**:

1\ First, import and register the Kanban component in the `script` section of the `src/App.vue` file.

#### ~/SRC/APP.VUE

```
<script>
import { KanbanComponent, ColumnDirective, ColumnsDirective } from
 '@syncfusion/ej2-vue-kanban';
export default {
  components: {
    'ejs-kanban': KanbanComponent,
    'e-column': ColumnDirective,
    'e-columns': ColumnsDirective
  }
}
</script>
```

2\ In the `template` section, define the Kanban component with [keyField](#) property.

#### ~/SRC/APP.VUE

```
<template>
<div id='app'>
  <ejs-kanban id="kanban" keyField="Status">
    <e-columns>
      <e-column headerText="To Do" keyField="Open"></e-column>
      <e-column headerText="In Progress" keyField="InProgress"></e-column>
      <e-column headerText="Testing" keyField="Testing"></e-column>
      <e-column headerText="Done" keyField="Close"></e-column>
    </e-columns>
  </ejs-kanban>
</div>
</template>
```

### Initialize the Kanban

Add the EJ2 Vue Kanban using `<ejs-kanban>` to the `<template>` section of the `App.vue` file in src directory.

#### ~/SRC/APP.VUE

```
<template>
  <div id='app'>
    <ejs-kanban id="kanban" keyField="Status">
      <e-columns>
        <e-column headerText="To Do" keyField="Open"></e-column>
        <e-column headerText="In Progress" keyField="InProgress"></e-
column>
        <e-column headerText="Testing" keyField="Testing"></e-column>
        <e-column headerText="Done" keyField="Close"></e-column>
      </e-columns>
    </ejs-kanban>
  </div>
</template>
<script>
  import { KanbanComponent, ColumnDirective, ColumnsDirective } from
  '@syncfusion/ej2-vue-kanban';
  export default {
    components: {
      'ejs-kanban': KanbanComponent,
      'e-column': ColumnDirective,
      'e-columns': ColumnsDirective
    }
  }
</script>
<style>
@import '../node_modules/@syncfusion/ej2-base/styles/material.css';
@import '../node_modules/@syncfusion/ej2-buttons/styles/material.css';
@import '../node_modules/@syncfusion/ej2-layouts/styles/material.css';
@import '../node_modules/@syncfusion/ej2-dropdowns/styles/material.css';
@import '../node_modules/@syncfusion/ej2-inputs/styles/material.css';
@import '../node_modules/@syncfusion/ej2-navigations/styles/material.css';
@import '../node_modules/@syncfusion/ej2-popups/styles/material.css';
@import '../node_modules/@syncfusion/ej2-vue-kanban/styles/material.css';
</style>
```

{% previewsample "page.domainurl/code-snippet/kanban/getting-started-empty-cs1" %}

The output will display the kanban header.

### Populating cards

To populate the empty Kanban with cards, define the local JSON data or remote data using the `dataSource` property. To define `dataSource`, the mandatory fields in JSON object should be relevant to `keyField`. In the following example, you can see the cards defined with default fields such as ID, Summary, and Status.

#### ~/SRC/APP.VUE

```
<template>
  <div id="app">
    <ejs-kanban id="kanban" keyField="Status" :dataSource="kanbanData">
```

```

        :cardSettings="cardSettings">
        <e-columns>
            <e-column headerText="To Do" keyField="Open"></e-column>
            <e-column headerText="In Progress" keyField="InProgress"></e-
column>
            <e-column headerText="Testing" keyField="Testing"></e-column>
            <e-column headerText="Done" keyField="Close"></e-column>
        </e-columns>
    </ejs-kanban>
</div>
</template>
<script>
import { KanbanComponent, ColumnDirective, ColumnsDirective } from
'@syncfusion/ej2-vue-kanban';
import { extend } from '@syncfusion/ej2-base';
import { kanbanData } from './datasource.js';
export default {
    components: {
        'ejs-kanban': KanbanComponent,
        'e-column': ColumnDirective,
        'e-ColumnsDirective': ColumnsDirective
    },
    data: function() {
        return {
            kanbanData: extend([], kanbanData, null, true),
            cardSettings: {
                contentField: "Summary",
                headerField: "Id",
            }
        };
    },
}
</script>
<style>
@import '../node_modules/@syncfusion/ej2-base/styles/material.css';
@import '../node_modules/@syncfusion/ej2-buttons/styles/material.css';
@import '../node_modules/@syncfusion/ej2-layouts/styles/material.css';
@import '../node_modules/@syncfusion/ej2-dropdowns/styles/material.css';
@import '../node_modules/@syncfusion/ej2-inputs/styles/material.css';
@import '../node_modules/@syncfusion/ej2-navigations/styles/material.css';
@import '../node_modules/@syncfusion/ej2-popups/styles/material.css';
@import '../node_modules/@syncfusion/ej2-vue-kanban/styles/material.css';
</style>

```

{% previewsample "page.domainurl/code-snippet/kanban/getting-started-key-field-cs2" %}

### Enable swimlane

Swimlane can be enabled by mapping the fields `swimlaneSettings.keyField` to appropriate column name in `dataSource`. This enables the grouping of the cards based on the mapped column values.

### ~/SRC/APP.VUE

```

<template>
    <div id="app">
        <ejs-kanban id="kanban" keyField="Status" :dataSource="kanbanData"
            :cardSettings="cardSettings" :swimlaneSettings="swimlaneSettings">

```

```

        <e-columns>
          <e-column headerText="To Do" keyField="Open"></e-column>
          <e-column headerText="In Progress" keyField="InProgress"></e-
column>
          <e-column headerText="Testing" keyField="Testing"></e-column>
          <e-column headerText="Done" keyField="Close"></e-column>
        </e-columns>
      </ejs-kanban>
    </div>
  </template>
  <script>
import { KanbanComponent, ColumnDirective, ColumnsDirective } from
'@syncfusion/ej2-vue-kanban';
import { extend } from '@syncfusion/ej2-base';
import { kanbanData } from './datasource.js';
export default {
  components: {
    'ejs-kanban': KanbanComponent,
    'e-column': ColumnDirective,
    'e-columns': ColumnsDirective
  },
  data: function() {
    return {
      kanbanData: extend([], kanbanData, null, true),
      cardSettings: {
        contentField: "Summary",
        headerField: "Id",
      },
      swimlaneSettings: {
        keyField: "Assignee"
      }
    };
  },
};
  </script>
  <style>
@import '../node_modules/@syncfusion/ej2-base/styles/material.css';
@import '../node_modules/@syncfusion/ej2-buttons/styles/material.css';
@import '../node_modules/@syncfusion/ej2-layouts/styles/material.css';
@import '../node_modules/@syncfusion/ej2-dropdowns/styles/material.css';
@import '../node_modules/@syncfusion/ej2-inputs/styles/material.css';
@import '../node_modules/@syncfusion/ej2-navigations/styles/material.css';
@import '../node_modules/@syncfusion/ej2-popups/styles/material.css';
@import '../node_modules/@syncfusion/ej2-vue-kanban/styles/material.css';
  </style>

```

{% previewsample "page.domainurl/code-snippet/kanban/getting-started-swimlane-cs1" %}

## Getting Started with the Vue Kanban Component in Vue 3

This article provides a step-by-step guide for setting up a [Vite](#) project with a JavaScript environment and integrating the Syncfusion Vue Kanban component using the [Composition API](#) / [Options API](#).

The **Composition API** is a new feature introduced in Vue.js 3 that provides an alternative way to organize and reuse component logic. It allows developers to write components as functions that use smaller, reusable functions called composition functions to manage their properties and behavior.

The **Options API** is the traditional way of writing Vue.js components, where the component logic is organized into a series of options that define the component's properties and behavior. These options include data, methods, computed properties, watchers, lifecycle hooks, and more.

### Prerequisites

#### [System requirements for Syncfusion Vue UI components](#)

#### Set up the Vite project

A recommended approach for beginning with Vue is to scaffold a project using [Vite](#). To create a new Vite project, use one of the commands that are specific to either NPM or Yarn.

```
`bash
```

```
npm create vite@latest
```

```
,
```

or

```
`bash
```

```
yarn create vite
```

```
,
```

Using one of the above commands will lead you to set up additional configurations for the project as below:

1. Define the project name: We can specify the name of the project directly. Let's specify the name of the project as **my-project** for this article.

```
`bash
```

```
? Project name: » my-project
```

```
,
```

2. Select **Vue** as the framework. It will create a Vue 3 project.

```
`bash
```

```
? Select a framework: » - Use arrow-keys. Return to submit.
```

```
Vanilla
```

```
Vue
```

```
React
```

```
Preact
```

```
Lit
```

```
Svelte
```

```
Others
```

```
,
```



3. Choose `JavaScript` as the framework variant to build this Vite project using JavaScript and Vue.

```
`bash
```

? Select a variant: » - Use arrow-keys. Return to submit.

```
JavaScript
```

```
TypeScript
```

```
Customize with create-vue ↗
```

```
Nuxt ↗
```

```
,
```

4. Upon completing the aforementioned steps to create the `my-project`, run the following command to install its dependencies:

```
`bash
```

```
cd my-project
```

```
npm install
```

```
,
```

```
or
```

```
`bash
```

```
cd my-project
```

```
yarn install
```

```
,
```

Now that `my-project` is ready to run with default settings, let's add Syncfusion components to the project.

#### Add Syncfusion Vue packages

Syncfusion Vue component packages are available at [npmjs.com](https://www.npmjs.com). To use Syncfusion Vue components in the project, install the corresponding npm package.

This article uses the [Vue Kanban component](#) as an example. To use the Vue Kanban component in the project, the `@syncfusion/ej2-vue-kanban` package needs to be installed using the following command:

```
`bash
```

```
npm install @syncfusion/ej2-vue-kanban --save
```

```
,
```

```
or
```

```
`bash
```

```
yarn add @syncfusion/ej2-vue-kanban
```

### Import Syncfusion CSS styles

You can import themes for the Syncfusion Vue component in various ways, such as using CSS or SASS styles from npm packages, CDN, [CRG](#) and [Theme Studio](#). Refer to [themes topic](#) to know more about built-in themes and different ways to refer to themes in a Vue project.

In this article, **Material** theme is applied using CSS styles, which are available in installed packages. The necessary **Material** CSS styles for the Kanban component and its dependents were imported into the `<style>` section of **src/App.vue** file.

#### ~/SRC/APP.VUE

```
<style>
@import '../node_modules/@syncfusion/ej2-base/styles/material.css';
@import '../node_modules/@syncfusion/ej2-buttons/styles/material.css';
@import '../node_modules/@syncfusion/ej2-layouts/styles/material.css';
@import '../node_modules/@syncfusion/ej2-dropdowns/styles/material.css';
@import '../node_modules/@syncfusion/ej2-inputs/styles/material.css';
@import '../node_modules/@syncfusion/ej2-navigations/styles/material.css';
@import '../node_modules/@syncfusion/ej2-popups/styles/material.css';
@import '../node_modules/@syncfusion/ej2-vue-kanban/styles/material.css';
</style>
```

The order of importing CSS styles should be in line with its dependency graph.

### Add Syncfusion Vue component

Follow the below steps to add the Vue Kanban component using **Composition API** or **Options API**:

1. First, import and register the Kanban component and its child directives in the **script** section of the **src/App.vue** file. If you are using the **Composition API**, you should add the **setup** attribute to the **script** tag to indicate that Vue will be using the **Composition API**.

#### COMPOSITION API (~/SRC/APP.VUE)

```
<script setup>
import { KanbanComponent as EjsKanban, ColumnsDirective as
EColumns, ColumnDirective as EColumn } from '@syncfusion/ej2-vue-kanban';
</script>
```

#### OPTIONS API (~/SRC/APP.VUE)

```
<script>
import { KanbanComponent, ColumnsDirective, ColumnDirective } from
"@syncfusion/ej2-vue-kanban";
//Component registration
components: {
  "ejs-kanban": KanbanComponent,
  "e-columns": ColumnsDirective,
  "e-column": ColumnDirective,
}
</script>
```

2. In the `template` section, define the Kanban component with the `dataSource` property and column definitions.

#### ~/SRC/APP.VUE

```
<template>
<ejs-kanban :dataSource='data'>
<e-columns>
<e-column headerText="To Do" keyField="Open"></e-column>
<e-column headerText="In Progress" keyField="InProgress"></e-column>
<e-column headerText="Testing" keyField="Testing"></e-column>
<e-column headerText="Done" keyField="Close"></e-column>
</e-columns>
</ejs-kanban>
</template>
```

3. Declare the values for the `dataSource` property in the `script` section.

#### COMPOSITION API (~/SRC/APP.VUE)

```
<script setup>
const data = [
{
Id: 1,
Status: 'Open',
Summary: 'Analyze the new requirements gathered from the customer.',
Assignee: 'Andrew Fuller'
},
{
Id: 2,
Status: 'InProgress',
Summary: 'Improve application performance',
Assignee: 'Andrew Fuller'
},
{
Id: 3,
Status: 'Close',
Summary: 'Arrange a web meeting with the customer to get new requirements.',
Assignee: 'Janet Leverling'
}]
</script>
```

#### OPTIONS API (~/SRC/APP.VUE)

```
<script>
data: function() {
return {
data: [
{
Id: 1,
Status: 'Open',
Summary: 'Analyze the new requirements gathered from the customer.',
Assignee: 'Andrew Fuller'
},

```

```
{
  Id: 2,
  Status: 'InProgress',
  Summary: 'Improve application performance',
  Assignee: 'Andrew Fuller'
},
{
  Id: 3,
  Status: 'Close',
  Summary: 'Arrange a web meeting with the customer to get new requirements.',
  Assignee: 'Janet Leverling'
}]
}
}
</script>
```

Here is the summarized code for the above steps in the **src/App.vue** file:

#### **COMPOSITION API (~SRC/APP.VUE)**

```
<template>
<ejs-kanban id="kanban" keyField="Status" :dataSource="data"
:cardSettings="cardSettings">
<e-columns>
<e-column headerText="To Do" keyField="Open" ></e-column>
<e-column headerText="In Progress" keyField="InProgress" ></e-column>
<e-column headerText="Testing" keyField="Testing" ></e-column>
<e-column headerText="Done" keyField="Close" ></e-column>
</e-columns>
</ejs-kanban>
</template>
<script setup>
import { KanbanComponent as EjsKanban , ColumnsDirective as EColumns,
ColumnDirective as EColumn } from '@syncfusion/ej2-vue-kanban';
const data = [
{
  Id: 1,
  Status: 'Open',
  Summary: 'Analyze the new requirements gathered from the customer.',
  Assignee: 'Andrew Fuller'
},
{
  Id: 2,
  Status: 'InProgress',
  Summary: 'Improve application performance',
  Assignee: 'Andrew Fuller'
},
{
  Id: 3,
  Status: 'Close',
  Summary: 'Arrange a web meeting with the customer to get new requirements.',
  Assignee: 'Janet Leverling'
}
];
let cardSettings = {
  contentField: "Summary",
```

```

headerField: "Id",
}
</script>
<style>
@import '../node_modules/@syncfusion/ej2-base/styles/material.css';
@import '../node_modules/@syncfusion/ej2-buttons/styles/material.css';
@import '../node_modules/@syncfusion/ej2-layouts/styles/material.css';
@import '../node_modules/@syncfusion/ej2-dropdowns/styles/material.css';
@import '../node_modules/@syncfusion/ej2-inputs/styles/material.css';
@import '../node_modules/@syncfusion/ej2-navigations/styles/material.css';
@import '../node_modules/@syncfusion/ej2-popups/styles/material.css';
@import '../node_modules/@syncfusion/ej2-vue-kanban/styles/material.css';
</style>

```

### OPTIONS API (~SRC/APP.VUE)

```

<template>
<ejs-kanban id="kanban" keyField="Status" :dataSource="data"
:cardSettings="cardSettings">
<e-columns>
<e-column headerText="To Do" keyField="Open" ></e-column>
<e-column headerText="In Progress" keyField="InProgress" ></e-column>
<e-column headerText="Testing" keyField="Testing" ></e-column>
<e-column headerText="Done" keyField="Close" ></e-column>
</e-columns>
</ejs-kanban>
</template>
<script>
import { KanbanComponent, ColumnsDirective, ColumnDirective } from
"@syncfusion/ej2-vue-kanban";
export default {
components: {
"ejs-kanban": KanbanComponent,
"e-columns": ColumnsDirective,
"e-column": ColumnDirective,
},
data: function() {
return {
data: [
{
Id: 1,
Status: 'Open',
Summary: 'Analyze the new requirements gathered from the customer.',
Assignee: 'Andrew Fuller'
},
{
Id: 2,
Status: 'InProgress',
Summary: 'Improve application performance',
Assignee: 'Andrew Fuller'
},
{
Id: 3,
Status: 'Close',
Summary: 'Arrange a web meeting with the customer to get new requirements.',
Assignee: 'Janet Leverling'
}
]
}
}

```

```

    }],
    cardSettings: {
      contentField: "Summary",
      headerField: "Id",
    }
  }
}
}
</script>
<style>
@import '../node_modules/@syncfusion/ej2-base/styles/material.css';
@import '../node_modules/@syncfusion/ej2-buttons/styles/material.css';
@import '../node_modules/@syncfusion/ej2-layouts/styles/material.css';
@import '../node_modules/@syncfusion/ej2-dropdowns/styles/material.css';
@import '../node_modules/@syncfusion/ej2-inputs/styles/material.css';
@import '../node_modules/@syncfusion/ej2-navigations/styles/material.css';
@import '../node_modules/@syncfusion/ej2-popups/styles/material.css';
@import '../node_modules/@syncfusion/ej2-vue-kanban/styles/material.css';
</style>

```

### Run the project

To run the project, use the following command:

```
`bash
```

```
npm run dev
```

```
,
```

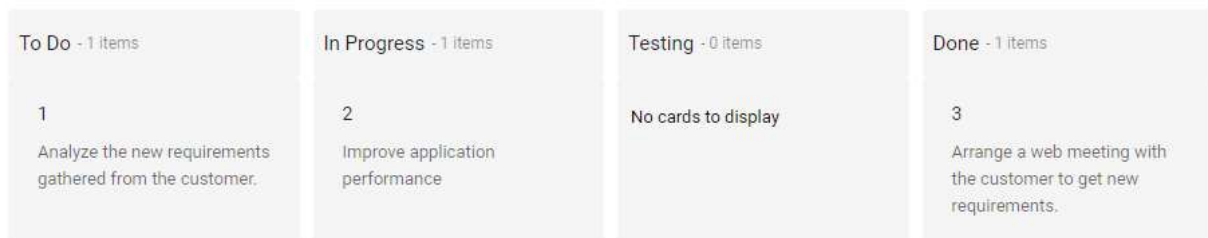
or

```
`bash
```

```
yarn run dev
```

```
,
```

The output will appear as follows:



### Columns in Vue Kanban component

The **Kanban** columns represent the each stage of the process. The column definitions are used as the **dataSource** schema in the Kanban. The Kanban operations such as drag-and-drop, swimlane, and toggle columns are performed based on column definitions.

### Single-key mapping

Kanban columns are categorized by mapping the **key** from the datasource using the **keyField** property. The corresponding **value** in the datasource is mapped inside the columns **keyField**. Based on this categorization, Kanban columns are split on this board.

The **keyField** property is mandatory to render the columns in the Kanban board.

### APP.VUE

```
<template>
  <div id="app">
    <ejs-kanban id="kanban" keyField="Status" :dataSource="kanbanData"
      :cardSettings="cardSettings">
      <e-columns>
        <e-column headerText="To Do" keyField="Open"></e-column>
        <e-column headerText="In Progress" keyField="InProgress"></e-
column>
        <e-column headerText="Testing" keyField="Testing"></e-column>
        <e-column headerText="Done" keyField="Close"></e-column>
      </e-columns>
    </ejs-kanban>
  </div>
</template>
<script>
import Vue from "vue";
import { KanbanPlugin } from '@syncfusion/ej2-vue-kanban';
import { extend } from '@syncfusion/ej2-base';
import { kanbanData } from './datasource.js';
Vue.use(KanbanPlugin);
export default {
  data: function() {
    return {
      kanbanData: extend([], kanbanData, null, true),
      cardSettings: {
        contentField: "Summary",
        headerField: "Id"
      }
    };
  },
};
</script>
<style>
@import '../node_modules/@syncfusion/ej2-base/styles/material.css';
@import '../node_modules/@syncfusion/ej2-buttons/styles/material.css';
@import '../node_modules/@syncfusion/ej2-layouts/styles/material.css';
@import '../node_modules/@syncfusion/ej2-dropdowns/styles/material.css';
@import '../node_modules/@syncfusion/ej2-inputs/styles/material.css';
@import '../node_modules/@syncfusion/ej2-navigations/styles/material.css';
@import '../node_modules/@syncfusion/ej2-popups/styles/material.css';
@import '../node_modules/@syncfusion/ej2-vue-kanban/styles/material.css';
</style>
```

{% previewsample "page.domainurl/code-snippet/kanban/single-key-cs1" %}

### Multi-key mapping

Kanban board allows to render a single column by mapping multiple keys using `keyField` property. In below sample, specified the multiple keys(Open, Validate) to a single column.

#### APP.VUE

```
<template>
  <div id="app">
    <ejs-kanban id="kanban" keyField="Status" :dataSource="kanbanData"
      :cardSettings="cardSettings">
      <e-columns>
        <e-column headerText="To Do" keyField="Open, Validate"></e-
column>
        <e-column headerText="In Progress" keyField="InProgress"></e-
column>
        <e-column headerText="Testing" keyField="Testing"></e-column>
        <e-column headerText="Done" keyField="Close"></e-column>
      </e-columns>
    </ejs-kanban>
  </div>
</template>
<script>
import Vue from "vue";
import { KanbanPlugin } from '@syncfusion/ej2-vue-kanban';
import { extend } from '@syncfusion/ej2-base';
import { kanbanData } from './datasource.js';
Vue.use(KanbanPlugin);
export default {
  data: function() {
    return {
      kanbanData: extend([], kanbanData, null, true),
      cardSettings: {
        contentField: "Summary",
        headerField: "Id"
      }
    };
  },
};
</script>
<style>
@import '../node_modules/@syncfusion/ej2-base/styles/material.css';
@import '../node_modules/@syncfusion/ej2-buttons/styles/material.css';
@import '../node_modules/@syncfusion/ej2-layouts/styles/material.css';
@import '../node_modules/@syncfusion/ej2-dropdowns/styles/material.css';
@import '../node_modules/@syncfusion/ej2-inputs/styles/material.css';
@import '../node_modules/@syncfusion/ej2-navigations/styles/material.css';
@import '../node_modules/@syncfusion/ej2-popups/styles/material.css';
@import '../node_modules/@syncfusion/ej2-vue-kanban/styles/material.css';
</style>
```

{% previewsample "page.domainurl/code-snippet/kanban/multiple-keys-cs1" %}

### Header text

You can provide the column header text of Kanban columns using the `headerText` property. If you have not specified any header text, it will render the header without any text.



### Header template

You can customize the column header with any HTML or CSS element using the `template` property as shown in the following code.

#### APP.VUE

```
<template>
  <div id="app">
    <ejs-kanban id="kanban" keyField="Status" :dataSource="kanbanData"
      :cardSettings="cardSettings" cssClass="kanban-header-template">
      <e-columns>
        <e-column headerText="To Do" keyField="Open"
:template="columnsTemplate"></e-column>
        <e-column headerText="In Progress" keyField="InProgress"
:template="columnsTemplate"></e-column>
        <e-column headerText="Testing" keyField="Testing"
:template="columnsTemplate"></e-column>
        <e-column headerText="Done" keyField="Close"
:template="columnsTemplate"></e-column>
      </e-columns>
    </ejs-kanban>
  </div>
</template>
<script>
import Vue from "vue";
import { KanbanPlugin } from '@syncfusion/ej2-vue-kanban';
import { extend } from '@syncfusion/ej2-base';
import { kanbanData } from './datasource.js';
Vue.use(KanbanPlugin);
export default {
  data: function() {
    return {
      kanbanData: extend([], kanbanData, null, true),
      cardSettings: {
        contentField: "Summary",
        headerField: "Id"
      },
      columnsTemplate: function () {
        return {
          template: Vue.component('columnsTemplate', {
            template: `<div class="header-template-wrap">
              <div :class="getClassName(data)"></div>
              <div class="header-text">{{data.headerText}}</div>
            </div>`,
            data() {
              return {
                data: {}
              };
            },
            methods: {
              getClassName: function(data) {
                return "header-icon e-icons " + data.keyField;
              }
            }
          })
        }
      }
    }
  },
}
```

```

    };
  },
}
</script>
<style>
@import '../node_modules/@syncfusion/ej2-base/styles/material.css';
@import '../node_modules/@syncfusion/ej2-buttons/styles/material.css';
@import '../node_modules/@syncfusion/ej2-layouts/styles/material.css';
@import '../node_modules/@syncfusion/ej2-dropdowns/styles/material.css';
@import '../node_modules/@syncfusion/ej2-inputs/styles/material.css';
@import '../node_modules/@syncfusion/ej2-navigations/styles/material.css';
@import '../node_modules/@syncfusion/ej2-popups/styles/material.css';
@import '../node_modules/@syncfusion/ej2-vue-kanban/styles/material.css';

.kanban-header-template.e-kanban .header-template-wrap {
  display: inline-flex;
  font-size: 15px;
  font-weight: 400;
}

.kanban-header-template.e-kanban .header-template-wrap .header-icon {
  font-family: 'Kanban priority icons';
  margin-top: 3px;
  width: 10%;
}

.kanban-header-template.e-kanban .header-template-wrap .header-text {
  margin-left: 15px;
}

.e-kanban.e-rtl .header-template-wrap .header-text {
  margin-right: 15px;
}

.kanban-header-template.e-kanban .card-header {
  padding-left: 12px;
}

.kanban-header-template.e-kanban .Open::before {
  content: '\e700';
  color: #0251cc;
  font-size: 16px;
}

.kanban-header-template.e-kanban .InProgress::before {
  content: '\e703';
  color: #ea9713;
  font-size: 16px;
}

.kanban-header-template.e-kanban .Review::before {
  content: '\e701';
  color: #8e4399;
  font-size: 16px;
}

.kanban-header-template.e-kanban .Close::before {
  content: '\e702';
}

```

```

color: #f3ba3c;
font-size: 16px;
}

@font-face {
  font-family: 'Kanban priority icons';
  src:
    url(data:application/x-font-ttf;charset=utf-
base64,AAEAAAAKAIAAAwAgT1MvMj1tSfUAAAEoAAAAVmNtYXNlE+dkAAABlAAAADxnbHlmg4w
AgAAADwAAAhQaGvH7sAAADQAAAAANmhoZWEIVQQAAGAAARAAAACRobXR4FAAAAAAAAAAAAA
oG9jYQNeBi4AAAHQAAAAADG1heHABGAfGAAABCAAAACBuYw1lH65UOQAACiWAAALNcG9zdFsyK1E
AAz8AAAAUgABAAAEAAAAAFwEAAAAAAD+AAABAAAAAAAAAAAAAAAAAAAAABQABAAAAQAA7pb8lF8
PPUACwQAAAAANpY0WMAAAAA21jRYwAAAAAD+AP4AAAAACAACAAAAAAAAAAAAAAFAVQACQAAAAA
AgAAAAoACgAAAP8AAAAAAAAAAQAAZAABQAAAokCzAAAAI8CiQLMAAAB6wAyAQgAAAIABQMAAAA
AAAAAAAAAAAAAAAAAAAAAUGZFZABA5wDnAwQAAAAAXAQAAAAAAAAABAAAAAAAAABAAAAAQAAAA
AAAAAABAAAAAQAAAAAAACAAAAAwAAABQAAwABAAAFAAEACgAAAAEAAQAAQAA5wP//wAA5wD//wA
AAEABAAAAAAEAAGADAQAAAAAAmWCBgKSBCgABAAAAAAD+AP4ACEAQwBlAKkAAAEfBw8HIS8HPwc
HwcPByEvBz8HJR8HDwchLwc/BycRHw8hPw8RLw8hDw4CXgcGBQUEAwEBAQEDBAUFBgf+hgYGBQU
AwEBAQEDBAUFBgYCOAYGBQUEAwEBAQEDBAUFBg9yAYGBQUEAwEBAQEDBAUFBgYCOAYGBQUEAwE
AQEDBAUFBg9yAYGBQUEAwEBAQEDBAUFBgbcAQIDBQUHCAkKCgsMDQ0ODQLgDQ4NDQwLcGJCAC
BQMCAQECAwUFBwgJCgoLDA0NDg39IA0ODQ0MCwoKCQgHBQUdAgFDAQEDBAUFBgYHBgUFBAMBAQE
AwQFBQYHBgYFBQQDAQG9AQEDBAUFBgCGBgUFBAMBAQEBAwQFBQYGBwYFBQQDAQG9AQEDBAUFBgY
3gUFBAMBAQEBAwQFBQYHBgYFBQQDAQGz/SANDg0NDAsKCgkIBwUFAwIBAQIDBQUHCAkKCgsMDQ0
OQlgDQ4NDQwLcGJCACFBQMCAQECAwUFBwgJCgoLDA0NDgAABAAAAAAD+AP4AD8AggDUARgAAAE
Bw8PLw4lPw8fBicPDx8PMz8OLxAHNzMFehUPESsBLx9AT8UJREfDyE/DxEvDyEPDgJlCAGBgQ
AgEBAgMEBQCHCAkCJwsMDAwNDgNDAsLCgkICAYFAWMAbQMDQBQUHBwgJCQoLCwwMDA4MDAwLCgg
0g8PDw4PDw8VFBQUEXmTEhUWfhYXfXgYehMSERISEREUEBEREBESERkZGRgXfXcXEA8QEBARERE
FxYVfHwUfHIEfAsXGBkYGRkYGSATEXISEhIRBQMBAGICHBkaGhscGx0UEXmTEXmTExoUFRQVFBu
HBoaGhkYGRKEAgIDGBQVfHXYfXcREREQEREQE8ODv4aAQIDBQUHCAkKCgsMDQ0ODQLgDQ4NDQw
LcGJCACFBQMCAQECAwUFBwgJCgoLDA0NDg39IA0ODQ0MCwoKCQgHBQUdAgJXCQoKCwsMDAwNDAw
CgsJCQgHBgUEAwIBAQIDBQUHCAkJCgsMCw0MDQwLDAoLCQkJBwcGBQOCAGEBAGMEBQYIWQMEBQY
BwgJDg4PERETEXUYfXUTEhAPDgkIBwUFAwEBAGIEBQYHCA0QEBMUfHcaEREQDw8NDQ0PDQsJCAY
AwEBMAIEBgJDA4PFg8PERESFBQHbWYGBgUEIBsZFhUTERAJCAYGBAMCAGQFBggJChAREhUWGB
CAUFBAyHGxcVFBMREQ8KCQgHBgYEBAMCAYT9IA0ODQ0MCwoKCQgHBQUdAgEBAGMFBQcICQoKCww
OQ4NAuANDg0NDAsKCgkIBwUFAwIBAQIDBQUHCAkKCgsMDQ00AAIAAAAAA/gD+AArAG8AAAEfAhU
AwEPay8INT8GMx8DAT8DHwILER8PIT8PES8PIQ8OAvMEAwIBAQME/r8FBQYGBgYFBXkEAWEBAGM
BQUGBgYGBgViAsOFBgYGBgYF/RoBAGMFBQcICQoKCwwNDQ4NAuANDg0NDAsKCgkIBwUFAwIBAQI
BQUHCAkKCgsMDQ0ODf0gDQ4NDQwLcGJCACFBQMCArQFBgYGBgYFBf7FBAMBAQEBAwR2BQUGBgY
3gUEAwEBAGMEYAE1BAMBAQEBA7j9IA0ODQ0MCwoKCQgHBQUdAgEBAGMFBQcICQoKCwwNDQ4NAuA
Og0NDAsKCgkIBwUFAwIBAQIDBQUHCAkKCgsMDQ00AAIAAAAAA/gAIQBDAUAhWCPAmS47QE
AVMAAAAEVDwcvBzU/Bx8GNx8EDw8AS8GPQE/BTsBHwEFHwMPBysBLwU9AT8GOWEfASUfBw8HIy8
PwchHwcPByMvBz8HJR8DDwcrAS8FPQE/BjsBHwEFHwMdaQ8FKwEvBz8GOWEfASUVDwcvBzU/Bx8
JREfDyE/DxEvDyEPDgIgAQIDBAQGBgYGBgYEBAMCAQECAwQEBgYGBgYGBAQDAopiBAMCAQECAwQ
3BYGBgYFBWIEAwICAwQFBQYGBgYF/t8EAWIBAQIDBGI FBQYGBgYFBQQDAgIDBGI FBQYGBgYFAdw
3gUFBAMBAQEBAwQFBQYHigYGBgQEAWIBAQIDBAQGBgb+YAYGBgQEAWIBAQIDBAQGBgaKBwYFBQQ
AQEBAQMEBQUGBwJlBAMCAQECAwRiBQUGBgYGBQUEAwICAwRiBQUGBgYGBf4bYgQDAgIDBAUFBgY
3gUFYgQDAgEBAGMEBQUGBgYGBQEEAQIDBAQGBgYGBgYEBAMCAQECAwQEBgYGBgYGBAQDAv3pAQI
BQUHCAkKCgsMDQ0ODQLgDQ4NDQwLcGJCACFBQMCAQECAwUFBwgJCgoLDA0NDg39IA0ODQ0MCwo
CQgHBQUdAgEwigcGBQUEAwEBAQEDBAUFBgEKBgYGBAQDAgEBAGMEBAYGTWIFBQYGBgYFBQQDAgI
BGI FBQYGBgYFBQQDAgIDBAUFBgYGBgYUFYgQDAgIDBAUFBgYGBgYUFYgQDAgIDmQECAwQEBgYGBgY
BAQDAgEBAGMEBAYGBgYGBgQEAWIBAQIDBAQGBgYGBgYEBAMCAQECAwQEBgYGBgYGBAQDAgHrBQU
3gYGBQViBAMCAGMEBQUGBgYGBQViBAMCAGMEYgUFBgYGBgUFBAMCAGMEYgUFBgYGBgUFBAMCAGN
igYGBgQEAWIBAQIDBAQGBgaKBwYFBQQDAQEBAQMEBQUGD/0gDQ4NDQwLcGJCACFBQMCAQECAwU
BwgJCgoLDA0NDg0C4A0ODQ0MCwoKCQgHBQUdAgEBAGMFBQcICQoKCwwNDQ4AAAAAEgDeAAEAAAA
AAAAAAQAAAAAFAAAAAAAAFQABAAEAAAAAAAIABWAAWAAEAAAAAAMAFQAdAAEAAAAAAAFQAYAAE
AAAAAAAUACwBHAAEAAAAAAAYAFQBSAAEAAAAAAAOALABnAAEAAAAAAASAEgCTAAMAAQJAAAAAgC
AAMAAQQJAAEAKgCnAAMAAQQJAAIADqDRAAMAAQQJAAMAKgDfAAMAAQQJAAQAKgEJAAMAAQQJAAU

```

```

AFgEzAAMAAQQJAAYAKgFJAAMAAQQJAAoAWAFzAAMAAQQJAAsAJAHLIEthbmJhbiBwcmlvcml0eSB
pY29uc1JlZ3VsYXJLYW5iYW4gcHJpb3JpdHkgaWNvbnNLYW5iYW4gcHJpb3JpdHkgaWNvbnNWZXJ
zaW9uIDEuMEthbmJhbiBwcmlvcml0eSBpY29uc0ZvbnQgZ2VuZXJhdGVkIHVzaW5nIFN5bmNmdXN
pb24gTWV0cm8gU3RlZGlvZ3d3LnN5bmNmdXNpb24uY29tACAASwBhAG4AYgBhAG4AIABwAHIAaQB
vAHIAaQB0AHkAIABpAGMABwBuAHMAUgBLAGcAdQBsAGEAcgBLAGEAbgBiAGEAbgAgAHAACgBpAG8
AcgBpAHQAeQAgAGkAYwBvAG4AcwBLAGEAbgBiAGEAbgAgAHAACgBpAG8AcgBpAHQAeQAgAGkAYwB
vAG4AcwBWAGUAcgBzAGkAbwBuACAAMQAuADAASwBhAG4AYgBhAG4AIABwAHIAaQBvAHIAaQB0AHk
AIABpAGMABwBuAHMARGBvAG4AdAAgAGcAZQBwAGUAcgBhAHQAZQBkACAADQBsAGkAbgBnACAAUwB
5AG4AYwBmAHUAcwBpAG8AbgAgAE0AZQB0AHIAbwAgAFMAdABLAGQAaQBvAHcAdwB3AC4AcwB5AG4
AYwBmAHUAcwBpAG8AbgAgAuAGMABwBtAAAAAIAAAAAAAAAACgAAAAAAAAAAAAAAAAAAAAAAAAA
ABQECAQMBBAEFAQYACFRvZG9saXN0BlJldmllZDdlb21wbGV0ZWQIUHJvZ3Jlc3MAAAAA)
format('true-type');
    font-weight: normal;
    font-style: normal;
}

[class^="sf-icon-"],
[class*=" sf-icon-"] {
    font-family: 'Kanban priority icons' !important;
    speak: none;
    font-size: 55px;
    font-style: normal;
    font-weight: normal;
    font-variant: normal;
    text-transform: none;
    line-height: 1;
    -webkit-font-smoothing: antialiased;
    -moz-osx-font-smoothing: grayscale;
}
</style>

```

{% previewsample "page.domainurl/code-snippet/kanban/header-template-cs1" %}

### Toggle columns

Kanban allows to expand or collapse its columns using `allowToggle` inside the `columns` property. When enable the property, it will render the expand or collapse icon to the column header.

By default, collapsed column width is set to `50px`.

### APP.VUE

```

<template>
  <div id="app">
    <ejs-kanban id="kanban" keyField="Status" :dataSource="kanbanData"
      :cardSettings="cardSettings">
      <e-columns>
        <e-column headerText="To Do" keyField="Open"
allowToggle="true"></e-column>
        <e-column headerText="In Progress" keyField="InProgress"
allowToggle="true"></e-column>
        <e-column headerText="Testing" keyField="Testing"
allowToggle="true"></e-column>
        <e-column headerText="Done" keyField="Close"
allowToggle="true"></e-column>
      </e-columns>
    </div>
  </template>

```

```

        </ejs-kanban>
    </div>
</template>
<script>
import Vue from "vue";
import { KanbanPlugin } from '@syncfusion/ej2-vue-kanban';
import { extend } from '@syncfusion/ej2-base';
import { kanbanData } from './datasource.js';
Vue.use(KanbanPlugin);
export default {
    data: function() {
        return {
            kanbanData: extend([], kanbanData, null, true),
            cardSettings: {
                contentField: "Summary",
                headerField: "Id"
            }
        };
    },
};
</script>
<style>
@import '../node_modules/@syncfusion/ej2-base/styles/material.css';
@import '../node_modules/@syncfusion/ej2-buttons/styles/material.css';
@import '../node_modules/@syncfusion/ej2-layouts/styles/material.css';
@import '../node_modules/@syncfusion/ej2-dropdowns/styles/material.css';
@import '../node_modules/@syncfusion/ej2-inputs/styles/material.css';
@import '../node_modules/@syncfusion/ej2-navigations/styles/material.css';
@import '../node_modules/@syncfusion/ej2-popups/styles/material.css';
@import '../node_modules/@syncfusion/ej2-vue-kanban/styles/material.css';
</style>

```

{% previewsample "page.domainurl/code-snippet/kanban/toggle-cs1" %}

#### Initially collapsed column

By default, all columns are on expanded state when loading the Kanban board initially. But, you can render the columns with collapsed state using the `isExpanded` property.

The `isExpanded` property only works when enabling the `allowToggle` property on particular column.

In the following example, the backlog column is collapsed on initialization of Kanban board.

#### APP.VUE

```

<template>
    <div id="app">
        <ejs-kanban id="kanban" keyField="Status" :dataSource="kanbanData"
            :cardSettings="cardSettings">
            <e-columns>
                <e-column headerText="To Do" keyField="Open" allowToggle="true"
isExpanded="false"></e-column>
                <e-column headerText="In Progress" keyField="InProgress"
allowToggle="true"></e-column>
                <e-column headerText="Testing" keyField="Testing"
allowToggle="true" isExpanded="false"></e-column>
                <e-column headerText="Done" keyField="Close"
allowToggle="true"></e-column>
            </e-columns>
        </ejs-kanban>
    </div>
</template>

```

```

        </e-columns>
      </ejs-kanban>
    </div>
  </template>
  <script>
import Vue from "vue";
import { KanbanPlugin } from '@syncfusion/ej2-vue-kanban';
import { extend } from '@syncfusion/ej2-base';
import { kanbanData } from './datasource.js';
Vue.use(KanbanPlugin);
export default {
  data: function() {
    return {
      kanbanData: extend([], kanbanData, null, true),
      cardSettings: {
        contentField: "Summary",
        headerField: "Id"
      }
    };
  },
};
  </script>
  <style>
@import '../node_modules/@syncfusion/ej2-base/styles/material.css';
@import '../node_modules/@syncfusion/ej2-buttons/styles/material.css';
@import '../node_modules/@syncfusion/ej2-layouts/styles/material.css';
@import '../node_modules/@syncfusion/ej2-dropdowns/styles/material.css';
@import '../node_modules/@syncfusion/ej2-inputs/styles/material.css';
@import '../node_modules/@syncfusion/ej2-navigations/styles/material.css';
@import '../node_modules/@syncfusion/ej2-popups/styles/material.css';
@import '../node_modules/@syncfusion/ej2-vue-kanban/styles/material.css';
  </style>

```

{% previewsample "page.domainurl/code-snippet/kanban/expanded-cs1" %}

### Stacked headers

Stacked headers are the additional headers to column header that will group the similar columns.

Define the grouping of columns **key** value to the **keyFields** property and provide the custom header text name to grouped columns using the **text** property, which is placed inside the **stackedHeaders** property.

In the following code, the kanban columns 'InProgress, Review' are grouped under 'Development Phase' category.

### APP.VUE

```

<template>
  <div id="app">
    <ejs-kanban id="kanban" keyField="Status" :dataSource="kanbanData"
      :cardSettings="cardSettings">
      <e-columns>
        <e-column headerText="Open" keyField="Open"></e-column>
        <e-column headerText="In Progress" keyField="InProgress"></e-
column>
        <e-column headerText="In Review" keyField="Review"></e-column>

```

```

        <e-column headerText="Completed" keyField="Close"></e-column>
    </e-columns>
    <e-stackedHeaders>
        <e-stackedHeader text= 'To Do' keyFields= 'Open'></e-
stackedHeader>
        <e-stackedHeader text= 'Development Phase' keyFields=
'InProgress, Review'></e-stackedHeader>
        <e-stackedHeader text= 'Done' keyFields= 'Close'></e-
stackedHeader>
    </e-stackedHeaders>
</ejs-kanban>
</div>
</template>
<script>
import Vue from "vue";
import { KanbanPlugin } from '@syncfusion/ej2-vue-kanban';
import { extend } from '@syncfusion/ej2-base';
import { kanbanData } from './datasource.js';
Vue.use(KanbanPlugin);
export default {
    data: function() {
        return {
            kanbanData: extend([], kanbanData, null, true),
            cardSettings: {
                contentField: "Summary",
                headerField: "Id"
            }
        };
    },
};
</script>
<style>
@import '../node_modules/@syncfusion/ej2-base/styles/material.css';
@import '../node_modules/@syncfusion/ej2-buttons/styles/material.css';
@import '../node_modules/@syncfusion/ej2-layouts/styles/material.css';
@import '../node_modules/@syncfusion/ej2-dropdowns/styles/material.css';
@import '../node_modules/@syncfusion/ej2-inputs/styles/material.css';
@import '../node_modules/@syncfusion/ej2-navigations/styles/material.css';
@import '../node_modules/@syncfusion/ej2-popups/styles/material.css';
@import '../node_modules/@syncfusion/ej2-vue-kanban/styles/material.css';
</style>

```

{% previewsample "page.domainurl/code-snippet/kanban/stacked-headers-cs1" %}

### Cards in Vue Kanban component

The cards are main elements in Kanban board, which represent the task information with header and content. The header and content of a card is fetched from the corresponding mapping fields. The card layout can be customized with template also.

### Drag-and-drop

Transit or change the card position using the drag-and-drop functionality. By default, the `allowDragAndDrop` property is enabled on the Kanban board, which is used to change the card position by column-to-column or within the column.

Added dotted border on Kanban cells except the dragged clone cells when dragging, which indicates the possible ways for dropping the cards into the cells.

### Header

The card header is achieved by mapping the `headerField` property, which is placed inside the `cardSettings` property. By default, the `showHeader` property enabled by Kanban board that is used to show the header at the top of the card.

The `headerField` property of `cardSettings` is mandatory to render the cards in the Kanban board. It acts as a unique field that is used to avoid the duplication of card data. You can not change the `headerField` of mapped data value using the `updateCard` public method or server-side update of data.

In the following demo, the `showHeader` property is disabled on Kanban board.

### APP.VUE

```
<template>
  <div id="app">
    <ejs-kanban id="kanban" keyField="Status" :dataSource="kanbanData"
      :cardSettings="cardSettings">
      <e-columns>
        <e-column headerText="To Do" keyField="Open"></e-column>
        <e-column headerText="In Progress" keyField="InProgress"></e-
column>
        <e-column headerText="Testing" keyField="Testing"></e-column>
        <e-column headerText="Done" keyField="Close"></e-column>
      </e-columns>
    </ejs-kanban>
  </div>
</template>
<script>
import Vue from "vue";
import { KanbanPlugin } from '@syncfusion/ej2-vue-kanban';
import { extend } from '@syncfusion/ej2-base';
Vue.use(KanbanPlugin);
let kanbanData = [
  {
    'Id': 1,
    'Status': 'Open',
    'Summary': 'Analyze the new requirements gathered from the customer.',
    'Type': 'Story',
    'Priority': 'Low',
    'Tags': 'Analyze, Customer',
    'Estimate': 3.5,
    'Assignee': 'Nancy Davloio',
    'RankId': 1
  },
  {
    'Id': 2,
    'Status': 'InProgress',
    'Summary': 'Improve application performance',
    'Type': 'Improvement',
    'Priority': 'Normal',
    'Tags': 'Improvement',
    'Estimate': 6,
    'Assignee': 'Andrew Fuller',
```



```

      'RankId': 1
    },
    {
      'Id': 3,
      'Status': 'Testing',
      'Summary': 'Arrange a web meeting with the customer to get new
requirements.',
      'Type': 'Others',
      'Priority': 'Critical',
      'Tags': 'Meeting',
      'Estimate': 5.5,
      'Assignee': 'Janet Leverling',
      'RankId': 2
    },
    {
      'Id': 4,
      'Status': 'Close',
      'Summary': 'Fix the issues reported in the IE browser.',
      'Type': 'Bug',
      'Priority': 'Release Breaker',
      'Tags': 'IE',
      'Estimate': 2.5,
      'Assignee': 'Janet Leverling',
      'RankId': 2
    },
    {
      'Id': 5,
      'Status': 'Open',
      'Summary': 'Fix the issues reported by the customer.',
      'Type': 'Bug',
      'Priority': 'Low',
      'Tags': 'Customer',
      'Estimate': 3.5,
      'Assignee': 'Steven walker',
      'RankId': 1
    },
    {
      'Id': 6,
      'Status': 'InProgress',
      'Summary': 'Arrange a web meeting with the customer to get the login
page requirements.',
      'Type': 'Others',
      'Priority': 'Low',
      'Tags': 'Meeting',
      'Estimate': 2,
      'Assignee': 'Michael Suyama',
      'RankId': 1
    },
    {
      'Id': 7,
      'Status': 'Testing',
      'Summary': 'Validate new requirements',
      'Type': 'Improvement',
      'Priority': 'Low',
      'Tags': 'Validation',
      'Estimate': 1.5,
      'Assignee': 'Robert King',

```

```

      'RankId': 1
    },
    {
      'Id': 8,
      'Status': 'Close',
      'Summary': 'Login page validation',
      'Type': 'Story',
      'Priority': 'Release Breaker',
      'Tags': 'Validation,Fix',
      'Estimate': 2.5,
      'Assignee': 'Laura Callahan',
      'RankId': 2
    }
  ];
  export default {
    data: function() {
      return {
        kanbanData: extend([], kanbanData, null, true),
        cardSettings: {
          contentField: "Summary",
          headerField: "Id",
          showHeader: false
        }
      };
    },
  },
}
</script>
<style>
@import '../node_modules/@syncfusion/ej2-base/styles/material.css';
@import '../node_modules/@syncfusion/ej2-buttons/styles/material.css';
@import '../node_modules/@syncfusion/ej2-layouts/styles/material.css';
@import '../node_modules/@syncfusion/ej2-dropdowns/styles/material.css';
@import '../node_modules/@syncfusion/ej2-inputs/styles/material.css';
@import '../node_modules/@syncfusion/ej2-navigations/styles/material.css';
@import '../node_modules/@syncfusion/ej2-popups/styles/material.css';
@import '../node_modules/@syncfusion/ej2-vue-kanban/styles/material.css';
</style>

```

{% previewsample "page.domainurl/code-snippet/kanban/card-header-cs1" %}

### Content

The card's content is fetched from data source using the `contentField` property, which is placed inside the `cardSettings` property. If the `contentField` property is not used, card is rendered with empty content.

### Template

You can customize the default card layout using template as per your application needs. This can be achieved by template of the `cardSettings` property.

### APP.VUE

```

<template>
  <div id="app">
    <ejs-kanban id="kanban" keyField="Status" :dataSource="kanbanData"
      :cardSettings="cardSettings">

```

```

        <e-columns>
            <e-column headerText="To Do" keyField="Open"></e-column>
            <e-column headerText="In Progress" keyField="InProgress"></e-
column>
            <e-column headerText="Testing" keyField="Testing"></e-column>
            <e-column headerText="Done" keyField="Close"></e-column>
        </e-columns>
    </ejs-kanban>
</div>
</template>
<script>
import Vue from "vue";
import { KanbanPlugin } from '@syncfusion/ej2-vue-kanban';
import { extend } from '@syncfusion/ej2-base';
import { kanbanData } from './datasource.js';
Vue.use(KanbanPlugin);
export default {
    data: function() {
        return {
            kanbanData: extend([], kanbanData, null, true),
            cardSettings: {
                headerField: "Id",
                template: function() {
                    return {
                        template: Vue.component('cardTemplate', {
                            template: `<div class='e-card-content'>
                                <table class="card-template-wrap">
                                    <tbody>
  <tr>
  <td class="CardHeader">Id:</td>
  <td>{{data.Id}}</td>
  </tr>
  <tr>
  <td class="CardHeader">Type:</td>
  <td>{{data.Type}}</td>
  </tr>
  <tr>
  <td class="CardHeader">Priority:</td>
  <td>{{data.Priority}}</td>
  </tr>
  <tr>
  <td class="CardHeader">Summary:</td>
  <td>{{data.Summary}}</td>
  </tr>
                                    </tbody>
                                </table>
                            </div>`,
                            data() { }
                        })
                    }
                }
            }
        };
    },
};
</script>
<style>

```

```
@import '../node_modules/@syncfusion/ej2-base/styles/material.css';
@import '../node_modules/@syncfusion/ej2-buttons/styles/material.css';
@import '../node_modules/@syncfusion/ej2-layouts/styles/material.css';
@import '../node_modules/@syncfusion/ej2-dropdowns/styles/material.css';
@import '../node_modules/@syncfusion/ej2-inputs/styles/material.css';
@import '../node_modules/@syncfusion/ej2-navigations/styles/material.css';
@import '../node_modules/@syncfusion/ej2-popups/styles/material.css';
@import '../node_modules/@syncfusion/ej2-vue-kanban/styles/material.css';
.e-kanban .card-template-wrap td {
    background: none !important;
}
.e-kanban .card-template-wrap .CardHeader {
    font-weight: 500;
}
</style>
```

{% previewsample "page.domainurl/code-snippet/kanban/card-template-cs1" %}

### Selection

Kanban board allows to select single and multiple selection of cards when mouse or keyboard interactions using `selectionType` property. The property contains following types.

- **None:** No cards are allowed to select from Kanban board.
- **Single:** Only one card allowed to select at a time in the Kanban board.
- **Multiple:** Multiple cards are allowed to select in a board.

### Multiple Selection

Select the multiple cards randomly using Ctrl + mouse click and select the multiple cards continuously using Shift + mouse click action on Kanban board. Set **Multiple** in `selectionType` to enable the multiple selection in a board.

### APP.VUE

```
<template>
  <div id="app">
    <ejs-kanban id="kanban" keyField="Status" :dataSource="kanbanData"
      :cardSettings="cardSettings">
      <e-columns>
        <e-column headerText="To Do" keyField="Open"></e-column>
        <e-column headerText="In Progress" keyField="InProgress"></e-
column>
        <e-column headerText="Testing" keyField="Testing"></e-column>
        <e-column headerText="Done" keyField="Close"></e-column>
      </e-columns>
    </ejs-kanban>
  </div>
</template>
<script>
import Vue from "vue";
import { KanbanPlugin } from '@syncfusion/ej2-vue-kanban';
import { extend } from '@syncfusion/ej2-base';
import { kanbanData } from './datasource.js';
Vue.use(KanbanPlugin);
export default {
```

```

data: function() {
  return {
    kanbanData: extend([], kanbanData, null, true),
    cardSettings: {
      contentField: "Summary",
      headerField: "Id",
      selectionType: 'Multiple'
    }
  };
},
}
</script>
<style>
@import '../node_modules/@syncfusion/ej2-base/styles/material.css';
@import '../node_modules/@syncfusion/ej2-buttons/styles/material.css';
@import '../node_modules/@syncfusion/ej2-layouts/styles/material.css';
@import '../node_modules/@syncfusion/ej2-dropdowns/styles/material.css';
@import '../node_modules/@syncfusion/ej2-inputs/styles/material.css';
@import '../node_modules/@syncfusion/ej2-navigations/styles/material.css';
@import '../node_modules/@syncfusion/ej2-popups/styles/material.css';
@import '../node_modules/@syncfusion/ej2-vue-kanban/styles/material.css';
</style>

```

{% previewsample "page.domainurl/code-snippet/kanban/multiple-selection-cs1" %}

### Data binding in Vue Kanban component

The Kanban uses **DataManager**, which supports both RESTful data service binding and JavaScript object array binding. The [dataSource](#) property of Kanban can be assigned either with the instance of **DataManager** or JavaScript object array collection, as it supports the following two data binding methods:

- Local data
- Remote data

#### Local data

To bind local JSON data to the Kanban, you can simply assign a JavaScript object array to the [dataSource](#) property. The JSON object dataSource can also be provided as an instance of **DataManager** and assigned to the Kanban **dataSource** property.

#### APP.VUE

```

<template>
  <div id="app">
    <ejs-kanban id="kanban" keyField="Status" :dataSource="kanbanData"
      :cardSettings="cardSettings">
      <e-columns>
        <e-column headerText="To Do" keyField="Open"></e-column>
        <e-column headerText="In Progress" keyField="InProgress"></e-
column>
        <e-column headerText="Testing" keyField="Testing"></e-column>
        <e-column headerText="Done" keyField="Close"></e-column>
      </e-columns>
    </ejs-kanban>
  </div>

```

```

</template>
<script>
import Vue from "vue";
import { KanbanPlugin } from '@syncfusion/ej2-vue-kanban';
import { extend } from '@syncfusion/ej2-base';
import { kanbanData } from './datasource.js';
Vue.use(KanbanPlugin);
export default {
  data: function() {
    return {
      kanbanData: extend([], kanbanData, null, true),
      cardSettings: {
        contentField: "Summary",
        headerField: "Id",
      }
    };
  },
};
</script>
<style>
@import '../node_modules/@syncfusion/ej2-base/styles/material.css';
@import '../node_modules/@syncfusion/ej2-buttons/styles/material.css';
@import '../node_modules/@syncfusion/ej2-layouts/styles/material.css';
@import '../node_modules/@syncfusion/ej2-dropdowns/styles/material.css';
@import '../node_modules/@syncfusion/ej2-inputs/styles/material.css';
@import '../node_modules/@syncfusion/ej2-navigations/styles/material.css';
@import '../node_modules/@syncfusion/ej2-popups/styles/material.css';
@import '../node_modules/@syncfusion/ej2-vue-kanban/styles/material.css';
</style>

```

{% previewsample "page.domainurl/code-snippet/kanban/local-data-cs1" %}

By default, **DataManager** uses **JsonAdaptor** for binding local data.

#### Remote data

To bind remote data to kanban component, assign service data as an instance of [DataManager](#) to the [dataSource](#) property. To interact with remote data source, provide the endpoint [url](#).

#### APP.VUE

```

<template>
  <div id="app">
    <ejs-kanban id="kanban" keyField="Status" :dataSource="kanbanData"
      :cardSettings="cardSettings" allowDragAndDrop="false"
      :dialogOpen="dialogOpen">
      <e-columns>
        <e-column headerText="To Do" keyField="Open"></e-column>
        <e-column headerText="In Progress" keyField="InProgress"></e-
column>
        <e-column headerText="Testing" keyField="Testing"></e-column>
        <e-column headerText="Done" keyField="Close"></e-column>
      </e-columns>
    </ejs-kanban>
  </div>
</template>
<script>

```

```

import Vue from "vue";
import { KanbanPlugin } from '@syncfusion/ej2-vue-kanban';
import { DataManager, ODataAdaptor } from "@syncfusion/ej2-data";
import { extend } from '@syncfusion/ej2-base';
Vue.use(KanbanPlugin);
export default {
  data: function() {
    let SERVICE_URI = "https://ej2services.syncfusion.com/production/web-services/api/Kanban";
    return {
      kanbanData: new DataManager({
        url: SERVICE_URI,
        adaptor: new ODataAdaptor(),
        crossDomain: true
      });
      cardSettings: {
        contentField: "Summary",
        headerField: "Id",
      }
    };
  },
  methods: {
    dialogOpen: function (args) {
      args.cancel = true;
    }
  }
}
</script>
<style>
@import '../node_modules/@syncfusion/ej2-base/styles/material.css';
@import '../node_modules/@syncfusion/ej2-buttons/styles/material.css';
@import '../node_modules/@syncfusion/ej2-layouts/styles/material.css';
@import '../node_modules/@syncfusion/ej2-dropdowns/styles/material.css';
@import '../node_modules/@syncfusion/ej2-inputs/styles/material.css';
@import '../node_modules/@syncfusion/ej2-navigations/styles/material.css';
@import '../node_modules/@syncfusion/ej2-popups/styles/material.css';
@import '../node_modules/@syncfusion/ej2-vue-kanban/styles/material.css';
</style>

```

{% previewsample "page.domainurl/code-snippet/kanban/remote-data-cs1" %}

By default, [DataManager](#) uses **ODataAdaptor** for remote data-binding.

#### *OData services*

[OData](#) is a standardized protocol for creating and consuming data. You can retrieve data from OData service using the **DataManager**. Refer to the following code example for remote Data binding using OData service.

#### **APP.VUE**

```

<template>
  <div id="app">
    <ejs-kanban id="kanban" keyField="Status" :dataSource="kanbanData"
      :cardSettings="cardSettings" allowDragAndDrop="false"
      :dialogOpen="dialogOpen">
      <e-columns>

```

```

        <e-column headerText="To Do" keyField="Open"></e-column>
        <e-column headerText="In Progress" keyField="InProgress"></e-
column>
        <e-column headerText="Testing" keyField="Testing"></e-column>
        <e-column headerText="Done" keyField="Close"></e-column>
    </e-columns>
</ejs-kanban>
</div>
</template>
<script>
import Vue from "vue";
import { KanbanPlugin } from '@syncfusion/ej2-vue-kanban';
import { DataManager, ODataAdaptor } from "@syncfusion/ej2-data";
import { extend } from '@syncfusion/ej2-base';
Vue.use(KanbanPlugin);
export default {
    data: function() {
        let SERVICE_URI = "https://ej2services.syncfusion.com/production/web-
services/api/Kanban";
        return {
            kanbanData: new DataManager({
                url: SERVICE_URI,
                adaptor: new ODataAdaptor(),
                crossDomain: true
            });
            cardSettings: {
                contentField: "Summary",
                headerField: "Id",
            }
        };
    },
    methods: {
        dialogOpen: function (args) {
            args.cancel = true;
        }
    }
}
</script>
<style>
@import '../node_modules/@syncfusion/ej2-base/styles/material.css';
@import '../node_modules/@syncfusion/ej2-buttons/styles/material.css';
@import '../node_modules/@syncfusion/ej2-layouts/styles/material.css';
@import '../node_modules/@syncfusion/ej2-dropdowns/styles/material.css';
@import '../node_modules/@syncfusion/ej2-inputs/styles/material.css';
@import '../node_modules/@syncfusion/ej2-navigations/styles/material.css';
@import '../node_modules/@syncfusion/ej2-popups/styles/material.css';
@import '../node_modules/@syncfusion/ej2-vue-kanban/styles/material.css';
</style>

```

{% previewsample "page.domainurl/code-snippet/kanban/odata-cs1" %}

#### [OData v4 services](#)

The ODataV4 is an improved version of OData protocols, and the [DataManager](#) can also retrieve and consume OData v4 services. For more details on OData v4 services, refer to the [odata documentation](#). To bind OData v4 service, use the **ODataV4Adaptor**.



**APP.VUE**

```

<template>
  <div id="app">
    <ejs-kanban id="kanban" keyField="ContactTitle"
:dataSource="kanbanData"
    :cardSettings="cardSettings" allowDragAndDrop="false"
:dialogOpen="dialogOpen">
      <e-columns>
        <e-column headerText="Order Administrator" keyField="Order
Administrator"></e-column>
        <e-column headerText="Sales Representative" keyField="Sales
Representative"></e-column>
        <e-column headerText="Export Administrator" keyField="Export
Administrator"></e-column>
      </e-columns>
    </ejs-kanban>
  </div>
</template>
<script>
import Vue from "vue";
import { KanbanPlugin } from '@syncfusion/ej2-vue-kanban';
import { DataManager, ODataV4Adaptor } from "@syncfusion/ej2-data";
import { extend } from '@syncfusion/ej2-base';
Vue.use(KanbanPlugin);
export default {
  data: function() {
    let SERVICE_URI =
"https://services.odata.org/v4/northwind/northwind.svc/Suppliers";
    return {
      kanbanData: new DataManager({
        url: SERVICE_URI,
        adaptor: new ODataV4Adaptor(),
        crossDomain: true
      });
      cardSettings: {
        contentField: "ContactName",
        headerField: "SupplierID",
      }
    };
  },
  methods: {
    dialogOpen: function (args) {
      args.cancel = true;
    }
  }
}
</script>
<style>
@import '../node_modules/@syncfusion/ej2-base/styles/material.css';
@import '../node_modules/@syncfusion/ej2-buttons/styles/material.css';
@import '../node_modules/@syncfusion/ej2-layouts/styles/material.css';
@import '../node_modules/@syncfusion/ej2-dropdowns/styles/material.css';
@import '../node_modules/@syncfusion/ej2-inputs/styles/material.css';
@import '../node_modules/@syncfusion/ej2-navigations/styles/material.css';
@import '../node_modules/@syncfusion/ej2-popups/styles/material.css';
@import '../node_modules/@syncfusion/ej2-vue-kanban/styles/material.css';

```

```
</style>
```

```
{% previewsample "page.domainurl/code-snippet/kanban/odataV4-cs1" %}
```

#### [Web API](#)

You can use **WebApiAdaptor** to bind kanban with Web API created using OData endpoint.

```
`ts
```

```
<template>
```

```
<div id="app">
```

```
<ejs-kanban id="kanban" keyField="Status" :dataSource="kanbanData"
```

```
:cardSettings="cardSettings" allowDragAndDrop="false" :dialogOpen="dialogOpen">
```

```
<e-columns>
```

```
<e-column headerText="To Do" keyField="Open"></e-column>
```

```
<e-column headerText="In Progress" keyField="InProgress"></e-column>
```

```
<e-column headerText="Testing" keyField="Testing"></e-column>
```

```
<e-column headerText="Done" keyField="Close"></e-column>
```

```
</e-columns>
```

```
</ejs-kanban>
```

```
</div>
```

```
</template>
```

```
<script>
```

```
import Vue from "vue";
```

```
import { KanbanPlugin } from '@syncfusion/ej2-vue-kanban';
```

```
import { DataManager, WebApiAdaptor } from "@syncfusion/ej2-data";
```

```
Vue.use(KanbanPlugin);
```

```
export default {
```

```
  data: function() {
```

```
    return {
```

```
      kanbanData: new DataManager({
```

```
        url: '/api/Tasks',
```

```
        adaptor: new WebApiAdaptor(),
```

```
        crossDomain: true
```

```
      });
```

```
      cardSettings: {
```

```
        contentField: "Summary",
```

```
headerField: "Id",
}
};
},
methods: {
  dialogOpen: function (args) {
    args.cancel = true;
  }
}
}
</script>
<style>
@import '../node_modules/@syncfusion/ej2-base/styles/material.css';
@import '../node_modules/@syncfusion/ej2-buttons/styles/material.css';
@import '../node_modules/@syncfusion/ej2-layouts/styles/material.css';
@import '../node_modules/@syncfusion/ej2-dropdowns/styles/material.css';
@import '../node_modules/@syncfusion/ej2-inputs/styles/material.css';
@import '../node_modules/@syncfusion/ej2-navigations/styles/material.css';
@import '../node_modules/@syncfusion/ej2-popups/styles/material.css';
@import '../node_modules/@syncfusion/ej2-vue-kanban/styles/material.css';
</style>
`
```

Below server-side controller code to get the Kanban data.

```
`ts
[HttpGet]
public object Get()
{
  var data = OrdersDetails.GetAllRecords().ToList();
  return data;
}
`
```

*URL adaptor*

The CRUD (Create, Read, Update and Delete) actions can be performed easily on Kanban cards using the various adaptors available within the `DataManager`. Most preferably, we will be using `UrlAdaptor` for performing CRUD actions on Kanban.

The CRUD operation in Kanban can be mapped to server-side controller actions using the properties `insertUrl`, `removeUrl`, `updateUrl`, and `crudUrl`.

- `insertUrl` – You can perform a single insertion operation on the server-side.
- `updateUrl` – You can update single data on the server-side.
- `removeUrl` – You can remove single data on the server-side.
- `crudUrl` – You can perform bulk data operation on the server-side.

```
`ts
```

```
<template>
<div id="app">
  <ejs-kanban id="kanban" keyField="Status" :dataSource="kanbanData"
    :cardSettings="cardSettings">
    <e-columns>
      <e-column headerText="To Do" keyField="Open"></e-column>
      <e-column headerText="In Progress" keyField="InProgress"></e-column>
      <e-column headerText="Testing" keyField="Testing"></e-column>
      <e-column headerText="Done" keyField="Close"></e-column>
    </e-columns>
  </ejs-kanban>
</div>
</template>
<script>
import Vue from "vue";
import { KanbanPlugin } from '@syncfusion/ej2-vue-kanban';
import { DataManager, UrlAdaptor } from "@syncfusion/ej2-data";
Vue.use(KanbanPlugin);
export default {
  data: function() {
    return {
      kanbanData: new DataManager({
        url: 'Home/DataSource',
```

```

updateUrl: 'Home/Update',
insertUrl: 'Home/Insert',
removeUrl: 'Home/Delete',
adaptor: new UrlAdaptor(),
crossDomain: true
});
cardSettings: {
contentField: "Summary",
headerField: "Id",
}
};
}
}
</script>
<style>
@import '../node_modules/@syncfusion/ej2-base/styles/material.css';
@import '../node_modules/@syncfusion/ej2-buttons/styles/material.css';
@import '../node_modules/@syncfusion/ej2-layouts/styles/material.css';
@import '../node_modules/@syncfusion/ej2-dropdowns/styles/material.css';
@import '../node_modules/@syncfusion/ej2-inputs/styles/material.css';
@import '../node_modules/@syncfusion/ej2-navigations/styles/material.css';
@import '../node_modules/@syncfusion/ej2-popups/styles/material.css';
@import '../node_modules/@syncfusion/ej2-vue-kanban/styles/material.css';
</style>
`

```

The server-side controller code to handle the CRUD operations are as follows.

```

`ts
private NORTHWNDEntities db = new NORTHWNDEntities();
public ActionResult DataSource() {
var DataSource = db.Tasks.ToList();
return Json(DataSource, JsonRequestBehavior.AllowGet);
}
public ActionResult Insert(Params value) {

```

```
//Insert card data into the database
return Json(value, JsonRequestBehavior.AllowGet);
}

public ActionResult Update(Params value) {
//Update card data into the database
return Json(value, JsonRequestBehavior.AllowGet);
}

public void Delete(Params value) {
//Delete card data from the database
}

public class Params {
public int Id { get; set; }
public string Status { get; set; }
public string Summary { get; set; }
public string Assignee { get; set; }
}
`
```

The `crudUrl` is used to update the bulk data sent to the server-side. Multiple selections and `sortBy` as `Index` properties are used for `crudUrl` properties to update the modified bulk data to the server-side.

#### Custom adaptor

It is possible to create your own custom adaptor by extending the built-in available adaptors. The following example demonstrates the custom adaptor usage and how to add a custom field `TaskId` for the cards by overriding the built-in response processing using the `processResponse` method of the `ODataAdaptor`.

#### APP.VUE

```
<template>
  <div id="app">
    <ejs-kanban id="kanban" keyField="Status" :dataSource="kanbanData"
      :cardSettings="cardSettings" allowDragAndDrop="false"
      :dialogOpen="dialogOpen">
      <e-columns>
        <e-column headerText="To Do" keyField="Open"></e-column>
        <e-column headerText="In Progress" keyField="InProgress"></e-
column>
        <e-column headerText="Testing" keyField="Testing"></e-column>
        <e-column headerText="Done" keyField="Close"></e-column>
      </e-columns>
    </ejs-kanban>
  </div>
</template>
<script>
```

```

import Vue from "vue";
import { KanbanPlugin } from '@syncfusion/ej2-vue-kanban';
import { DataManager, ODataAdaptor } from "@syncfusion/ej2-data";
import { extend } from '@syncfusion/ej2-base';
Vue.use(KanbanPlugin);
class TaskIdAdaptor extends ODataAdaptor {
  processResponse() {
    let i = 0;
    // calling base class processResponse function
    let original = super.processResponse.apply(this, arguments);
    // adding Task Id
    original.forEach((item) => item['Id'] = 'Task - ' + ++i);
    return original;
  }
}
export default {
  data: function() {
    let SERVICE_URI = "https://ej2services.syncfusion.com/production/web-services/api/Kanban";
    return {
      kanbanData: new DataManager({
        url: SERVICE_URI,
        adaptor: new TaskIdAdaptor
      });
      cardSettings: {
        contentField: "Summary",
        headerField: "Id",
      }
    };
  },
  methods: {
    dialogOpen: function (args) {
      args.cancel = true;
    }
  }
}
</script>
<style>
@import '../node_modules/@syncfusion/ej2-base/styles/material.css';
@import '../node_modules/@syncfusion/ej2-buttons/styles/material.css';
@import '../node_modules/@syncfusion/ej2-layouts/styles/material.css';
@import '../node_modules/@syncfusion/ej2-dropdowns/styles/material.css';
@import '../node_modules/@syncfusion/ej2-inputs/styles/material.css';
@import '../node_modules/@syncfusion/ej2-navigations/styles/material.css';
@import '../node_modules/@syncfusion/ej2-popups/styles/material.css';
@import '../node_modules/@syncfusion/ej2-vue-kanban/styles/material.css';
</style>

```

{% previewsample "page.domainurl/code-snippet/kanban/custom-cs1" %}

#### [Sending additional parameters to the server](#)

To add a custom parameter to the data request, use the **addParams** method of **Query** class. Assign the **Query** object with additional parameters to the kanban [query](#) property.

#### **APP.VUE**

```

<template>
  <div id="app">
    <ejs-kanban id="kanban" keyField="Status" :dataSource="kanbanData"
      :cardSettings="cardSettings" allowDragAndDrop="false"
      :dialogOpen="dialogOpen" :query='query'>
      <e-columns>
        <e-column headerText="To Do" keyField="Open"></e-column>
        <e-column headerText="In Progress" keyField="InProgress"></e-
column>
        <e-column headerText="Testing" keyField="Testing"></e-column>
        <e-column headerText="Done" keyField="Close"></e-column>
      </e-columns>
    </ejs-kanban>
  </div>
</template>
<script>
import Vue from "vue";
import { KanbanPlugin } from '@syncfusion/ej2-vue-kanban';
import { DataManager, ODataAdaptor, Query } from "@syncfusion/ej2-data";
Vue.use(KanbanPlugin);
export default {
  data: function() {
    let SERVICE_URI = "https://ej2services.syncfusion.com/production/web-
services/api/Kanban";
    return {
      kanbanData: new DataManager({
        url: SERVICE_URI,
        adaptor: new ODataAdaptor(),
        crossDomain: true
      });
      cardSettings: {
        contentField: "Summary",
        headerField: "Id"
      },
      query: new Query().addParams('ej2kanban', 'true');
    };
  },
  methods: {
    dialogOpen: function (args) {
      args.cancel = true;
    }
  }
}
</script>
<style>
@import '../node_modules/@syncfusion/ej2-base/styles/material.css';
@import '../node_modules/@syncfusion/ej2-buttons/styles/material.css';
@import '../node_modules/@syncfusion/ej2-layouts/styles/material.css';
@import '../node_modules/@syncfusion/ej2-dropdowns/styles/material.css';
@import '../node_modules/@syncfusion/ej2-inputs/styles/material.css';
@import '../node_modules/@syncfusion/ej2-navigations/styles/material.css';
@import '../node_modules/@syncfusion/ej2-popups/styles/material.css';
@import '../node_modules/@syncfusion/ej2-vue-kanban/styles/material.css';
</style>

```

{% previewsample "page.domainurl/code-snippet/kanban/additional-cs1" %}



The parameters added using the [query](#) property will be sent along with the data request for every kanban action.

#### Handling HTTP error

During server interaction from the kanban, some server-side exceptions may occur, and you can acquire those error messages or exception details

in client-side using the [actionFailure](#) event.

The argument passed to the [actionFailure](#) event contains the error details returned from the server.

#### APP.VUE

```
<template>
  <div id="app">
    <ejs-kanban id="kanban" ref="KanbanObj" keyField="Status"
      :dataSource="kanbanData"
      :cardSettings="cardSettings" :actionFailure='actionFailure'>
      <e-columns>
        <e-column headerText="To Do" keyField="Open"></e-column>
        <e-column headerText="In Progress" keyField="InProgress"></e-column>
        <e-column headerText="Testing" keyField="Testing"></e-column>
        <e-column headerText="Done" keyField="Close"></e-column>
      </e-columns>
    </ejs-kanban>
  </div>
</template>
<script>
import Vue from "vue";
import { KanbanPlugin } from '@syncfusion/ej2-vue-kanban';
import { DataManager, ODataAdaptor } from "@syncfusion/ej2-data";
Vue.use(KanbanPlugin);
export default {
  data: function() {
    return {
      kanbanData: new DataManager({
        url: 'http://some.com/invalidUrl',
        adaptor: new ODataAdaptor()
      }),
      cardSettings: {
        contentField: "Summary",
        headerField: "Id"
      },
    },
  },
  methods: {
    actionFailure: function() {
      var span = document.createElement('span');

      this.$refs.KanbanObj.ej2Instances.element.parentNode.insertBefore(span,
      this.$refs.KanbanObj.ej2Instances.element);
      span.style.color = '#FF0000'
      span.innerHTML = 'Server exception: 404 Not found';
    }
  }
}
</script>
```

```
<style>
@import '../node_modules/@syncfusion/ej2-base/styles/material.css';
@import '../node_modules/@syncfusion/ej2-buttons/styles/material.css';
@import '../node_modules/@syncfusion/ej2-layouts/styles/material.css';
@import '../node_modules/@syncfusion/ej2-dropdowns/styles/material.css';
@import '../node_modules/@syncfusion/ej2-inputs/styles/material.css';
@import '../node_modules/@syncfusion/ej2-navigations/styles/material.css';
@import '../node_modules/@syncfusion/ej2-popups/styles/material.css';
@import '../node_modules/@syncfusion/ej2-vue-kanban/styles/material.css';
</style>
```

{% previewsample "page.domainurl/code-snippet/kanban/error-cs1" %}

The [actionFailure](#) event will be triggered not only for the server errors, but

also when there is an exception while processing the kanban actions.

#### Loading data via ajax

You can use Kanban [dataSource](#) property to bind the datasource to Kanban from external ajax request. In the following code, we have fetched the datasource from the server using ajax request and provided that to the [dataSource](#) property by using the **onSuccess** event of ajax.

#### APP.VUE

```
<template>
  <div id="app">
    <ejs-button v-on:click.native="btnClick">Load Data</ejs-button>
    <ejs-kanban ref="kanbanObj" id="kanban" keyField="ShipCountry"
      :cardSettings="cardSettings">
      <e-columns>
        <e-column headerText="Denmark" keyField="Denmark"></e-column>
        <e-column headerText="Brazil" keyField="Brazil"></e-column>
        <e-column headerText="Switzerland" keyField="Switzerland"></e-
column>
        <e-column headerText="Germany" keyField="Germany"></e-column>
      </e-columns>
    </ejs-kanban>
  </div>
</template>
<script>
import Vue from "vue";
import { KanbanPlugin } from '@syncfusion/ej2-vue-kanban';
import { ButtonPlugin } from '@syncfusion/ej2-vue-buttons';
import { Ajax } from '@syncfusion/ej2-base';
Vue.use(KanbanPlugin);
Vue.use(ButtonPlugin);
export default {
  data: function() {
    return {
      cardSettings: {
        contentField: "ShippedDate",
        headerField: "OrderID"
      },
    };
  },
  methods: {
```

```

    btnClick: function (args){
        var kanbanData = this.$refs.kanbanObj.ej2Instances;
        var ajax = new
    Ajax("https://ej2services.syncfusion.com/production/web-
    services/api/Orders", "GET");
        ajax.send();
        ajax.onSuccess = function (result) {
            kanbanData.dataSource = JSON.parse(result);
        };
    }
}
</script>
<style>
@import '../node_modules/@syncfusion/ej2-base/styles/material.css';
@import '../node_modules/@syncfusion/ej2-buttons/styles/material.css';
@import '../node_modules/@syncfusion/ej2-layouts/styles/material.css';
@import '../node_modules/@syncfusion/ej2-dropdowns/styles/material.css';
@import '../node_modules/@syncfusion/ej2-inputs/styles/material.css';
@import '../node_modules/@syncfusion/ej2-navigations/styles/material.css';
@import '../node_modules/@syncfusion/ej2-popups/styles/material.css';
@import '../node_modules/@syncfusion/ej2-vue-kanban/styles/material.css';
</style>

```

{% previewsample "page.domainurl/code-snippet/kanban/ajax-cs1" %}

\* If you bind the dataSource from this way, then it acts like a local dataSource. So you cannot perform any server-side crud actions.

### Swimlane in Vue Kanban component

Swimlanes are horizontal categorizations of cards on the Kanban board. It is used for grouping of cards, which brings transparency to the workflow process.

#### Render swimlane row

Cards can be grouped based on **keyField** and displayed in rows, which are separated by columns. It is mandatory to define the **keyField** that is mapped from the datasource for rendering swimlane rows in the Kanban board.

#### APP.VUE

```

<template>
  <div id="app">
    <ejs-kanban id="kanban" keyField="Status" :dataSource="kanbanData"
      :cardSettings="cardSettings" :swimlaneSettings="swimlaneSettings">
      <e-columns>
        <e-column headerText="To Do" keyField="Open"></e-column>
        <e-column headerText="In Progress" keyField="InProgress"></e-
column>
        <e-column headerText="Testing" keyField="Testing"></e-column>
        <e-column headerText="Done" keyField="Close"></e-column>
      </e-columns>
    </ejs-kanban>
  </div>
</template>
<script>

```

```

import Vue from "vue";
import { KanbanPlugin } from '@syncfusion/ej2-vue-kanban';
import { extend } from '@syncfusion/ej2-base';
import { kanbanData } from './datasource.js';
Vue.use(KanbanPlugin);
export default {
  data: function() {
    return {
      kanbanData: extend([], kanbanData, null, true),
      cardSettings: {
        contentField: "Summary",
        headerField: "Id",
      },
      swimlaneSettings: {
        keyField: "Assignee"
      }
    };
  },
};
</script>
<style>
@import '../node_modules/@syncfusion/ej2-base/styles/material.css';
@import '../node_modules/@syncfusion/ej2-buttons/styles/material.css';
@import '../node_modules/@syncfusion/ej2-layouts/styles/material.css';
@import '../node_modules/@syncfusion/ej2-dropdowns/styles/material.css';
@import '../node_modules/@syncfusion/ej2-inputs/styles/material.css';
@import '../node_modules/@syncfusion/ej2-navigations/styles/material.css';
@import '../node_modules/@syncfusion/ej2-popups/styles/material.css';
@import '../node_modules/@syncfusion/ej2-vue-kanban/styles/material.css';
</style>

```

{% previewsample "page.domainurl/code-snippet/kanban/swimlane-key-cs1" %}

### Custom row text

Customize the swimlane row header text by using the `textField` property mapped from `datasource`.

It is not mandatory to define the `textField` to `swimlaneSettings`. It will automatically consider the `keyField` to swimlane row header text.

If the mapping `textField` key is not present in the `datasource`, it will consider the swimlane `keyField` as swimlane row header text.

### APP.VUE

```

<template>
  <div id="app">
    <ejs-kanban id="kanban" keyField="Status" :dataSource="kanbanData"
      :cardSettings="cardSettings" :swimlaneSettings="swimlaneSettings">
      <e-columns>
        <e-column headerText="To Do" keyField="Open"></e-column>
        <e-column headerText="In Progress" keyField="InProgress"></e-
column>
        <e-column headerText="Testing" keyField="Testing"></e-column>
        <e-column headerText="Done" keyField="Close"></e-column>
      </e-columns>
    </ejs-kanban>
  </div>
</template>

```

```

</div>
</template>
<script>
import Vue from "vue";
import { KanbanPlugin } from '@syncfusion/ej2-vue-kanban';
import { extend } from '@syncfusion/ej2-base';
import { kanbanData } from './datasource.js';
Vue.use(KanbanPlugin);
export default {
  data: function() {
    return {
      kanbanData: extend([], kanbanData, null, true),
      cardSettings: {
        contentField: "Summary",
        headerField: "Id",
      },
      swimlaneSettings: {
        keyField: "Assignee",
        textField: 'AssigneeName'
      }
    };
  },
}
</script>
<style>
@import '../node_modules/@syncfusion/ej2-base/styles/material.css';
@import '../node_modules/@syncfusion/ej2-buttons/styles/material.css';
@import '../node_modules/@syncfusion/ej2-layouts/styles/material.css';
@import '../node_modules/@syncfusion/ej2-dropdowns/styles/material.css';
@import '../node_modules/@syncfusion/ej2-inputs/styles/material.css';
@import '../node_modules/@syncfusion/ej2-navigations/styles/material.css';
@import '../node_modules/@syncfusion/ej2-popups/styles/material.css';
@import '../node_modules/@syncfusion/ej2-vue-kanban/styles/material.css';
</style>

```

{% previewsample "page.domainurl/code-snippet/kanban/swimlane-text-cs1" %}

### Template

You can customize the Kanban swimlane row by using the `template` property, which is specified within the `swimlaneSettings` property. In this demo, the swimlane header is customized with HTML element.

### APP.VUE

```

<template>
  <div id="app">
    <ejs-kanban id="kanban" keyField="Status" :dataSource="kanbanData"
      :cardSettings="cardSettings" :swimlaneSettings="swimlaneSettings">
      <e-columns>
        <e-column headerText="To Do" keyField="Open"></e-column>
        <e-column headerText="In Progress" keyField="InProgress"></e-
column>
        <e-column headerText="Testing" keyField="Testing"></e-column>
        <e-column headerText="Done" keyField="Close"></e-column>
      </e-columns>
    </ejs-kanban>
  </div>

```

```

</template>
<script>
import Vue from "vue";
import { KanbanPlugin } from '@syncfusion/ej2-vue-kanban';
import { extend } from '@syncfusion/ej2-base';
import { kanbanData } from './datasource.js';
Vue.use(KanbanPlugin);
export default {
  data: function() {
    return {
      kanbanData: extend([], kanbanData, null, true),
      cardSettings: {
        contentField: "Summary",
        headerField: "Id",
      },
      swimlaneSettings: {
        template: function() {
          return {
            template: Vue.component('swimlaneTemplate', {
              template: `<div class='swimlane-template e-swimlane-
template-table'>
                                <div class="e-swimlane-row-
text">
<span>{{data.textField}}</span></div>
                                </div>`,
              data() {
                return {
                  data: {}
                };
              },
              methods: {
                image: function(data) {
                  return data.keyField + '.png';
                }
              }
            })
          }
        }
      }
    };
  },
};
</script>
<style>
@import '../node_modules/@syncfusion/ej2-base/styles/material.css';
@import '../node_modules/@syncfusion/ej2-buttons/styles/material.css';
@import '../node_modules/@syncfusion/ej2-layouts/styles/material.css';
@import '../node_modules/@syncfusion/ej2-dropdowns/styles/material.css';
@import '../node_modules/@syncfusion/ej2-inputs/styles/material.css';
@import '../node_modules/@syncfusion/ej2-navigations/styles/material.css';
@import '../node_modules/@syncfusion/ej2-popups/styles/material.css';
@import '../node_modules/@syncfusion/ej2-vue-kanban/styles/material.css';
.swimlane-template {
  display: inline-block;
  font-size: 15px;
  font-weight: 500;

```

```

}
.swimlane-template img {
  height: 24px;
  width: 24px;
  border-radius: 50%;
}
.swimlane-template span {
  padding-left: 5px;
  vertical-align: middle;
}
.e-kanban .e-kanban-content .e-content-row.e-swimlane-row .e-content-cells
.e-swimlane-header .e-item-count {
  padding: 4px;
}
</style>

```

{% previewsample "page.domainurl/code-snippet/kanban/swimlane-template-cs1" %}

### Sorting

Swimlane rows are rendered on descending order when using the `sortBy` property set to `Descending` order. By default, swimlane rows are rendered by **Ascending** order.

### APP.VUE

```

<template>
  <div id="app">
    <ejs-kanban id="kanban" keyField="Status" :dataSource="kanbanData"
      :cardSettings="cardSettings" :swimlaneSettings="swimlaneSettings">
      <e-columns>
        <e-column headerText="To Do" keyField="Open"></e-column>
        <e-column headerText="In Progress" keyField="InProgress"></e-
column>
        <e-column headerText="Testing" keyField="Testing"></e-column>
        <e-column headerText="Done" keyField="Close"></e-column>
      </e-columns>
    </ejs-kanban>
  </div>
</template>
<script>
import Vue from "vue";
import { KanbanPlugin } from '@syncfusion/ej2-vue-kanban';
import { extend } from '@syncfusion/ej2-base';
import { kanbanData } from './datasource.js';
Vue.use(KanbanPlugin);
export default {
  data: function() {
    return {
      kanbanData: extend([], kanbanData, null, true),
      cardSettings: {
        contentField: "Summary",
        headerField: "Id",
      },
      swimlaneSettings: {
        keyField: 'Assignee',
        sortBy: 'Descending'
      }
    }
  }
}

```

```

    };
  },
}
</script>
<style>
@import '../node_modules/@syncfusion/ej2-base/styles/material.css';
@import '../node_modules/@syncfusion/ej2-buttons/styles/material.css';
@import '../node_modules/@syncfusion/ej2-layouts/styles/material.css';
@import '../node_modules/@syncfusion/ej2-dropdowns/styles/material.css';
@import '../node_modules/@syncfusion/ej2-inputs/styles/material.css';
@import '../node_modules/@syncfusion/ej2-navigations/styles/material.css';
@import '../node_modules/@syncfusion/ej2-popups/styles/material.css';
@import '../node_modules/@syncfusion/ej2-vue-kanban/styles/material.css';
</style>

```

{% previewsample "page.domainurl/code-snippet/kanban/swimlane-sort-cs1" %}

### Drag-and-drop

By default, The Kanban does not allow dragging the cards across the swimlane rows. Enabling the **dragAndDrop** property allows you to drag the cards across the swimlane rows, which is specified inside **swimlaneSettings** property.

### APP.VUE

```

<template>
  <div id="app">
    <ejs-kanban id="kanban" keyField="Status" :dataSource="kanbanData"
      :cardSettings="cardSettings" :swimlaneSettings="swimlaneSettings">
      <e-columns>
        <e-column headerText="To Do" keyField="Open"></e-column>
        <e-column headerText="In Progress" keyField="InProgress"></e-
column>
        <e-column headerText="Testing" keyField="Testing"></e-column>
        <e-column headerText="Done" keyField="Close"></e-column>
      </e-columns>
    </ejs-kanban>
  </div>
</template>
<script>
import Vue from "vue";
import { KanbanPlugin } from '@syncfusion/ej2-vue-kanban';
import { extend } from '@syncfusion/ej2-base';
import { kanbanData } from './datasource.js';
Vue.use(KanbanPlugin);
export default {
  data: function() {
    return {
      kanbanData: extend([], kanbanData, null, true),
      cardSettings: {
        contentField: "Summary",
        headerField: "Id",
      },
      swimlaneSettings: {
        keyField: 'Assignee',
        allowDragAndDrop: true
      }
    }
  }
}

```



```

    };
  },
}
</script>
<style>
@import '../node_modules/@syncfusion/ej2-base/styles/material.css';
@import '../node_modules/@syncfusion/ej2-buttons/styles/material.css';
@import '../node_modules/@syncfusion/ej2-layouts/styles/material.css';
@import '../node_modules/@syncfusion/ej2-dropdowns/styles/material.css';
@import '../node_modules/@syncfusion/ej2-inputs/styles/material.css';
@import '../node_modules/@syncfusion/ej2-navigations/styles/material.css';
@import '../node_modules/@syncfusion/ej2-popups/styles/material.css';
@import '../node_modules/@syncfusion/ej2-vue-kanban/styles/material.css';
</style>

```

{% previewsample "page.domainurl/code-snippet/kanban/swimlane-drag-cs1" %}

### Create empty row

You can render the empty swimlane row by enabling the `showEmptyRow` property. If mapping `keyField` does not have cards, empty swimlane row will be rendered.

### APP.VUE

```

<template>
  <div id="app">
    <ejs-kanban id="kanban" keyField="Status" :dataSource="kanbanData"
      :cardSettings="cardSettings" :swimlaneSettings="swimlaneSettings">
      <e-columns>
        <e-column headerText="To Do" keyField="Open"></e-column>
        <e-column headerText="In Progress" keyField="InProgress"></e-
column>
        <e-column headerText="Done" keyField="Close"></e-column>
      </e-columns>
    </ejs-kanban>
  </div>
</template>
<script>
import Vue from "vue";
import { KanbanPlugin } from '@syncfusion/ej2-vue-kanban';
import { extend } from '@syncfusion/ej2-base';
import { kanbanData } from './datasource.js';
Vue.use(KanbanPlugin);
export default {
  data: function() {
    return {
      kanbanData: extend([], kanbanData, null, true),
      cardSettings: {
        contentField: "Summary",
        headerField: "Id",
      },
      swimlaneSettings: {
        keyField: 'Assignee',
        showEmptyRow: true
      }
    };
  },
};

```

```

}
</script>
<style>
@import '../node_modules/@syncfusion/ej2-base/styles/material.css';
@import '../node_modules/@syncfusion/ej2-buttons/styles/material.css';
@import '../node_modules/@syncfusion/ej2-layouts/styles/material.css';
@import '../node_modules/@syncfusion/ej2-dropdowns/styles/material.css';
@import '../node_modules/@syncfusion/ej2-inputs/styles/material.css';
@import '../node_modules/@syncfusion/ej2-navigations/styles/material.css';
@import '../node_modules/@syncfusion/ej2-popups/styles/material.css';
@import '../node_modules/@syncfusion/ej2-vue-kanban/styles/material.css';
</style>

```

{% previewsample "page.domainurl/code-snippet/kanban/swimlane-empty-cs1" %}

### Calculate cards count

Users can show or hide the cards count by swimlane row in header when enabling the `showItemCount` property, which is enabled by default on the Kanban board.

Provided localization support for **items** text.

In below demo, disabled on `showItemCount` property on rendering swimlane row without total count.

### APP.VUE

```

<template>
  <div id="app">
    <ejs-kanban id="kanban" keyField="Status" :dataSource="kanbanData"
      :cardSettings="cardSettings" :swimlaneSettings="swimlaneSettings">
      <e-columns>
        <e-column headerText="To Do" keyField="Open"></e-column>
        <e-column headerText="In Progress" keyField="InProgress"></e-
column>
        <e-column headerText="Testing" keyField="Testing"></e-column>
        <e-column headerText="Done" keyField="Close"></e-column>
      </e-columns>
    </ejs-kanban>
  </div>
</template>
<script>
import Vue from "vue";
import { KanbanPlugin } from '@syncfusion/ej2-vue-kanban';
import { extend } from '@syncfusion/ej2-base';
import { kanbanData } from './datasource.js';
Vue.use(KanbanPlugin);
export default {
  data: function() {
    return {
      kanbanData: extend([], kanbanData, null, true),
      cardSettings: {
        contentField: "Summary",
        headerField: "Id",
      },
      swimlaneSettings: {
        keyField: 'Assignee',
        showItemCount: false
      }
    }
  }
}

```

```

    }
    };
  },
}
</script>
<style>
@import '../node_modules/@syncfusion/ej2-base/styles/material.css';
@import '../node_modules/@syncfusion/ej2-buttons/styles/material.css';
@import '../node_modules/@syncfusion/ej2-layouts/styles/material.css';
@import '../node_modules/@syncfusion/ej2-dropdowns/styles/material.css';
@import '../node_modules/@syncfusion/ej2-inputs/styles/material.css';
@import '../node_modules/@syncfusion/ej2-navigations/styles/material.css';
@import '../node_modules/@syncfusion/ej2-popups/styles/material.css';
@import '../node_modules/@syncfusion/ej2-vue-kanban/styles/material.css';
</style>

```

{% previewsample "page.domainurl/code-snippet/kanban/swimlane-total-cs1" %}

### Enable frozen rows

Frozen rows provide an option to make the current swimlane row header text always visible on top of the content while scrolling the Kanban content. The swimlane header text will be changed dynamically, when you scroll to another swimlane row.

By default, the `enableFrozenRows` property is set as `false`. If you wish to show the swimlane frozen rows, you can enable the `enableFrozenRows` property.

This feature support only when using Kanban content scrolling.

### APP.VUE

```

<template>
  <div id="app">
    <ejs-kanban id="kanban" keyField="Status" :dataSource="kanbanData"
      :cardSettings="cardSettings" :swimlaneSettings="swimlaneSettings"
      height="500px">
      <e-columns>
        <e-column headerText="To Do" keyField="Open"></e-column>
        <e-column headerText="In Progress" keyField="InProgress"></e-
column>
        <e-column headerText="Testing" keyField="Testing"></e-column>
        <e-column headerText="Done" keyField="Close"></e-column>
      </e-columns>
    </ejs-kanban>
  </div>
</template>
<script>
import Vue from "vue";
import { KanbanPlugin } from '@syncfusion/ej2-vue-kanban';
import { extend } from '@syncfusion/ej2-base';
import { kanbanData } from './datasource.js';
Vue.use(KanbanPlugin);
export default {
  data: function() {
    return {
      kanbanData: extend([], kanbanData, null, true),
      cardSettings: {

```

```

        contentField: "Summary",
        headerField: "Id",
      },
      swimlaneSettings: {
        keyField: 'Assignee',
        enableFrozenRows: true
      }
    };
  },
}
</script>
<style>
@import '../node_modules/@syncfusion/ej2-base/styles/material.css';
@import '../node_modules/@syncfusion/ej2-buttons/styles/material.css';
@import '../node_modules/@syncfusion/ej2-layouts/styles/material.css';
@import '../node_modules/@syncfusion/ej2-dropdowns/styles/material.css';
@import '../node_modules/@syncfusion/ej2-inputs/styles/material.css';
@import '../node_modules/@syncfusion/ej2-navigations/styles/material.css';
@import '../node_modules/@syncfusion/ej2-popups/styles/material.css';
@import '../node_modules/@syncfusion/ej2-vue-kanban/styles/material.css';
</style>

```

{% previewsample "page.domainurl/code-snippet/kanban/swimlane-enable-frozen-cs1" %}

## Drag and drop in Vue Kanban component

All cards can be dragged and dropped across the columns or within the columns or swimlane row or kanban to an external source and vice versa.

The following drag and drop types are available in the Kanban board.

- Internal drag and drop
- Column drag and drop
- Swimlane drag and drop
- External drag and drop
- Kanban to Kanban
- Kanban to External source and vice versa.

Dropped card position varies based on the `sortSettings` property.

### Internal drag and drop

Allows the user to drag and drop the cards within the kanban board. Based on this, we can categorize into two ways.

- Column drag and drop
- Swimlane drag and drop

### Column drag and drop

By default, all cards can be dragged and dropped across the columns and within the columns. You cannot drag and drop the cards when disabling the `allowDragAndDrop` property.

You can prevent the drag or drop behavior of the particular column by disabling the `allowDrag` or `allowDrop` property.

You can also control the flow of transition cards between the columns by using the `transitionColumns` property.

In the following example, disable the drag and drop behavior on the Kanban board.

#### APP.VUE

```
<template>
  <div id="app">
    <ejs-kanban id="kanban" keyField="Status" :dataSource="kanbanData"
      :cardSettings="cardSettings" :allowDragAndDrop="false">
      <e-columns>
        <e-column headerText="To Do" keyField="Open"></e-column>
        <e-column headerText="In Progress" keyField="InProgress"></e-
column>
        <e-column headerText="Testing" keyField="Testing"></e-column>
        <e-column headerText="Done" keyField="Close"></e-column>
      </e-columns>
    </ejs-kanban>
  </div>
</template>
<script>
import Vue from "vue";
import { KanbanPlugin } from '@syncfusion/ej2-vue-kanban';
import { extend } from '@syncfusion/ej2-base';
import { kanbanData } from './datasource.js';
Vue.use(KanbanPlugin);
export default {
  data: function() {
    return {
      kanbanData: extend([], kanbanData, null, true),
      cardSettings: {
        contentField: "Summary",
        headerField: "Id"
      }
    };
  },
};
</script>
<style>
@import '../node_modules/@syncfusion/ej2-base/styles/material.css';
@import '../node_modules/@syncfusion/ej2-buttons/styles/material.css';
@import '../node_modules/@syncfusion/ej2-layouts/styles/material.css';
@import '../node_modules/@syncfusion/ej2-dropdowns/styles/material.css';
@import '../node_modules/@syncfusion/ej2-inputs/styles/material.css';
@import '../node_modules/@syncfusion/ej2-navigations/styles/material.css';
@import '../node_modules/@syncfusion/ej2-popups/styles/material.css';
@import '../node_modules/@syncfusion/ej2-vue-kanban/styles/material.css';
</style>
```

{% previewsample "page.domainurl/code-snippet/kanban/drag-and-drop-cs1" %}

#### Swimlane drag and drop

By default, Swimlane allows drag and drop across the columns within the swimlane row. Kanban does not allow dragging the cards across the swimlane rows.

Enabling the `dragAndDrop` property allows you to drag the cards across the swimlane rows, which is specified inside the `swimlaneSettings` property.

#### APP.VUE

```
<template>
  <div id="app">
    <ejs-kanban id="kanban" keyField="Status" :dataSource="kanbanData"
      :cardSettings="cardSettings" :swimlaneSettings="swimlaneSettings">
      <e-columns>
        <e-column headerText="To Do" keyField="Open"></e-column>
        <e-column headerText="In Progress" keyField="InProgress"></e-
column>
        <e-column headerText="Testing" keyField="Testing"></e-column>
        <e-column headerText="Done" keyField="Close"></e-column>
      </e-columns>
    </ejs-kanban>
  </div>
</template>
<script>
import Vue from "vue";
import { KanbanPlugin } from '@syncfusion/ej2-vue-kanban';
import { extend } from '@syncfusion/ej2-base';
import { kanbanData } from './datasource.js';
Vue.use(KanbanPlugin);
export default {
  data: function() {
    return {
      kanbanData: extend([], kanbanData, null, true),
      cardSettings: {
        contentField: "Summary",
        headerField: "Id",
      },
      swimlaneSettings: {
        keyField: 'Assignee',
        allowDragAndDrop: true
      }
    };
  },
};
</script>
<style>
@import '../node_modules/@syncfusion/ej2-base/styles/material.css';
@import '../node_modules/@syncfusion/ej2-buttons/styles/material.css';
@import '../node_modules/@syncfusion/ej2-layouts/styles/material.css';
@import '../node_modules/@syncfusion/ej2-dropdowns/styles/material.css';
@import '../node_modules/@syncfusion/ej2-inputs/styles/material.css';
@import '../node_modules/@syncfusion/ej2-navigations/styles/material.css';
@import '../node_modules/@syncfusion/ej2-popups/styles/material.css';
@import '../node_modules/@syncfusion/ej2-vue-kanban/styles/material.css';
</style>
```

{% previewsample "page.domainurl/code-snippet/kanban/swimlane-drag-and-drop-cs1" %}

### External drag and drop

Allows the user to drag and drop the cards from one kanban to another kanban or Kanban to an external source and vice versa.

#### Kanban to kanban

Drag and drop the card from one kanban to another kanban and vice versa. This can be achieved by specifying the `externalDropId` property which is used to specify the id of the dropped kanban element and the `dragStop` event which is used to delete the card on dragged Kanban and add the card on dropped Kanban using the `deleteCard` and `addCard` public methods.

Before adding a card to dropped kanban, you can manually change the card data `headerField` when the same card data `headerField` is dropped to another Kanban.

In the following example, Drag the card from one Kanban and drop it into another kanban using the `dragStop` event. In this event, remove the card from the dragged Kanban by using the `deleteCard` public method and add the card to the dropped Kanban by using the `addCard` public method.

#### APP.VUE

```
<template>
  <div id="app">
    <div class="container-fluid">
      <div class="row">
        <div class="col-sm-6">
          <h4>Kanban A</h4>
          <ejs-kanban id="kanbanA" ref="kanbanObjA" keyField="Status"
:dataSource="kanbanData"
:cardSettings="cardSettings" :externalDropId='externalKanbanADropId'
:dragStop="kanbanDragStopA">
            <e-columns>
              <e-column headerText="To Do" keyField="Open"></e-column>
              <e-column headerText="Done" keyField="Close"></e-column>
            </e-columns>
          </ejs-kanban>
        </div>
        <div class="col-sm-6">
          <h4>Kanban B</h4>
          <ejs-kanban id="kanbanB" ref="kanbanObjB" keyField="Status"
:dataSource="kanbanData"
:cardSettings="cardSettings" :externalDropId='externalKanbanBDropId'
:dragStop="kanbanDragStopB">
            <e-columns>
              <e-column headerText="To Do" keyField="Open"></e-column>
              <e-column headerText="Done" keyField="Close"></e-column>
            </e-columns>
          </ejs-kanban>
        </div>
      </div>
    </div>
  </template>
<script>
import Vue from "vue";
import { KanbanPlugin } from '@syncfusion/ej2-vue-kanban';
import { extend, closest } from '@syncfusion/ej2-base';
```

```

import { kanbanData } from './datasource.js';
Vue.use(KanbanPlugin);
export default {
  data: function() {
    return {
      kanbanData: extend([], kanbanData, null, true),
      cardSettings: {
        contentField: "Summary",
        headerField: "Id"
      },
      externalKanbanADropId: ['#kanbanB'],
      externalKanbanBDropId: ['#kanbanA']
    };
  },
  methods: {
    kanbanDragStopA: function (args) {
      let kanbanBElement = closest(args.event.target as Element,
        '#kanbanB');
      let kanbanObjA = this.$refs.kanbanObjA.ej2Instances;
      let kanbanObjB = this.$refs.kanbanObjB.ej2Instances;
      if (kanbanBElement) {
        kanbanObjA.deleteCard(args.data);
        args.data.forEach((card, i) => {
          const index = kanbanObjB.kanbanData.findIndex((colData) =>
            colData[kanbanObjB.cardSettings.headerField] ===
            card[kanbanObjB.cardSettings.headerField]);
          if (index !== -1) {
            card[kanbanObjB.cardSettings.headerField] =
              Math.max(...kanbanObjB.kanbanData.map(
                (obj) =>
                  parseInt(obj[kanbanObjB.cardSettings.headerField], 10))) + ++i;
          }
        });
        kanbanObjB.addCard(args.data, args.dropIndex);
        args.cancel = true;
      }
    },
    kanbanDragStopB: function (args) {
      let kanbanAElement = closest(args.event.target, '#kanbanA');
      let kanbanObjA = this.$refs.kanbanObjA.ej2Instances;
      let kanbanObjB = this.$refs.kanbanObjB.ej2Instances;
      if (kanbanAElement) {
        kanbanObjB.deleteCard(args.data);
        args.data.forEach((card, i) => {
          const index = kanbanObjA.kanbanData.findIndex((colData) =>
            colData[kanbanObjA.cardSettings.headerField] ===
            card[kanbanObjA.cardSettings.headerField]);
          if (index !== -1) {
            card[kanbanObjA.cardSettings.headerField] =
              Math.max(...kanbanObjA.kanbanData.map(
                (obj) =>
                  parseInt(obj[kanbanObjA.cardSettings.headerField], 10))) + ++i;
          }
        });
        kanbanObjA.addCard(args.data, args.dropIndex);
        args.cancel = true;
      }
    }
  }
}

```



```

    }
  }
}
</script>
<style>
@import '../node_modules/@syncfusion/ej2-base/styles/material.css';
@import '../node_modules/@syncfusion/ej2-buttons/styles/material.css';
@import '../node_modules/@syncfusion/ej2-layouts/styles/material.css';
@import '../node_modules/@syncfusion/ej2-dropdowns/styles/material.css';
@import '../node_modules/@syncfusion/ej2-inputs/styles/material.css';
@import '../node_modules/@syncfusion/ej2-navigations/styles/material.css';
@import '../node_modules/@syncfusion/ej2-popups/styles/material.css';
@import '../node_modules/@syncfusion/ej2-vue-kanban/styles/material.css';
.row {
  display: flex;
}
</style>

```

{% previewsample "page.domainurl/code-snippet/kanban/kanban-to-kanban-cs1" %}

#### Treeview to Kanban

Drag the card from the Kanban board and drop it to the Treeview component and vice versa.

In the following sample, remove the data from the Kanban board using the `deleteCard` public method and add to the Treeview component using the `addNodes` public method at Kanban `dragStop` event when dragging the card and dropping it to the Treeview component. Remove the data from Treeview using the `removeNodes` public method and add to Kanban board using the `openDialog` public method when dragging the list from the Treeview component and dropping it to the kanban board.

#### APP.VUE

```

<template>
  <div id="app">
    <div class="container-fluid">
      <div class="row">
        <div class="col-sm-6">
          <h4>Kanban</h4>
          <ejs-kanban id="kanban" ref="kanbanObj" keyField="Status"
:dataSource="kanbanData"
:cardSettings="cardSettings" :externalDropId='externalKanbanDropId'
:dragStop="kanbanDragStop">
            <e-columns>
              <e-column headerText="To Do" keyField="Open"></e-column>
              <e-column headerText="Done" keyField="Close"></e-column>
            </e-columns>
          </ejs-kanban>
        </div>
        <div class="col-sm-6">
          <h4>TreeView</h4>
          <ejs-treeview id='treeView' ref="treeViewObj"
:nodeTemplate="treeTemplate" :fields='treeViewFields' :allowDragAndDrop=true
:nodeDragStop="onItemDragStop"></ejs-treeview>
        </div>
      </div>
    </div>
  </div>
</template>

```

```

</template>
<script>
import Vue from "vue";
import { KanbanPlugin } from '@syncfusion/ej2-vue-kanban';
import { extend, closest } from '@syncfusion/ej2-base';
import { kanbanData, treeViewData } from './datasource.js';
Vue.use(KanbanPlugin);
import { TreeViewPlugin } from "@syncfusion/ej2-vue-navigations";
Vue.use(TreeViewPlugin);
var treeVue = Vue.component("tree-template", {
  template: '<div id="treelist"><div id="treeviewlist">{{data.Id}} - {{data.Status}}</div></div>',
  data() {
    return {
      data: {}
    };
  }
});
export default {
  data: function() {
    return {
      kanbanData: extend([], kanbanData, null, true),
      cardSettings: {
        contentField: "Summary",
        headerField: "Id"
      },
      externalKanbanDropId: ['#treeView'],
      treeViewFields: { dataSource: treeViewData, id: 'Id', text: 'Status'
    },
    treeTemplate: function(e) {
      return { template: treeVue }
    }
  },
  methods: {
    kanbanDragStop: function (args) {
      let treeViewElement = closest(args.event.target as Element, '#treeView');
      let kanbanObj = this.$refs.kanbanObj.ej2Instances;
      let treeObj = this.$refs.treeViewObj.ej2Instances;
      if (treeViewElement) {
        kanbanObj.deleteCard(args.data);
        treeObj.addNodes(args.data);
        args.cancel = true;
      }
    },
    onItemDragStop: function (args) {
      let kanbanElement = closest(args.event.target as Element, '#kanban');
      let kanbanObj = this.$refs.kanbanObj.ej2Instances;
      let treeObj = this.$refs.treeViewObj.ej2Instances;
      if (kanbanElement) {
        let treeData =
          treeObj.fields.dataSource as { [key: string]: Object }[];
        const filteredData =
          treeData.filter((item) => item.Id ===
parseInt(args.draggedNodeData.id as string, 10));

```

```

        treeObj.removeNodes([args.draggedNodeData.id] as string[]);
        kanbanObj.openDialog('Add', filteredData[0]);
        args.cancel = true;
    }
}
}
}
</script>
<style>
@import '../node_modules/@syncfusion/ej2-base/styles/material.css';
@import '../node_modules/@syncfusion/ej2-buttons/styles/material.css';
@import '../node_modules/@syncfusion/ej2-layouts/styles/material.css';
@import '../node_modules/@syncfusion/ej2-dropdowns/styles/material.css';
@import '../node_modules/@syncfusion/ej2-inputs/styles/material.css';
@import '../node_modules/@syncfusion/ej2-navigations/styles/material.css';
@import '../node_modules/@syncfusion/ej2-popups/styles/material.css';
@import '../node_modules/@syncfusion/ej2-vue-kanban/styles/material.css';
.row {
    display: flex;
}
</style>

```

{% previewsample "page.domainurl/code-snippet/kanban/kanban-to-treeview-cs1" %}

#### *Schedule to Kanban*

Drag the card from the Kanban board and drop it to the Schedule component and vice versa.

In the following sample, remove the data from the Kanban board using the `deleteCard` public method and add to the schedule component using the `addNodes` public method at Kanban `dragStop` event when dragging the card and dropping it to the Treeview component. Remove the data from Treeview using the `removeNodes` public method and add to Kanban board using the `addCard` public method when dragging the list from the Treeview component and dropping it to the kanban board.

#### **APP.VUE**

```

<template>
  <div id="app">
    <div class="container-fluid">
      <div class="row">
        <div class="col-sm-6" style="width: 30%">
          <h4>Kanban</h4>
          <ejs-kanban id="kanban" ref="kanbanObj" keyField="DepartmentName"
            :dataSource="kanbanData"
            :cardSettings="cardSettings" :externalDropId='externalKanbanDropId'
            :dragStop="kanbanDragStop">
            <e-columns>
              <e-column headerText="GENERAL" keyField="GENERAL"></e-column>
            </e-columns>
          </ejs-kanban>
        </div>
        <div class="col-sm-6" style="width: 70%">
          <h4>TreeView</h4>
          <ejs-schedule id='schedule' ref="scheduleObj" height="650px"
            :cssClass='cssClass' :selectedDate='selectedDate'
            :eventSettings='eventSettings'

```

```

        :group='group' :currentView='currentView'
:resourceHeaderTemplate='resourceHeaderTemplate' :dragStop="onItemDragStop">
    <e-views>
        <e-view option="TimelineDay"></e-view>
        <e-view option="TimelineMonth"></e-view>
    </e-views>
    <e-resources>
        <e-resource field='DepartmentID' title='Department'
name='Departments' :dataSource='departmentDataSource'
        textField='Text' idField='Id' colorField='Color'>
        </e-resource>
        <e-resource field='ConsultantID' title='Consultant'
name='Consultants' :dataSource='consultantDataSource'
        textField='Text' idField='Id'
groupIDField='GroupId' colorField='Color'>
        </e-resource>
    </e-resources>
</ejs-schedule>
</div>
</div>
</div>
</div>
</template>
<script>
import Vue from "vue";
import { KanbanPlugin } from '@syncfusion/ej2-vue-kanban';
import { extend, closest } from '@syncfusion/ej2-base';
import { kanbanData, scheduleData } from './datasource.js';
Vue.use(KanbanPlugin);
import { SchedulePlugin, TimelineViews, TimelineMonth, View, Resize,
DragAndDrop } from '@syncfusion/ej2-vue-schedule';
Vue.use(SchedulePlugin);
var resourceHeaderVue = Vue.component("resource-headerTemplate", {
    template: '<div className="template-wrap"><div class="specialist-
category"><div v-if=getConsultantImageName(data)></div><div
class="specialist-name">' +
        '{{getConsultantName(data)}}</div><div class="specialist-
designation">{{getConsultantDesignation(data)}}</div></div></div>',
    data() {
        return {
            data: {}
        };
    },
    methods: {
        getConsultantName: function (data) {
            let value = JSON.parse(JSON.stringify(data));
            return (value.resourceData) ?
value.resourceData[value.resource.textField] : value.resourceName;
        },
        getConsultantImageName: function (data) {
            let value = JSON.parse(JSON.stringify(data));
            let resourceName = (value.resourceData) ?
value.resourceData[value.resource.textField] : value.resourceName;
            if (resourceName === 'GENERAL' || resourceName === 'DENTAL') {
                return false;
            } else {
                return true;
            }
        }
    }
});

```

```

    },
    getConsultantDesignation: function (data) {
      let value = JSON.parse(JSON.stringify(data));
      var resourceName =
value.resourceData[value.resource.textField];
      if (resourceName === "GENERAL" || resourceName === "DENTAL") {
        return '';
      } else {
        return value.resourceData.Designation;
      }
    }
  }
});
export default {
  data: function() {
    return {
      kanbanData: extend([], kanbanData, null, true),
      cardSettings: {
        contentField: "Name",
        headerField: "Id"
      },
      externalKanbanDropId: ['#schedule'],
      eventSettings: {
        dataSource: extend([], scheduleData, null, true),
        fields: {
          subject: { title: 'Patient Name', name: 'Name' },
          startTime: { title: "From", name: "StartTime" },
          endTime: { title: "To", name: "EndTime" },
          description: { title: 'Reason', name: 'Description' }
        },
      },
      selectedDate: new Date(2018, 7, 1),
      currentView: 'TimelineDay',
      cssClass: 'schedule-drag-drop',
      group: {
        enableCompactView: false,
        resources: ['Departments', 'Consultants']
      },
      departmentDataSource: [
        { Text: 'GENERAL', Id: 1, Color: '#bbdc00' },
        { Text: 'DENTAL', Id: 2, Color: '#9e5fff' }
      ],
      consultantDataSource: [
        { Text: 'Alice', Id: 1, GroupId: 1, Color: '#bbdc00', Designation:
'Cardiologist' },
        { Text: 'Nancy', Id: 2, GroupId: 2, Color: '#9e5fff', Designation:
'Orthodontist' },
        { Text: 'Robert', Id: 3, GroupId: 1, Color: '#bbdc00',
Designation: 'Optometrist' },
        { Text: 'Robson', Id: 4, GroupId: 2, Color: '#9e5fff',
Designation: 'Periodontist' },
        { Text: 'Laura', Id: 5, GroupId: 1, Color: '#bbdc00', Designation:
'Orthopedic' },
        { Text: 'Margaret', Id: 6, GroupId: 2, Color: '#9e5fff',
Designation: 'Endodontist' }
      ],
    },
  },

```

```

        resourceHeaderTemplate: function (e) {
            return { template: resourceHeaderVue }
        },
    },
    methods: {
        kanbanDragStop: function (args) {
            let scheduleElement: Element = closest(args.event.target as Element,
            '#schedule');
            let kanbanObj = this.$refs.kanbanObj.ej2Instances;
            let scheduleObj = this.$refs.scheduleObj.ej2Instances;
            if (scheduleElement) {
                kanbanObj.deleteCard(args.data);
                scheduleObj.openEditor(args.data[0], 'Add', true);
                args.cancel = true;
            }
        },
        onItemDragStop: function (args) {
            let kanbanElement: Element = closest(args.event.target as Element,
            '#kanban');
            let kanbanObj = this.$refs.kanbanObj.ej2Instances;
            let scheduleObj = this.$refs.scheduleObj.ej2Instances;
            if (kanbanElement) {
                scheduleObj.deleteEvent(args.data.Id);
                removeClass([scheduleObj.element.querySelector('.e-selected-cell')], 'e-selected-cell');
                kanbanObj.openDialog('Add', args.data);
                args.cancel = true;
            }
        },
    },
    provide: {
        schedule: [TimelineViews, TimelineMonth, Resize, DragAndDrop]
    }
}
</script>
<style>
@import '../node_modules/@syncfusion/ej2-base/styles/material.css';
@import '../node_modules/@syncfusion/ej2-buttons/styles/material.css';
@import '../node_modules/@syncfusion/ej2-layouts/styles/material.css';
@import '../node_modules/@syncfusion/ej2-dropdowns/styles/material.css';
@import '../node_modules/@syncfusion/ej2-inputs/styles/material.css';
@import '../node_modules/@syncfusion/ej2-navigations/styles/material.css';
@import '../node_modules/@syncfusion/ej2-popups/styles/material.css';
@import '../node_modules/@syncfusion/ej2-calendars/styles/material.css';
@import '../node_modules/@syncfusion/ej2-excel-export/styles/material.css';
@import '../node_modules/@syncfusion/ej2-file-utils/styles/material.css';
@import '../node_modules/@syncfusion/ej2-schedule/styles/material.css';
@import '../node_modules/@syncfusion/ej2-compression/styles/material.css';
@import '../node_modules/@syncfusion/ej2-vue-kanban/styles/material.css';
.row {
    display: flex;
}
</style>

```

{% previewsample "page.domainurl/code-snippet/kanban/kanban-to-schedule-cs1" %}

## Sort in Vue Kanban component

The Kanban provides built-in support to arrange the cards in their columns based on the JSON data order and drop the cards in the columns based on the dropped clone. Initially, users can change the arrangement of cards in the columns and position of the dropped card by using the [sortBy](#) property. The [sortBy](#) property contains three enumeration values as follows.

- Index
- DataSourceOrder
- Custom

### Index

SortBy **Index** property can be used with or without [field](#) mapping.

#### Index without field mapping

By default, SortBy **Index** property support without any [field](#) mapping. In this behavior, cards are loaded based on the JSON data order and cards are dropped based on the dropped clone.

### APP.VUE

```
<template>
  <div id="app">
    <ejs-kanban id="kanban" keyField="Status" :dataSource="kanbanData"
      :cardSettings="cardSettings">
      <e-columns>
        <e-column headerText="To Do" keyField="Open"></e-column>
        <e-column headerText="In Progress" keyField="InProgress"></e-
column>
        <e-column headerText="Testing" keyField="Testing"></e-column>
        <e-column headerText="Done" keyField="Close"></e-column>
      </e-columns>
    </ejs-kanban>
  </div>
</template>
<script>
import Vue from "vue";
import { KanbanPlugin } from '@syncfusion/ej2-vue-kanban';
import { extend } from '@syncfusion/ej2-base';
import { kanbanData } from './datasource.js';
Vue.use(KanbanPlugin);
export default {
  data: function() {
    return {
      kanbanData: extend([], kanbanData, null, true),
      cardSettings: {
        contentField: "Summary",
        headerField: "Id"
      }
    };
  },
};
</script>
<style>
@import '../node_modules/@syncfusion/ej2-base/styles/material.css';
@import '../node_modules/@syncfusion/ej2-buttons/styles/material.css';
```

```
@import '../node_modules/@syncfusion/ej2-layouts/styles/material.css';
@import '../node_modules/@syncfusion/ej2-dropdowns/styles/material.css';
@import '../node_modules/@syncfusion/ej2-inputs/styles/material.css';
@import '../node_modules/@syncfusion/ej2-navigations/styles/material.css';
@import '../node_modules/@syncfusion/ej2-popups/styles/material.css';
@import '../node_modules/@syncfusion/ej2-vue-kanban/styles/material.css';
</style>
```

{% previewsample "page.domainurl/code-snippet/kanban/index-cs1" %}

### *Index with field mapping*

SortBy **Index** property also supports with [field](#) mapping. In this behavior, cards are loaded based on mapping **field** values, and cards are dropped based on the dropped clone.

Cards are placed in a particular position in the columns where you can drop the cards by specifying the [field](#) property, which is mapped from the data source. This property allows the users to drop the cards in the Kanban board where the dropped clone is created exactly. It is also helpful to render the cards based on the [field](#) property value.

The [field](#) property mapping key value must be in **number** format.

The following cases will dynamically change their [field](#) value when dropping the cards.

- If the cell has no cards, the dropped card [field](#) value does not change.
- If the cell has one card and dropped a card to the last position or previous/next cards that do not have continuous order, then the dropped card [field](#) value will be changed based on their previous card value.
- If the cell has one card and dropped a card on the previous position, then it will compare both the values, and the dropped card [field](#) value will be changed if the cards have continuous order otherwise values will not be changed.
- When the previous and next cards do not have continuous order, the dropped card [field](#) value will be changed based on the previous card value.
- When the previous and next cards have continuous order or odd/even value, then the [field](#) value of the dropped card and the cards followed by the dropped card will be changed based on the **previous** card value with continuous order.

For Example,

### **Continuous Order -**

Consider, Column A has Card A with priority value **1**, Card B with priority value **2**, and Card C with priority value **3**. And Column B has Card D with priority value **5**, then the dropped Card D will be placed between Card A and Card B. Now, the Cards D, B, and C will be dynamically changed to the priority values as **2**, **3**, and **4** respectively.

### **Odd/Even order -**

Consider, Column A has Card A with priority value **1**, Card B with priority value **3**, and Card C with priority value **5**.



and Column B has Card D with priority value 5, then the Dropped Card D will be placed between Card A and Card B. Now, the Cards D, B, and C will be dynamically changed to the priority values as 2, 3, and 5 respectively.

### APP.VUE

```
<template>
  <div id="app">
    <ejs-kanban id="kanban" keyField="Status" :dataSource="kanbanData"
      :cardSettings="cardSettings" :sortSettings="sortSettings">
      <e-columns>
        <e-column headerText="To Do" keyField="Open"></e-column>
        <e-column headerText="In Progress" keyField="InProgress"></e-
column>
        <e-column headerText="Testing" keyField="Testing"></e-column>
        <e-column headerText="Done" keyField="Close"></e-column>
      </e-columns>
    </ejs-kanban>
  </div>
</template>
<script>
import Vue from "vue";
import { KanbanPlugin } from '@syncfusion/ej2-vue-kanban';
import { extend } from '@syncfusion/ej2-base';
import { kanbanData } from './datasource.js';
Vue.use(KanbanPlugin);
export default {
  data: function() {
    return {
      kanbanData: extend([], kanbanData, null, true),
      cardSettings: {
        contentField: "Summary",
        headerField: "Id"
      },
      sortSettings: {
        sortBy: "Index",
        field: "RankId"
      }
    };
  },
}
</script>
<style>
@import '../node_modules/@syncfusion/ej2-base/styles/material.css';
@import '../node_modules/@syncfusion/ej2-buttons/styles/material.css';
@import '../node_modules/@syncfusion/ej2-layouts/styles/material.css';
@import '../node_modules/@syncfusion/ej2-dropdowns/styles/material.css';
@import '../node_modules/@syncfusion/ej2-inputs/styles/material.css';
@import '../node_modules/@syncfusion/ej2-navigations/styles/material.css';
@import '../node_modules/@syncfusion/ej2-popups/styles/material.css';
@import '../node_modules/@syncfusion/ej2-vue-kanban/styles/material.css';
</style>
```

```
{% previewsample "page.domainurl/code-snippet/kanban/index-field-cs1" %}
```

### DataSource Order

The SortBy **DataSourceOrder** property does not require any [field](#) mapping. In this behavior, cards are loaded based on the JSON data order, and also cards are dropped based on the JSON data order.

### APP.VUE

```
<template>
  <div id="app">
    <ejs-kanban id="kanban" keyField="Status" :dataSource="kanbanData"
      :cardSettings="cardSettings" :sortSettings="sortSettings">
      <e-columns>
        <e-column headerText="To Do" keyField="Open"></e-column>
        <e-column headerText="In Progress" keyField="InProgress"></e-
column>
        <e-column headerText="Testing" keyField="Testing"></e-column>
        <e-column headerText="Done" keyField="Close"></e-column>
      </e-columns>
    </ejs-kanban>
  </div>
</template>
<script>
import Vue from "vue";
import { KanbanPlugin } from '@syncfusion/ej2-vue-kanban';
import { extend } from '@syncfusion/ej2-base';
import { kanbanData } from './datasource.js';
Vue.use(KanbanPlugin);
export default {
  data: function() {
    return {
      kanbanData: extend([], kanbanData, null, true),
      cardSettings: {
        contentField: "Summary",
        headerField: "Id"
      },
      sortSettings: {
        sortBy: "DataSourceOrder"
      }
    };
  },
}
</script>
<style>
@import '../node_modules/@syncfusion/ej2-base/styles/material.css';
@import '../node_modules/@syncfusion/ej2-buttons/styles/material.css';
@import '../node_modules/@syncfusion/ej2-layouts/styles/material.css';
@import '../node_modules/@syncfusion/ej2-dropdowns/styles/material.css';
@import '../node_modules/@syncfusion/ej2-inputs/styles/material.css';
@import '../node_modules/@syncfusion/ej2-navigations/styles/material.css';
@import '../node_modules/@syncfusion/ej2-popups/styles/material.css';
@import '../node_modules/@syncfusion/ej2-vue-kanban/styles/material.css';
</style>
```

{% previewsample "page.domainurl/code-snippet/kanban/data-source-order-cs1" %}

## Custom

*Custom with field mapping*

The SortBy **Custom** property must require datasource [field](#) mapping. In this behavior, cards are loaded based on the [field](#) mapping value and also cards are dropped based on the [field](#) mapping value.

**APP.VUE**

```

<template>
  <div id="app">
    <ejs-kanban id="kanban" keyField="Status" :dataSource="kanbanData"
      :cardSettings="cardSettings" :sortSettings="sortSettings">
      <e-columns>
        <e-column headerText="To Do" keyField="Open"></e-column>
        <e-column headerText="In Progress" keyField="InProgress"></e-
column>
        <e-column headerText="Testing" keyField="Testing"></e-column>
        <e-column headerText="Done" keyField="Close"></e-column>
      </e-columns>
    </ejs-kanban>
  </div>
</template>
<script>
import Vue from "vue";
import { KanbanPlugin } from '@syncfusion/ej2-vue-kanban';
import { extend } from '@syncfusion/ej2-base';
import { kanbanData } from './datasource.js';
Vue.use(KanbanPlugin);
export default {
  data: function() {
    return {
      kanbanData: extend([], kanbanData, null, true),
      cardSettings: {
        contentField: "Summary",
        headerField: "Id"
      },
      sortSettings: {
        sortBy: "Custom",
        field: "Summary"
      }
    };
  },
};
</script>
<style>
@import '../node_modules/@syncfusion/ej2-base/styles/material.css';
@import '../node_modules/@syncfusion/ej2-buttons/styles/material.css';
@import '../node_modules/@syncfusion/ej2-layouts/styles/material.css';
@import '../node_modules/@syncfusion/ej2-dropdowns/styles/material.css';
@import '../node_modules/@syncfusion/ej2-inputs/styles/material.css';
@import '../node_modules/@syncfusion/ej2-navigations/styles/material.css';
@import '../node_modules/@syncfusion/ej2-popups/styles/material.css';
@import '../node_modules/@syncfusion/ej2-vue-kanban/styles/material.css';
</style>

```

{% previewsample "page.domainurl/code-snippet/kanban/custom-mapping-cs1" %}

### Change the direction

Kanban board also provides support for aligning the cards in the columns using the [direction](#) property inside the [sortSettings](#) property. Based on this, cards can be aligned in the columns either in **Ascending** or **Descending** order. Sorting direction will be performed based on [sortBy](#) property.

By default, cards are aligned in the columns based on **Ascending** order.

In the following sample, cards are aligned in **Descending** order.

#### APP.VUE

```
<template>
  <div id="app">
    <ejs-kanban id="kanban" keyField="Status" :dataSource="kanbanData"
      :cardSettings="cardSettings" :sortSettings="sortSettings">
      <e-columns>
        <e-column headerText="To Do" keyField="Open"></e-column>
        <e-column headerText="In Progress" keyField="InProgress"></e-
column>
        <e-column headerText="Testing" keyField="Testing"></e-column>
        <e-column headerText="Done" keyField="Close"></e-column>
      </e-columns>
    </ejs-kanban>
  </div>
</template>
<script>
import Vue from "vue";
import { KanbanPlugin } from '@syncfusion/ej2-vue-kanban';
import { extend } from '@syncfusion/ej2-base';
import { kanbanData } from './datasource.js';
Vue.use(KanbanPlugin);
export default {
  data: function() {
    return {
      kanbanData: extend([], kanbanData, null, true),
      cardSettings: {
        contentField: "Summary",
        headerField: "Id"
      },
      sortSettings: {
        sortBy: "Custom",
        field: "Summary",
        direction: "Descending"
      }
    };
  },
};
</script>
<style>
@import '../node_modules/@syncfusion/ej2-base/styles/material.css';
@import '../node_modules/@syncfusion/ej2-buttons/styles/material.css';
@import '../node_modules/@syncfusion/ej2-layouts/styles/material.css';
@import '../node_modules/@syncfusion/ej2-dropdowns/styles/material.css';
@import '../node_modules/@syncfusion/ej2-inputs/styles/material.css';
@import '../node_modules/@syncfusion/ej2-navigations/styles/material.css';
@import '../node_modules/@syncfusion/ej2-popups/styles/material.css';
@import '../node_modules/@syncfusion/ej2-vue-kanban/styles/material.css';
```

```
</style>
```

```
{% previewsample "page.domainurl/code-snippet/kanban/sort-direction-cs1" %}
```

## Dialog in Vue Kanban component

The Kanban provides built-in support to add, edit and delete a card using dialog module. User can edit a card using the following ways.

- Built-in dialog module
- Custom Fields
- Dialog template

### Default Dialog

When double-click on the cards, the dialog is opened with below fields to edit a card. This dialog contains **Delete**, **Save** and **Cancel** buttons.

- To edit a card, modify the card details and click the **Save** button.
- To delete a card, click **Delete** button.
- Click on the **Cancel** button to cancel the editing action.

The dialog displays with the following fields which mapped to dialog fields by default.

Key | Type | Text

cardSettings.headerField | Input | ID

keyField | DropDown | -

cardSettings.contentField | TextArea | -

cardSettings.priority(If applicable) | Numeric | -

swimlaneSettings.keyField(If applicable) | DropDown | -

### APP.VUE

```
<template>
  <div id="app">
    <ejs-kanban id="kanban" keyField="Status" :dataSource="kanbanData"
      :cardSettings="cardSettings">
      <e-columns>
        <e-column headerText="To Do" keyField="Open"></e-column>
        <e-column headerText="In Progress" keyField="InProgress"></e-
column>
        <e-column headerText="Testing" keyField="Testing"></e-column>
        <e-column headerText="Done" keyField="Close"></e-column>
      </e-columns>
    </ejs-kanban>
  </div>
</template>
<script>
import Vue from "vue";
import { KanbanPlugin } from '@syncfusion/ej2-vue-kanban';
import { extend } from '@syncfusion/ej2-base';
```

```

import { kanbanData } from './datasource.js';
Vue.use(KanbanPlugin);
export default {
  data: function() {
    return {
      kanbanData: extend([], kanbanData, null, true),
      cardSettings: {
        contentField: "Summary",
        headerField: "Id",
      }
    };
  },
};
</script>
<style>
@import '../node_modules/@syncfusion/ej2-base/styles/material.css';
@import '../node_modules/@syncfusion/ej2-buttons/styles/material.css';
@import '../node_modules/@syncfusion/ej2-layouts/styles/material.css';
@import '../node_modules/@syncfusion/ej2-dropdowns/styles/material.css';
@import '../node_modules/@syncfusion/ej2-inputs/styles/material.css';
@import '../node_modules/@syncfusion/ej2-navigations/styles/material.css';
@import '../node_modules/@syncfusion/ej2-popups/styles/material.css';
@import '../node_modules/@syncfusion/ej2-vue-kanban/styles/material.css';
</style>

```

{% previewsample "page.domainurl/code-snippet/kanban/getting-started-key-field-cs1" %}

### Custom Fields

You can change the default fields of dialog using `fields` property inside the `dialogSettings` property. The `key` property used to map the `dataSource` value and rendered the corresponding component based on specified `type` property.

The following types are available in dialog fields.

- String
- Numeric
- TextArea
- DropDown
- TextBox
- Input

If `type` is not defined in the fields, then it renders as the HTML input element in dialog.

### APP.VUE

```

<template>
  <div id="app">
    <ejs-kanban id="kanban" keyField="Status" :dataSource="kanbanData"
      :cardSettings="cardSettings" :dialogSettings="dialogSettings">
      <e-columns>
        <e-column headerText="To Do" keyField="Open"></e-column>
        <e-column headerText="In Progress" keyField="InProgress"></e-
column>
        <e-column headerText="Testing" keyField="Testing"></e-column>

```

```

        <e-column headerText="Done" keyField="Close"></e-column>
    </e-columns>
</ejs-kanban>
</div>
</template>
<script>
import Vue from "vue";
import { KanbanPlugin } from '@syncfusion/ej2-vue-kanban';
import { extend } from '@syncfusion/ej2-base';
import { kanbanData } from './datasource.js';
Vue.use(KanbanPlugin);
export default {
    data: function() {
        return {
            kanbanData: extend([], kanbanData, null, true),
            cardSettings: {
                contentField: "Summary",
                headerField: "Id",
            },
            dialogSettings: {
                fields: [
                    { text: 'ID', key: 'Id', type: 'Input' },
                    { key: 'Category', type: 'DropDown' },
                    { key: 'Title', type: 'Input' },
                    { key: 'Size', type: 'Input' },
                    { key: 'Description', type: 'TextArea' }
                ]
            }
        };
    },
}
</script>
<style>
@import '../node_modules/@syncfusion/ej2-base/styles/material.css';
@import '../node_modules/@syncfusion/ej2-buttons/styles/material.css';
@import '../node_modules/@syncfusion/ej2-layouts/styles/material.css';
@import '../node_modules/@syncfusion/ej2-dropdowns/styles/material.css';
@import '../node_modules/@syncfusion/ej2-inputs/styles/material.css';
@import '../node_modules/@syncfusion/ej2-navigations/styles/material.css';
@import '../node_modules/@syncfusion/ej2-popups/styles/material.css';
@import '../node_modules/@syncfusion/ej2-vue-kanban/styles/material.css';
</style>

```

{% previewsample "page.domainurl/code-snippet/kanban/custom-dialog-cs1" %}

#### Custom Fields label

By default, the fields **key** mapping value is considered as a **label** and you can change this label by using **text** property.

#### APP.VUE

```

<template>
  <div id="app">
    <ejs-kanban id="kanban" keyField="Status" :dataSource="kanbanData"
      :cardSettings="cardSettings" :dialogSettings="dialogSettings">
      <e-columns>

```

```

        <e-column headerText="To Do" keyField="Open"></e-column>
        <e-column headerText="In Progress" keyField="InProgress"></e-
column>
        <e-column headerText="Testing" keyField="Testing"></e-column>
        <e-column headerText="Done" keyField="Close"></e-column>
    </e-columns>
</ejs-kanban>
</div>
</template>
<script>
import Vue from "vue";
import { KanbanPlugin } from '@syncfusion/ej2-vue-kanban';
import { extend } from '@syncfusion/ej2-base';
import { kanbanData } from './datasource.js';
Vue.use(KanbanPlugin);
export default {
    data: function() {
        return {
            kanbanData: extend([], kanbanData, null, true),
            cardSettings: {
                contentField: "Summary",
                headerField: "Id",
            },
            dialogSettings: {
                fields: [
                    { text: 'ID', key: 'Id', type: 'Input' },
                    { key: 'Status', type: 'DropDown' },
                    { key: 'Estimate', type: 'Numeric' },
                    { key: 'Summary', type: 'TextArea' }
                ]
            }
        };
    },
}
</script>
<style>
@import '../node_modules/@syncfusion/ej2-base/styles/material.css';
@import '../node_modules/@syncfusion/ej2-buttons/styles/material.css';
@import '../node_modules/@syncfusion/ej2-layouts/styles/material.css';
@import '../node_modules/@syncfusion/ej2-dropdowns/styles/material.css';
@import '../node_modules/@syncfusion/ej2-inputs/styles/material.css';
@import '../node_modules/@syncfusion/ej2-navigations/styles/material.css';
@import '../node_modules/@syncfusion/ej2-popups/styles/material.css';
@import '../node_modules/@syncfusion/ej2-vue-kanban/styles/material.css';
</style>

```

{% previewsample "page.domainurl/code-snippet/kanban/label-cs1" %}

#### Fields Validation

The dialog fields can be validated while click on the **Save** button. This can be achieved by using **validationRules** property.

#### APP.VUE

```

<template>
  <div id="app">

```



```

    <ejs-kanban id="kanban" keyField="Status" :dataSource="kanbanData"
      :cardSettings="cardSettings" :dialogSettings="dialogSettings">
      <e-columns>
        <e-column headerText="To Do" keyField="Open"></e-column>
        <e-column headerText="In Progress" keyField="InProgress"></e-
column>
        <e-column headerText="Testing" keyField="Testing"></e-column>
        <e-column headerText="Done" keyField="Close"></e-column>
      </e-columns>
    </ejs-kanban>
  </div>
</template>
<script>
import Vue from "vue";
import { KanbanPlugin } from '@syncfusion/ej2-vue-kanban';
import { extend } from '@syncfusion/ej2-base';
import { kanbanData } from './datasource.js';
Vue.use(KanbanPlugin);
export default {
  data: function() {
    return {
      kanbanData: extend([], kanbanData, null, true),
      cardSettings: {
        contentField: "Summary",
        headerField: "Id",
      },
      dialogSettings: {
        fields: [
          { text: 'ID', key: 'Id', type: 'Input' },
          { key: 'Status', type: 'DropDown' },
          { key: 'Estimate', type: 'Numeric', validationRules: { range:
[0, 1000] } },
          { key: 'Summary', type: 'TextArea', validationRules: { required:
true } }
        ]
      }
    };
  },
}
</script>
<style>
@import '../node_modules/@syncfusion/ej2-base/styles/material.css';
@import '../node_modules/@syncfusion/ej2-buttons/styles/material.css';
@import '../node_modules/@syncfusion/ej2-layouts/styles/material.css';
@import '../node_modules/@syncfusion/ej2-dropdowns/styles/material.css';
@import '../node_modules/@syncfusion/ej2-inputs/styles/material.css';
@import '../node_modules/@syncfusion/ej2-navigations/styles/material.css';
@import '../node_modules/@syncfusion/ej2-popups/styles/material.css';
@import '../node_modules/@syncfusion/ej2-vue-kanban/styles/material.css';
</style>

```

{% previewsample "page.domainurl/code-snippet/kanban/fields-validation-cs1" %}

Dialog Template

Using the dialog template, you can render your own dialog by defining the `template` property. Initialize the template as SCRIPT element Id or HTML string which holds the template and map it to the template property.

### APP.VUE

```
<template>
  <div id="app">
    <ejs-kanban id="kanban" keyField="Status" :dataSource="kanbanData"
      :cardSettings="cardSettings" :dialogSettings="dialogSettings">
      <e-columns>
        <e-column headerText="To Do" keyField="Open"></e-column>
        <e-column headerText="In Progress" keyField="InProgress"></e-
column>
        <e-column headerText="Testing" keyField="Testing"></e-column>
        <e-column headerText="Done" keyField="Close"></e-column>
      </e-columns>
    </ejs-kanban>
  </div>
</template>
<script>
import Vue from "vue";
import { KanbanPlugin } from '@syncfusion/ej2-vue-kanban';
import { extend } from '@syncfusion/ej2-base';
import { kanbanData } from './datasource.js';
Vue.use(KanbanPlugin);
var ContentTemplate = Vue.component("dialogTemplate", {
  template: `<div>
<table>
  <tbody>
    <tr>
      <td class="e-label">ID</td>
      <td>
        <div class="e-float-input e-control-wrapper">
          <input id="Id" name="Id" type="text" class="e-field" v-
model='data.Id' disabled />
        </div>
      </td>
    </tr>
    <tr>
      <td class="e-label">Status</td>
      <td>
        <ejs-dropdownlist id="Status" name="Status" class="e-
field" v-model='data.Status' :dataSource="dataSource1"
placeholder="Status"></ejs-dropdownlist>
      </td>
    </tr>
    <tr>
      <td class="e-label">Assignee</td>
      <td>
        <ejs-dropdownlist id="Assignee" name="Assignee" v-
model='data.Assignee' class="e-field" :dataSource="dataSource2"
placeholder="Assignee"></ejs-dropdownlist>
      </td>
    </tr>
    <tr>
      <td class="e-label">Priority</td>

```

```

        <td>
            <ejs-dropdownlist type="text" name="Priority"
id="Priority" v-model='data.Priority' popupHeight="300px" class="e-field"
placeholder="Priority" :dataSource="dataSource3"></ejs-dropdownlist>
        </td>
    </tr>
    <tr>
        <td class="e-label">Summary</td>
        <td>
            <div class="e-float-input e-control-wrapper">
                <textarea type="text" name="Summary" id="Summary"
class="e-field" v-model='data.Summary'></textarea>
            </div>
        </td>
    </tr>
</tbody>
</table>
</div>`,
data() {
    var statusData = [
        { text: "Open" },
        { text: "InProgress" },
        { text: "Testing" },
        { text: "Close" }
    ];
    var priorityData = ["Low", "Normal", "Critical", "Release Breaker",
"High"];
    var assigneeData = [
        "Nancy Davloio",
        "Andrew Fuller",
        "Janet Leverling",
        "Steven walker",
        "Robert King",
        "Margaret hamilt",
        "Michael Suyama"
    ];
    return {
        dataSource1: statusData,
        dataSource2: assigneeData,
        dataSource3: priorityData,
        data: {}
    };
}
});
export default {
    data: function() {
        return {
            kanbanData: extend([], kanbanData, null, true),
            cardSettings: {
                contentField: "Summary",
                headerField: "Id",
            },
            dialogSettings: {
                template: function() {
                    return { template: ContentTemplate };
                }
            }
        }
    }
}

```

```

    };
  },
}
</script>
<style>
@import '../node_modules/@syncfusion/ej2-base/styles/material.css';
@import '../node_modules/@syncfusion/ej2-buttons/styles/material.css';
@import '../node_modules/@syncfusion/ej2-layouts/styles/material.css';
@import '../node_modules/@syncfusion/ej2-dropdowns/styles/material.css';
@import '../node_modules/@syncfusion/ej2-inputs/styles/material.css';
@import '../node_modules/@syncfusion/ej2-navigations/styles/material.css';
@import '../node_modules/@syncfusion/ej2-popups/styles/material.css';
@import '../node_modules/@syncfusion/ej2-vue-kanban/styles/material.css';
</style>

```

{% previewsample "page.domainurl/code-snippet/kanban/dialog-template-cs1" %}

### Prevent Dialog

The Kanban allows to prevent to open a dialog on card double-click by enabling `args.cancel` in `dialogOpen` event.

### APP.VUE

```

<template>
  <div id="app">
    <ejs-kanban id="kanban" keyField="Status" :dataSource="kanbanData"
      :cardSettings="cardSettings" :dialogOpen="dialogOpen">
      <e-columns>
        <e-column headerText="To Do" keyField="Open"></e-column>
        <e-column headerText="In Progress" keyField="InProgress"></e-
column>
        <e-column headerText="Testing" keyField="Testing"></e-column>
        <e-column headerText="Done" keyField="Close"></e-column>
      </e-columns>
    </ejs-kanban>
  </div>
</template>
<script>
import Vue from "vue";
import { KanbanPlugin } from '@syncfusion/ej2-vue-kanban';
import { extend } from '@syncfusion/ej2-base';
import { kanbanData } from './datasource.js';
Vue.use(KanbanPlugin);
export default {
  data: function() {
    return {
      kanbanData: extend([], kanbanData, null, true),
      cardSettings: {
        contentField: "Summary",
        headerField: "Id",
      },
      dialogOpen: function (args) {
        args.cancel = true;
      }
    };
  },
};

```

```

}
</script>
<style>
@import '../node_modules/@syncfusion/ej2-base/styles/material.css';
@import '../node_modules/@syncfusion/ej2-buttons/styles/material.css';
@import '../node_modules/@syncfusion/ej2-layouts/styles/material.css';
@import '../node_modules/@syncfusion/ej2-dropdowns/styles/material.css';
@import '../node_modules/@syncfusion/ej2-inputs/styles/material.css';
@import '../node_modules/@syncfusion/ej2-navigations/styles/material.css';
@import '../node_modules/@syncfusion/ej2-popups/styles/material.css';
@import '../node_modules/@syncfusion/ej2-vue-kanban/styles/material.css';
</style>

```

{% previewsample "page.domainurl/code-snippet/kanban/prevent-dialog-cs1" %}

### Persisting data in server

The modified card data can be persisted in the database using the RESTful web services. All the CRUD operations in the Kanban are done through [DataManager](#). The [DataManager](#) has an option to bind all the CRUD related data in server-side.

For your information, the ODataAdaptor persists data in the server as per OData protocol.

In the below section covers how to get the edited data details on the server-side using the [UrlAdaptor](#).

### URL adaptor

You can use the [UrlAdaptor](#) of [DataManager](#) when binding data source for remote data. In the initial load of Kanban, data are fetched from remote data and bound to the Kanban using [url](#) property of [DataManager](#).

You can map the CRUD operation in Kanban can be mapped to server-side controller actions using the properties [insertUrl](#), [removeUrl](#), [updateUrl](#), and [crudUrl](#).

- [insertUrl](#) – You can perform single insertion operation on server-side.
- [updateUrl](#) – You can update single data on server-side.
- [removeUrl](#) – You can remove single data on server-side.
- [crudUrl](#) – You can perform bulk data operation on server-side.

The following code example describes the above behavior.

```

`ts
<template>
<div id="app">
<ejs-kanban id="kanban" keyField="Status" :dataSource="kanbanData"
:cardSettings="cardSettings">
<e-columns>
<e-column headerText="To Do" keyField="Open"></e-column>
<e-column headerText="In Progress" keyField="InProgress"></e-column>
<e-column headerText="Testing" keyField="Testing"></e-column>

```

```
<e-column headerText="Done" keyField="Close"></e-column>
</e-columns>
</ejs-kanban>
</div>
</template>
<script>
import Vue from "vue";
import { KanbanPlugin } from '@syncfusion/ej2-vue-kanban';
import { DataManager, UrlAdaptor } from "@syncfusion/ej2-data";
Vue.use(KanbanPlugin);
export default {
  data: function() {
    return {
      kanbanData: new DataManager({
        url: 'Home/DataSource',
        updateUrl: 'Home/Update',
        insertUrl: 'Home/Insert',
        removeUrl: 'Home/Delete',
        adaptor: new UrlAdaptor(),
        crossDomain: true
      });
      cardSettings: {
        contentField: "Summary",
        headerField: "Id",
      }
    };
  }
}
</script>
<style>
@import '../node_modules/@syncfusion/ej2-base/styles/material.css';
@import '../node_modules/@syncfusion/ej2-buttons/styles/material.css';
@import '../node_modules/@syncfusion/ej2-layouts/styles/material.css';
```

```

@import '../node_modules/@syncfusion/ej2-dropdowns/styles/material.css';
@import '../node_modules/@syncfusion/ej2-inputs/styles/material.css';
@import '../node_modules/@syncfusion/ej2-navigations/styles/material.css';
@import '../node_modules/@syncfusion/ej2-popups/styles/material.css';
@import '../node_modules/@syncfusion/ej2-vue-kanban/styles/material.css';
</style>
`

```

The server-side controller code to handle the CRUD operations are as follows.

```

`ts
private NORTHWNDEntities db = new NORTHWNDEntities();
public ActionResult DataSource() {
    var DataSource = db.Tasks.ToList();
    return Json(DataSource, JsonRequestBehavior.AllowGet);
}
public ActionResult Insert(Params value) {
    //Insert card data into the database
    return Json(value, JsonRequestBehavior.AllowGet);
}
public ActionResult Update(Params value) {
    //Update card data into the database
    return Json(value, JsonRequestBehavior.AllowGet);
}
public void Delete(Params value) {
    //Delete card data from the database
}
public class Params {
    public int Id { get; set; }
    public string Status { get; set; }
    public string Summary { get; set; }
    public string Assignee { get; set; }
}
`

```

### *Insert card*

Using the `insertUrl` property, you can specify the controller action mapping URL to perform insert operation on the server-side.

The following code example describes the above behavior.

```
`ts
public ActionResult Insert(Params value)
{
    //Insert card in the database
}
`
```

The newly added record details are bound to the `value` parameter.

### *Update card*

Using the `updateUrl` property, the controller action mapping URL can be specified to perform save/update operation on the server-side.

The following code example describes the above behavior.

```
`ts
public ActionResult Update(Params value)
{
    //Update card data in the database
}
`
```

The updated record details are bound to the `value` parameter.

### *Delete card*

Using the `removeUrl` property, the controller action mapping URL can be specified to perform delete operation on the server-side.

The following code example describes the above behavior.

```
`ts
public void Delete(int key)
{
    //Delete card in the database
}
`
```

The deleted card primary key value is bound to the `key` parameter.



### CRUD URL

Using the `crudUrl` property, the controller action mapping URL can be specified to perform all the CRUD operations at the server-side using a single method instead of specifying a separate controller action method for CRUD (insert, update and delete) operations.

The action parameter of `crudUrl` is used to get the corresponding CRUD action.

The following code example describes the above behavior.

```
`ts
<template>
  <div id="app">
    <ejs-kanban id="kanban" keyField="Status" :dataSource="kanbanData"
      :cardSettings="cardSettings">
      <e-columns>
        <e-column headerText="To Do" keyField="Open"></e-column>
        <e-column headerText="In Progress" keyField="InProgress"></e-column>
        <e-column headerText="Testing" keyField="Testing"></e-column>
        <e-column headerText="Done" keyField="Close"></e-column>
      </e-columns>
    </ejs-kanban>
  </div>
</template>
<script>
import Vue from "vue";
import { KanbanPlugin } from '@syncfusion/ej2-vue-kanban';
import { DataManager, UrlAdaptor } from "@syncfusion/ej2-data";
Vue.use(KanbanPlugin);
export default {
  data: function() {
    return {
      kanbanData: new DataManager({
        url: 'Home/DataSource',
        updateUrl: 'Home/UpdateData',
        insertUrl: 'Home/UpdateData',
        removeUrl: 'Home/UpdateData',
        crudUrl: 'Home/UpdateData',
      })
    }
  }
}
```

```

adaptor: new UrlAdaptor(),
crossDomain: true
});
cardSettings: {
contentField: "Summary",
headerField: "Id",
}
};
}
}
</script>
<style>
@import '../node_modules/@syncfusion/ej2-base/styles/material.css';
@import '../node_modules/@syncfusion/ej2-buttons/styles/material.css';
@import '../node_modules/@syncfusion/ej2-layouts/styles/material.css';
@import '../node_modules/@syncfusion/ej2-dropdowns/styles/material.css';
@import '../node_modules/@syncfusion/ej2-inputs/styles/material.css';
@import '../node_modules/@syncfusion/ej2-navigations/styles/material.css';
@import '../node_modules/@syncfusion/ej2-popups/styles/material.css';
@import '../node_modules/@syncfusion/ej2-vue-kanban/styles/material.css';
</style>
`
`ts
private NORTHWNDEntities db = new NORTHWNDEntities();
public ActionResult DataSource() {
var DataSource = db.Tasks.ToList();
return Json(DataSource, JsonRequestBehavior.AllowGet);
}
public ActionResult UpdateData(EditParams param) {
if (param.action == "insert" || (param.action == "batch" && param.added != null)) {
if (param.action == "insert") {
db.Tasks.Add(param.value);
} else {

```

```
foreach (var temp in param.added) {
    db.Tasks.Add(temp);
}
}
}
if (param.action == "update" || (param.action == "batch" && param.changed != null)) {
    if (param.action == "update") {
        Task old = db.Tasks.Where(o => o.Id == param.value.Id).SingleOrDefault();
        if (old != null) {
            db.Entry(old).CurrentValues.SetValues(param.value);
        }
    } else {
        foreach (var temp in param.changed) {
            Task old = db.Tasks.Where(o => o.Id == temp.Id).SingleOrDefault();
            if (old != null) {
                db.Entry(old).CurrentValues.SetValues(temp);
            }
        }
    }
}
if (param.action == "remove" || (param.action == "batch" && param.deleted != null)) {
    if (param.action == "remove") {
        int key = Convert.ToInt32(param.key);
        db.Tasks.Remove(db.Tasks.Where(o => o.Id == key).SingleOrDefault());
    } else {
        foreach (var temp in param.deleted) {
            db.Tasks.Remove(db.Tasks.Where(o => o.Id == temp.Id).SingleOrDefault());
        }
    }
}
db.SaveChanges();
return Json(param, JsonRequestBehavior.AllowGet);
}
```

```

public class EditParams {
public string key { get; set; }
public string action { get; set; }
public List<Tasks> added { get; set; }
public List<Tasks> changed { get; set; }
public List<Tasks> deleted { get; set; }
public Tasks value { get; set; }
}

```

The `crudUrl` is used to update the bulk data sent to the server-side. Multiple selections and `sortBy` as `Index` properties are used for `crudUrl` properties to update the modified bulk data to the server-side.

### Tooltip in Vue Kanban component

The tooltip is used to show the card information when the cursor hover over the card elements using the `enableTooltip` property. Tooltip content is dynamically set based on hovering over the card elements.

If you wish to show tooltip on Kanban board custom elements, you need to add `e-tooltip-text` class name of a particular element.

### Tooltip template

You can customize the tooltip content with any HTML or CSS element and styling using the `tooltipTemplate` property. In the following demo, the tooltip is customized with HTML elements.

### APP.VUE

```

<template>
  <div id="app">
    <ejs-kanban id="kanban" keyField="Status" :dataSource="kanbanData"
      :cardSettings="cardSettings" :enableTooltip='true'
      :tooltipTemplate="tooltipTemplate">
      <e-columns>
        <e-column headerText="To Do" keyField="Open"></e-column>
        <e-column headerText="In Progress" keyField="InProgress"></e-
column>
        <e-column headerText="Testing" keyField="Testing"></e-column>
        <e-column headerText="Done" keyField="Close"></e-column>
      </e-columns>
    </ejs-kanban>
  </div>
</template>
<script>
import Vue from "vue";
import { KanbanPlugin } from '@syncfusion/ej2-vue-kanban';
import { extend } from '@syncfusion/ej2-base';
import { kanbanData } from './datasource.js';
Vue.use(KanbanPlugin);
export default {
  data: function() {

```

```

return {
  kanbanData: extend([], kanbanData, null, true),
  cardSettings: {
    contentField: "Summary",
    headerField: "Id",
  },
  tooltipTemplate: function () {
    return {
      template: Vue.component('tooltipTemplate', {
        template: `<div class='e-kanbanTooltipTemp'>
          <table>
            <tr>
              <td class="details">
                <table>
                  <colgroup>
                    <col style="width:30%">
                    <col style="width:70%">
                  </colgroup>
                  <tbody>
                    <tr>
                      <td
class="CardHeader">Assignee:</td>
                      <td>{{data.Assignee}}</td>
                    </tr>
                    <tr>
                      <td
class="CardHeader">Type:</td>
                      <td>{{data.Type}}</td>
                    </tr>
                    <tr>
                      <td
class="CardHeader">Estimate:</td>
                      <td>{{data.Estimate}}</td>
                    </tr>
                    <tr>
                      <td
class="CardHeader">Summary:</td>
                      <td>{{data.Summary}}</td>
                    </tr>
                  </tbody>
                </table>
              </td>
            </tr>
          </table>
        </div>`,
        data() { }
      })
    }
  },
};

```

```

@import '../node_modules/@syncfusion/ej2-dropdowns/styles/material.css';
@import '../node_modules/@syncfusion/ej2-inputs/styles/material.css';
@import '../node_modules/@syncfusion/ej2-navigations/styles/material.css';
@import '../node_modules/@syncfusion/ej2-popups/styles/material.css';
@import '../node_modules/@syncfusion/ej2-vue-kanban/styles/material.css';
.e-kanbanTooltipTemp {
    width: 250px;
    padding: 3px;
}
.e-kanbanTooltipTemp>table {
    width: 100%;
}
.e-kanbanTooltipTemp td {
    vertical-align: top;
}
</style>

```

{% previewsample "page.domainurl/code-snippet/kanban/tooltip-template-cs1" %}

### Validation in Vue Kanban component

Validate particular column using the `minCount` or `maxCount` properties. The corresponding columns gets different appearance when validation fails. In default layout, `constraintType` property accept only `Column` type. In swimlane layout, accept both `Column` and `Swimlane` constraint type.

There are two types of constraints:

1. Column
2. Swimlane

By default, the column count validation is performed based on Kanban **columns**.

#### Minimum card limit

The `minCount` property is used to specify the minimum cards hold on particular column or swimlane cell. If the column or swimlane total card count falls short of the minimum count value, it shows the column or cell background colour with validation fails.

#### Maximum card limit

The `maxCount` property is used to specify the maximum cards hold on particular column or swimlane cell. If the column or swimlane cell total card count exceeds the maximum count value, it shows the column or cell background colour with validation fails.

#### APP.VUE

```

<template>
  <div id="app">
    <ejs-kanban id="kanban" keyField="Status" :dataSource="kanbanData"
      :cardSettings="cardSettings" :enableTooltip=true>
      <e-columns>
        <e-column headerText="To Do" keyField="Open" minCount="6"></e-
column>
        <e-column headerText="In Progress" keyField="InProgress"
maxCount="5"></e-column>
        <e-column headerText="Testing" keyField="Testing" maxCount="4"
minCount="3"></e-column>
      </e-columns>
    </ejs-kanban>
  </div>
</template>

```

```

        <e-column headerText="Done" keyField="Close"></e-column>
      </e-columns>
    </ejs-kanban>
  </div>
</template>
<script>
import Vue from "vue";
import { KanbanPlugin } from '@syncfusion/ej2-vue-kanban';
import { extend } from '@syncfusion/ej2-base';
import { kanbanData } from './datasource.js';
Vue.use(KanbanPlugin);
export default {
  data: function() {
    return {
      kanbanData: extend([], kanbanData, null, true),
      cardSettings: {
        contentField: "Summary",
        headerField: "Id",
      }
    };
  },
};
</script>
<style>
@import '../node_modules/@syncfusion/ej2-base/styles/material.css';
@import '../node_modules/@syncfusion/ej2-buttons/styles/material.css';
@import '../node_modules/@syncfusion/ej2-layouts/styles/material.css';
@import '../node_modules/@syncfusion/ej2-dropdowns/styles/material.css';
@import '../node_modules/@syncfusion/ej2-inputs/styles/material.css';
@import '../node_modules/@syncfusion/ej2-navigations/styles/material.css';
@import '../node_modules/@syncfusion/ej2-popups/styles/material.css';
@import '../node_modules/@syncfusion/ej2-vue-kanban/styles/material.css';
</style>

```

{% previewsample "page.domainurl/code-snippet/kanban/column-validation-cs1" %}

## Virtualization in Vue Kanban component

Kanban allows you to load a large amount of data without any performance degradation. This feature can be enabled by setting the [enableVirtualization](#) property in the Kanban to **true**.

### Virtual scrolling

Virtual scrolling optimizes data rendering within each column when using large datasets. Only a subset of cards that are visible and about to be loaded on the screen are rendered. The number of records displayed in the Kanban is determined implicitly by the height of the Kanban area and the card height. The [cardHeight](#) property of Kanban can be used to set the cards' height in pixel value. By default, the card height will be **auto**.

When the Kanban column is scrolled, the virtual scrolling feature dynamically loads additional data on demand into view and unloads the data that is no longer visible.

### APP.VUE

```

<template>
  <div id="app">

```

```

    <ejs-kanban id="KanbanVirtualScrolling" :enableVirtualization="true"
    keyField="Status"
      :dataSource="kanbanData" :enableTooltip="enableTooltip"
    :cardSettings="cardSettings" :dialogSettings="dialogSettings">
      <e-columns>
        <e-column headerText="To Do" keyField="Open" ></e-column>
        <e-column headerText="In Progress" keyField="InProgress" ></e-
column>
        <e-column headerText="In Review" keyField="Review" ></e-column>
        <e-column headerText="Done" keyField="Close" ></e-column>
      </e-columns>
    </ejs-kanban>
  </div>
</template>
<script>
import Vue from "vue";
import { KanbanPlugin } from '@syncfusion/ej2-vue-kanban';
import { extend } from '@syncfusion/ej2-base';
Vue.use(KanbanPlugin);
let BUG_TASKS = [
  'UI component not displaying images in IE browser',
  'Button not responding on hover action',
  'Text overlapping in mobile view',
  'Dropdown menu not functioning properly',
  'Form validation error',
  'Alignment issue in tables',
  'Column not loading completely',
  'Broken UI Designs',
  'Font size inconsistency',
  'UI element misaligned on scroll'
];
let FEATURE_TASKS = [
  'Implement new user registration flow',
  'Add pagination to search results',
  'Improve accessibility for visually impaired users',
  'Create custom dashboard for users',
  'Develop user profile editing functionality',
  'Integrate with third-party API for weather data',
  'Implement social media sharing for articles',
  'Add support for multiple languages',
  'Create onboarding tutorial for new users',
  'Implement push notifications for mobile app'
];
let EPIC_TASKS = [
  'Revamp UI design for entire application',
  'Develop mobile application for iOS and Android',
  'Create API for integration with external systems',
  'Implement machine learning algorithms for personalized
recommendations',
  'Upgrade database infrastructure for scalability',
  'Integrate with payment gateway for subscription model',
  'Develop chatbot for customer support',
  'Implement real-time collaboration features for team projects',
  'Create analytics dashboard for administrators',
  'Introduce gamification elements to increase user engagement',
];

```



```

let assignee = ['Andrew Fuller', 'Janet Leverling', 'Steven walker', 'Robert King', 'Margaret hamilt', 'Nancy Davloio', 'Margaret Buchanan', 'Laura Bergs', 'Anton Fleet', 'Jack Kathryn', 'Martin Davolio', 'Fleet Jack'];
let status = ['Open', 'InProgress', 'Review', 'Testing', 'Close'];
let priority = ['Ultra-Critical', 'Critical', 'High', 'Normal', 'Low'];
let types = ['Epic', 'Bug', 'Story'];
let tagsField = ['Feature', 'Bug', 'Enhancement', 'Documentation', 'Automation', 'Mobile', 'Web', 'iOS', 'Safari', 'Chrome', 'Firefox', 'Manual Testing'];
let storyPoints = ['1', '2', '3', '3.5', '4', '4.5', '5', '6', '7.5', '8'];
let count = 60000;
let generateKanbanDataVirtualScrollData = () => {
  let kanbanVirtualData = [];
  for (let a = 50000, id = 50000; a < count; a++) {
    let typeValue = types[Math.floor(Math.random() * types.length)];
    let summary = typeValue === 'Bug' ?
BUG_TASKS[Math.floor(Math.random() * BUG_TASKS.length)] :
    typeValue === 'Story' ? FEATURE_TASKS[Math.floor(Math.random() *
FEATURE_TASKS.length)] :
    EPIC_TASKS[Math.floor(Math.random() * EPIC_TASKS.length)];
    kanbanVirtualData.push({
      Id: id,
      Type: typeValue,
      Priority: priority[Math.floor(Math.random() * priority.length)],
      Status: status[Math.floor(Math.random() * status.length)],
      Assignee: assignee[Math.floor(Math.random() * assignee.length)],
      StoryPoints: storyPoints[Math.floor(Math.random() *
storyPoints.length)],
      Tags: [tagsField[Math.floor(Math.random() * tagsField.length)],
tagsField[Math.floor(Math.random() * tagsField.length)]],
      Title: 'Task ' + id,
      Summary: summary,
    });
    id++;
  }
  return kanbanVirtualData;
}
export default {
  data: function() {
    return {
      kanbanData: extend([], generateKanbanDataVirtualScrollData(), null,
true),
      enableTooltip: true,
      cardSettings: {
        headerField: "Id",
        contentField: "Summary",
        selectionType: "Multiple"
      },
      dialogSettings: {
        fields: [
          {key: 'Id', text: 'ID', type: 'TextBox'},
          {key: 'Status', text: 'Status', type: 'DropDown'},
          {key: 'StoryPoints', text: 'Story Points', type: 'Numeric' },
          {key: 'Summary', text: 'Summary', type: 'TextArea'}
        ]
      }
    }
  };
};

```

```

    },
  }
</script>
<style>
  @import '../node_modules/@syncfusion/ej2-base/styles/material.css';
  @import '../node_modules/@syncfusion/ej2-buttons/styles/material.css';
  @import '../node_modules/@syncfusion/ej2-layouts/styles/material.css';
  @import '../node_modules/@syncfusion/ej2-dropdowns/styles/material.css';
  @import '../node_modules/@syncfusion/ej2-inputs/styles/material.css';
  @import '../node_modules/@syncfusion/ej2-navigations/styles/material.css';
  @import '../node_modules/@syncfusion/ej2-popups/styles/material.css';
  @import '../node_modules/@syncfusion/ej2-vue-kanban/styles/material.css';
</style>

```

{% previewsample "page.domainurl/code-snippet/kanban/virtual-scrolling-cs1" %}

#### *Configure the remote data service*

When the remote data is configured for the [dataSource](#), the service method will receive an additional **KanbanVirtualization** parameter to handle the initial data load for Kanban Virtualization.

To handle Kanban virtual scrolling, the server-side code needs to handle the **Where** and **Take** queries differently using the **KanbanVirtualization** parameter. The following is the example code for handling Kanban virtualization's initial data load using the **KanbanVirtualization** parameter.

```

`ts
public IActionResult LoadCard([FromBody] ExtendedDataManagerRequest dm)
{
    kanbanData = _context.KanbanCards.ToList();
    IEnumerable<KanbanCard> DataSource = kanbanData.AsEnumerable();
    DataOperations operation = new DataOperations();
    // For normal kanban data load Where query handling.
    if (dm.Where != null && dm.Where.Count > 0 && dm.KanbanVirtualization != "KanbanVirtualization")
    {
        dm.Where[0].value = dm.Where[0].value.ToString();
        DataSource = operation.PerformFiltering(DataSource, dm.Where, dm.Where[0].Operator);
    }
    if (dm.Skip != 0)
    {
        DataSource = operation.PerformSkip(DataSource, dm.Skip);
    }
    // For normal Kanban data load Take query handling.
    if (dm.Take != 0 && dm.KanbanVirtualization != "KanbanVirtualization")

```

```
{
DataSource = operation.PerformTake(DataSource, dm.Take);
}

// For Kanban virtual scrolling data load Where and Take query handling.
var columnCount = new List<KeyValuePair<string, int>>();
if (dm.KanbanVirtualization == "KanbanVirtualization" && dm.Where != null && dm.Where.Count > 0
&& dm.Take != 0)
{
IEnumerable<KanbanCard> currentData = new List<KanbanCard>();
List<WhereFilter> currentFilter = new List<WhereFilter>();
for (int i = 0; i < dm.Where.Count; i++)
{
dm.Where[i].value = dm.Where[i].value.ToString();
currentFilter.Add(dm.Where[i]);
var filterData = operation.PerformFiltering(DataSource, currentFilter, dm.Where[i].Operator);
columnCount.Add(new KeyValuePair<string, int>(dm.Where[i].value.ToString(), filterData.Count()));
filterData = operation.PerformTake(filterData, dm.Take);
currentData = currentData.Concat(filterData);
currentFilter.Clear();
}
DataSource = currentData;
}

// To return the data for Kanban virtual scrolling.
if (dm.KanbanVirtualization == "KanbanVirtualization") {
return Json(new { result = DataSource, count = columnCount });
}

// To return the data for Kanban virtual scrolling.
else
{
return Json(DataSource);
}
}
```

### Limitations for virtual scrolling

- When virtualization is enabled in a Kanban board and the card height is not explicitly set, it will not default to `auto` height. Instead, a fixed height of `100px` will be applied to the cards. It's important to note that the card height should be specified in pixel values, as percentage values are not accepted.
- When a card is dragged and dropped, the index position of the card will not be preserved when scrolling through the column.
- Virtualization is not supported for swimlanes in the Kanban board.

### Localization in Vue Kanban component

The localization library allows you to localize the default text content of the Kanban to different cultures using the `locale` property.

In Kanban, total count and min or max count text alone will be localized based on culture.

Locale key	en-US (default)
-----	-----
items	items
min	Min
max	Max
cardsSelected	Cards Selected
addTitle	Add New Card
editTitle	Edit Card Details
deleteTitle	Delete Card
deleteContent	Are you sure you want to delete this card?
save	Save
delete	Delete
cancel	Cancel
yes	Yes
no	No
close	Close
noCard	No cards to display
unassigned	Unassigned

### Loading translations

To load translation object in an application, use `load` function of `L10n` class.

The following example demonstrates the Kanban in `Deutsch` culture.

#### **APP.VUE**

```
<template>
```

```

<div id="app">
  <ejs-kanban id="kanban" keyField="Status" :dataSource="kanbanData"
    :cardSettings="cardSettings" :swimlaneSettings="swimlaneSettings"
    locale='de'>
    <e-columns>
      <e-column headerText="To Do" keyField="Open" minCount= "6"></e-
column>
      <e-column headerText="In Progress" keyField="InProgress"
maxCount= "3"></e-column>
      <e-column headerText="Done" keyField="Close"></e-column>
    </e-columns>
  </ejs-kanban>
</div>
</template>
<script>
import Vue from "vue";
import { KanbanPlugin } from '@syncfusion/ej2-vue-kanban';
import { extend, L10n } from '@syncfusion/ej2-base';
import { kanbanData } from './datasource.js';
Vue.use(KanbanPlugin);
L10n.load({
  'de': {
    'kanban': {
      'items': 'Artikel',
      'min': 'Min',
      'max': 'Max',
      'cardsSelected': 'Karten ausgewählt',
      'addTitle': 'Neue Karte hinzufügen',
      'editTitle': 'Kartendetails bearbeiten',
      'deleteTitle': 'Karte löschen',
      'deleteContent': 'Möchten Sie diese Karte wirklich löschen?',
      'save': 'speichern',
      'delete': 'Löschen',
      'cancel': 'Stornieren',
      'yes': 'Ja',
      'no': 'Nein',
      'close': 'Schließen',
      'noCard': 'Keine Karten zum Anzeigen',
      'unassigned': 'nicht zugewiesen'
    }
  }
});
export default {
  data: function() {
    return {
      kanbanData: extend([], kanbanData, null, true),
      cardSettings: {
        contentField: "Summary",
        headerField: "Id",
      },
      swimlaneSettings: {
        keyField: "Assignee"
      }
    };
  },
}
</script>

```

```

<style>
@import '../node_modules/@syncfusion/ej2-base/styles/material.css';
@import '../node_modules/@syncfusion/ej2-buttons/styles/material.css';
@import '../node_modules/@syncfusion/ej2-layouts/styles/material.css';
@import '../node_modules/@syncfusion/ej2-dropdowns/styles/material.css';
@import '../node_modules/@syncfusion/ej2-inputs/styles/material.css';
@import '../node_modules/@syncfusion/ej2-navigations/styles/material.css';
@import '../node_modules/@syncfusion/ej2-popups/styles/material.css';
@import '../node_modules/@syncfusion/ej2-vue-kanban/styles/material.css';
</style>

```

```
{% previewsample "page.domainurl/code-snippet/kanban/locale-cs1" %}
```

### Right to left (RTL)

The Kanban provides an option to switch its text direction and layout from right to left. It improves the user experiences and accessibility for users who use right-to-left languages (Arabic, Farsi, Urdu, etc.). To enable right-to-left mode in Kanban, set the `enableRtl` to true.

### APP.VUE

```

<template>
  <div id="app">
    <ejs-kanban id="kanban" keyField="Status" :dataSource="kanbanData"
      :cardSettings="cardSettings" :swimlaneSettings="swimlaneSettings"
      locale='ar' :enableRtl='true'>
      <e-columns>
        <e-column headerText="To Do" keyField="Open" minCount= "2"></e-
column>
        <e-column headerText="In Progress" keyField="InProgress"
maxCount= "3"></e-column>
        <e-column headerText="Done" keyField="Close"></e-column>
      </e-columns>
    </ejs-kanban>
  </div>
</template>
<script>
import Vue from "vue";
import { KanbanPlugin } from '@syncfusion/ej2-vue-kanban';
import { extend, L10n } from '@syncfusion/ej2-base';
import { kanbanData } from './datasource.js';
Vue.use(KanbanPlugin);
L10n.load({
  'ar': {
    'kanban': {
      'items': 'العناصر',
      'min': 'أنا',
      'max': 'ماكس',
      'cardsSelected': 'تم تحديد البطاقات',
      'addTitle': 'إضافة بطاقة جديدة',
      'editTitle': 'تحرير تفاصيل البطاقة',
      'deleteTitle': 'حذف البطاقة',
      'deleteContent': 'هل أنت متأكد أنك تريد حذف هذه البطاقة؟',
      'save': 'حفظ',
      'delete': 'حذف',
      'cancel': 'إلغاء',
      'yes': 'نعم',

```

```

        'no': 'لا',
        'close': 'قريب',
        'noCard': 'لا توجد بطاقات لعرضها',
        'unassigned': 'غير معين'
    }
}
});
export default {
    data: function() {
        return {
            kanbanData: extend([], kanbanData, null, true),
            cardSettings: {
                contentField: "Summary",
                headerField: "Id",
            },
            swimlaneSettings: {
                keyField: "Assignee"
            }
        };
    },
}
</script>
<style>
@import '../node_modules/@syncfusion/ej2-base/styles/material.css';
@import '../node_modules/@syncfusion/ej2-buttons/styles/material.css';
@import '../node_modules/@syncfusion/ej2-layouts/styles/material.css';
@import '../node_modules/@syncfusion/ej2-dropdowns/styles/material.css';
@import '../node_modules/@syncfusion/ej2-inputs/styles/material.css';
@import '../node_modules/@syncfusion/ej2-navigations/styles/material.css';
@import '../node_modules/@syncfusion/ej2-popups/styles/material.css';
@import '../node_modules/@syncfusion/ej2-vue-kanban/styles/material.css';
</style>

```

{% previewsample "page.domainurl/code-snippet/kanban/rtl-cs1" %}

## Dimensions in Vue Kanban component

The Kanban dimensions refers to both height and width of the entire layout and it accepts three types of values.

- Auto
- Pixel
- Percentage

### Auto height and width

When height and width of the Kanban are set to `auto`, it will try as hard as possible to keep an element the same width as its parent container. In other words, the parent container that holds Kanban, its width or height will be the sum of its children. By default, Kanban is assigned with `auto` values for both the height and width properties.

### APP.VUE

```

<template>
  <div id="app">
    <ejs-kanban id="kanban" keyField="Status" :dataSource="kanbanData"

```

```

        :cardSettings="cardSettings" width="auto" height="auto">
        <e-columns>
            <e-column headerText="To Do" keyField="Open"></e-column>
            <e-column headerText="In Progress" keyField="InProgress"></e-
column>
            <e-column headerText="Testing" keyField="Testing"></e-column>
            <e-column headerText="Done" keyField="Close"></e-column>
        </e-columns>
    </ejs-kanban>
</div>
</template>
<script>
import Vue from "vue";
import { KanbanPlugin } from '@syncfusion/ej2-vue-kanban';
import { extend } from '@syncfusion/ej2-base';
import { kanbanData } from './datasource.js';
Vue.use(KanbanPlugin);
export default {
    data: function() {
        return {
            kanbanData: extend([], kanbanData, null, true),
            cardSettings: {
                contentField: "Summary",
                headerField: "Id"
            }
        };
    },
}
</script>
<style>
@import '../node_modules/@syncfusion/ej2-base/styles/material.css';
@import '../node_modules/@syncfusion/ej2-buttons/styles/material.css';
@import '../node_modules/@syncfusion/ej2-layouts/styles/material.css';
@import '../node_modules/@syncfusion/ej2-dropdowns/styles/material.css';
@import '../node_modules/@syncfusion/ej2-inputs/styles/material.css';
@import '../node_modules/@syncfusion/ej2-navigations/styles/material.css';
@import '../node_modules/@syncfusion/ej2-popups/styles/material.css';
@import '../node_modules/@syncfusion/ej2-vue-kanban/styles/material.css';
</style>

```

{% previewsample "page.domainurl/code-snippet/kanban/auto-cs1" %}

### Height and width in pixel

The Kanban height and width will be rendered exactly as per the given pixel values. It accepts both string and number values.

### APP.VUE

```

<template>
    <div id="app">
        <ejs-kanban id="kanban" keyField="Status" :dataSource="kanbanData"
        :cardSettings="cardSettings" width=650 height="550px">
            <e-columns>
                <e-column headerText="To Do" keyField="Open"></e-column>
                <e-column headerText="In Progress" keyField="InProgress"></e-
column>
            </e-columns>
        </ejs-kanban>
    </div>
</template>

```



```

        <e-column headerText="Testing" keyField="Testing"></e-column>
        <e-column headerText="Done" keyField="Close"></e-column>
    </e-columns>
</ejs-kanban>
</div>
</template>
<script>
import Vue from "vue";
import { KanbanPlugin } from '@syncfusion/ej2-vue-kanban';
import { extend } from '@syncfusion/ej2-base';
import { kanbanData } from './datasource.js';
Vue.use(KanbanPlugin);
export default {
    data: function() {
        return {
            kanbanData: extend([], kanbanData, null, true),
            cardSettings: {
                contentField: "Summary",
                headerField: "Id"
            }
        };
    },
}
</script>
<style>
@import '../node_modules/@syncfusion/ej2-base/styles/material.css';
@import '../node_modules/@syncfusion/ej2-buttons/styles/material.css';
@import '../node_modules/@syncfusion/ej2-layouts/styles/material.css';
@import '../node_modules/@syncfusion/ej2-dropdowns/styles/material.css';
@import '../node_modules/@syncfusion/ej2-inputs/styles/material.css';
@import '../node_modules/@syncfusion/ej2-navigations/styles/material.css';
@import '../node_modules/@syncfusion/ej2-popups/styles/material.css';
@import '../node_modules/@syncfusion/ej2-vue-kanban/styles/material.css';
</style>

```

{% previewsample "page.domainurl/code-snippet/kanban/pixel-cs1" %}

### Height and width in percentage

When height and width of the Kanban are given in percentage, it will make the Kanban as wide as the parent container.

### APP.VUE

```

<template>
    <div id="app">
        <ejs-kanban id="kanban" keyField="Status" :dataSource="kanbanData"
            :cardSettings="cardSettings" width="100%" height="100%">
            <e-columns>
                <e-column headerText="To Do" keyField="Open"></e-column>
                <e-column headerText="In Progress" keyField="InProgress"></e-
column>
                <e-column headerText="Testing" keyField="Testing"></e-column>
                <e-column headerText="Done" keyField="Close"></e-column>
            </e-columns>
        </ejs-kanban>
    </div>

```

```

</template>
<script>
import Vue from "vue";
import { KanbanPlugin } from '@syncfusion/ej2-vue-kanban';
import { extend } from '@syncfusion/ej2-base';
import { kanbanData } from './datasource.js';
Vue.use(KanbanPlugin);
export default {
  data: function() {
    return {
      kanbanData: extend([], kanbanData, null, true),
      cardSettings: {
        contentField: "Summary",
        headerField: "Id"
      }
    };
  },
};
</script>
<style>
@import '../node_modules/@syncfusion/ej2-base/styles/material.css';
@import '../node_modules/@syncfusion/ej2-buttons/styles/material.css';
@import '../node_modules/@syncfusion/ej2-layouts/styles/material.css';
@import '../node_modules/@syncfusion/ej2-dropdowns/styles/material.css';
@import '../node_modules/@syncfusion/ej2-inputs/styles/material.css';
@import '../node_modules/@syncfusion/ej2-navigations/styles/material.css';
@import '../node_modules/@syncfusion/ej2-popups/styles/material.css';
@import '../node_modules/@syncfusion/ej2-vue-kanban/styles/material.css';
</style>

```

{% previewsample "page.domainurl/code-snippet/kanban/percentage-cs1" %}

### Persistence in Vue Kanban component

State persistence refers to the Kanban state maintained in the browser's [localStorage](#) even if the browser is refreshed or if you move to the next page within the browser.

State persistence stores Kanban datasource, column and swimlane expand/collapse state in the local storage when the [enablePersistence](#) is defined as true.

### APP.VUE

```

<template>
  <div id="app">
    <ejs-kanban id="kanban" keyField="Status" :dataSource="kanbanData"
      :cardSettings="cardSettings" enablePersistence="true"
      :swimlaneSettings="swimlaneSettings">
      <e-columns>
        <e-column headerText="To Do" keyField="Open"
          allowToggle="true"></e-column>
        <e-column headerText="In Progress" keyField="InProgress"
          allowToggle="true"></e-column>
        <e-column headerText="Testing" keyField="Testing"
          allowToggle="true"></e-column>
        <e-column headerText="Done" keyField="Close"
          allowToggle="true"></e-column>
      </e-columns>
    </div>
  </template>

```

```

        </ejs-kanban>
    </div>
</template>
<script>
import Vue from "vue";
import { KanbanPlugin } from '@syncfusion/ej2-vue-kanban';
import { extend } from '@syncfusion/ej2-base';
import { kanbanData } from './datasource.js';
Vue.use(KanbanPlugin);
export default {
    data: function() {
        return {
            kanbanData: extend([], kanbanData, null, true),
            cardSettings: {
                contentField: "Summary",
                headerField: "Id",
            },
            swimlaneSettings: {
                keyField: "Assignee"
            }
        };
    },
}
</script>
<style>
@import '../node_modules/@syncfusion/ej2-base/styles/material.css';
@import '../node_modules/@syncfusion/ej2-buttons/styles/material.css';
@import '../node_modules/@syncfusion/ej2-layouts/styles/material.css';
@import '../node_modules/@syncfusion/ej2-dropdowns/styles/material.css';
@import '../node_modules/@syncfusion/ej2-inputs/styles/material.css';
@import '../node_modules/@syncfusion/ej2-navigations/styles/material.css';
@import '../node_modules/@syncfusion/ej2-popups/styles/material.css';
@import '../node_modules/@syncfusion/ej2-vue-kanban/styles/material.css';
</style>

```

{% previewsample "page.domainurl/code-snippet/kanban/persistence-cs1" %}

### Responsive mode in Vue Kanban component

The Kanban component has support for responsive behavior based on the client browser's width and height.

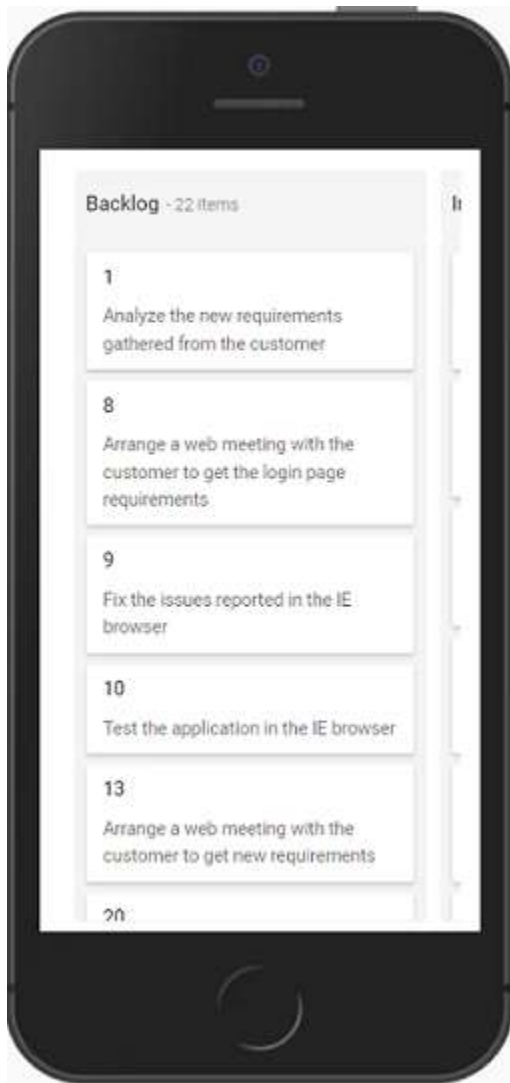
#### Layouts

Possible layouts are:

- Default Layout
- Swimlane Layout

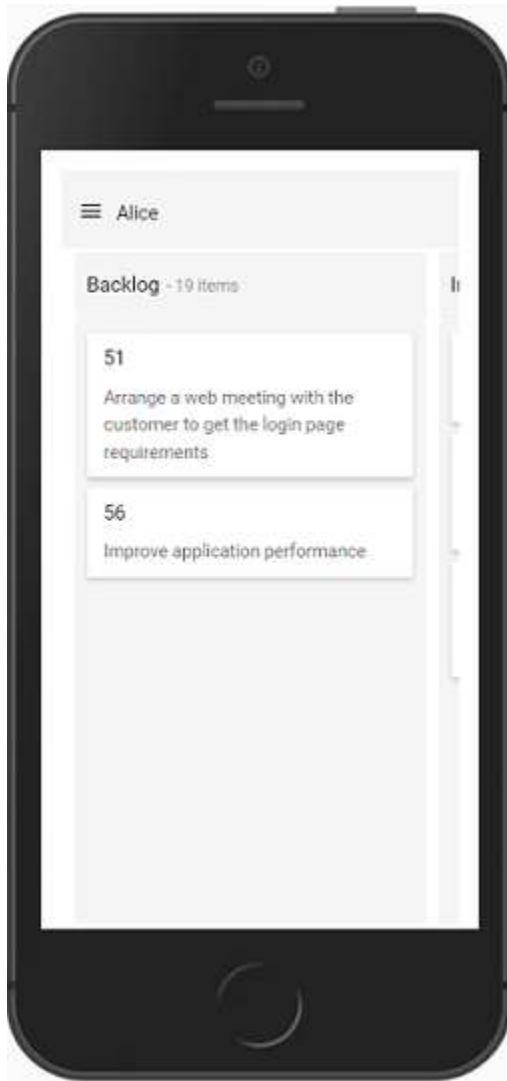
#### Default Layout

Kanban user interface is customized and redesigned for the best view on small screens. In responsive mode, the first column occupies 80% and the second column occupies 20% of the screen layout. Tap and hold the Kanban card to drag and drop it. Swipe left or right to view the columns.



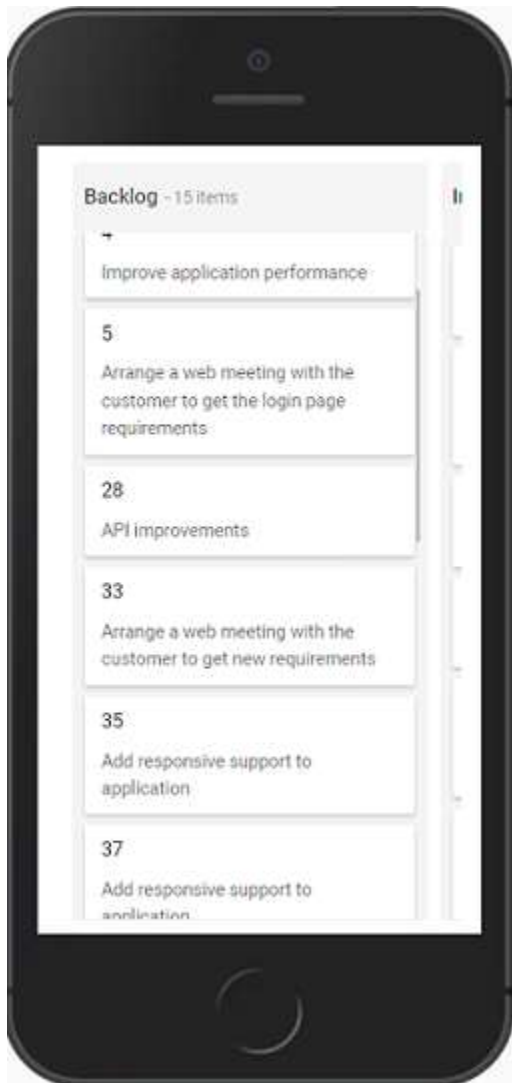
### *Swimlane Layout*

Kanban swimlane header is rendered with menu icon on top of the kanban board. It will show all the available swimlane groups of the header text with a popup when clicking the menu icon. Swimlane selected grouped header text resultant data is rendered on the Kanban board. By default, the first swimlane grouped header text is selected and the resultant data is shown on the Kanban board. The Kanban board data will be changed when changing the swimlane group header text.



### Scrolling

Column scrolling will be shown when exceeding the screen size in the columns.

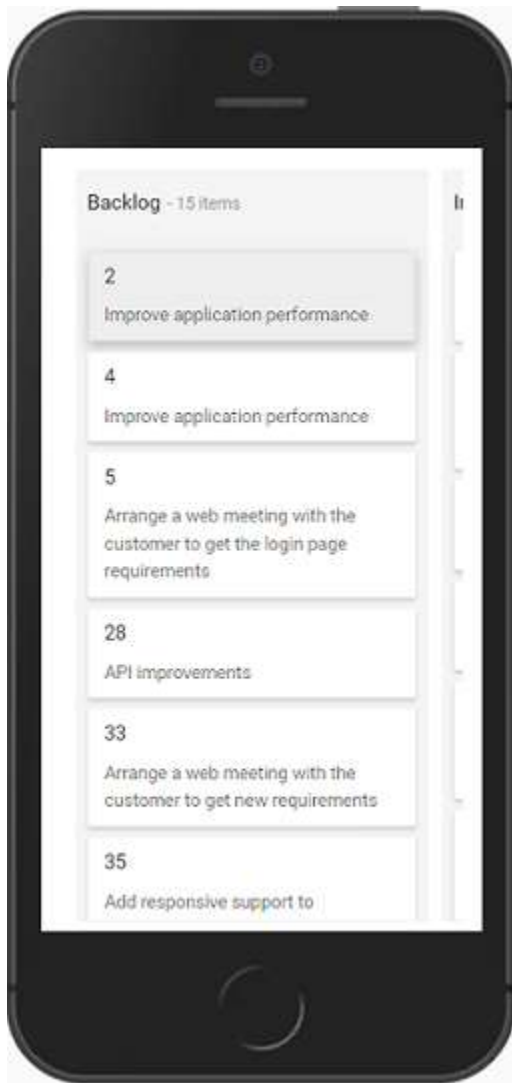


### Selection

Select particular cards in the Kanban board by tapping the card.

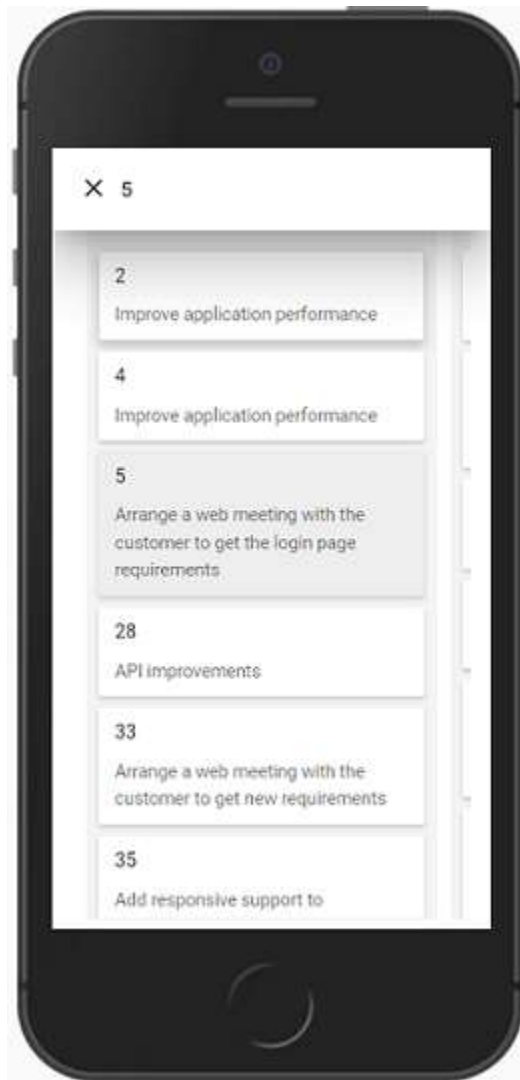
#### Single Selection

Single card will be selected when you tap the card once and selection will be removed when you select another card.

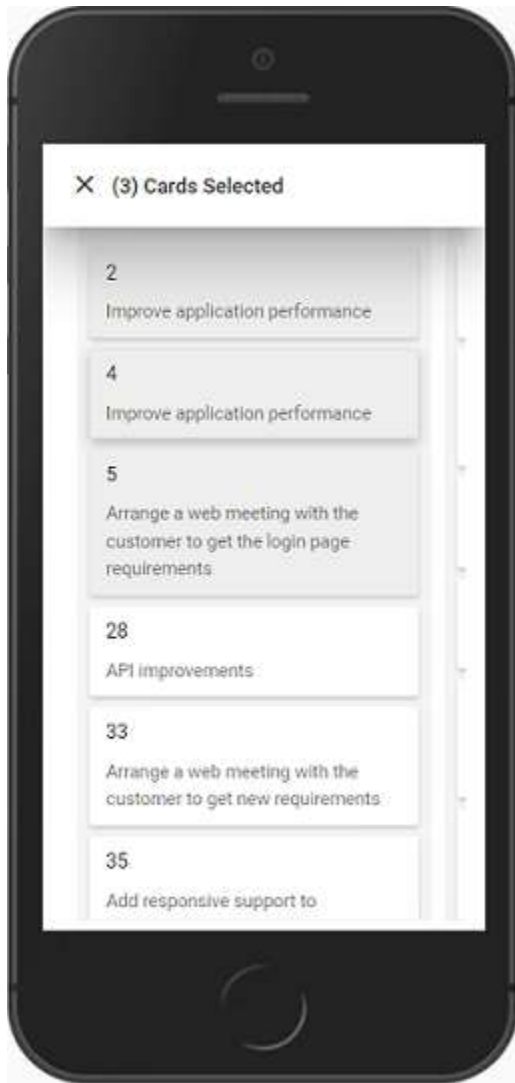


#### *Multiple Selection*

Enable [selectionType](#) as **Multiple** to select multiple cards. It will open the popup on the screen top. Selected card header text will be shown when selecting single card with a tap and hold action. If single card is selected, only tap action is required to select multiple cards. Multiple Selected card count will be shown on the popup when selecting multiple cards.







### Style in Vue Kanban component

To modify the Kanban appearance, you need to override the default CSS of Kanban. Also, you have an option to create your own custom theme using our [Theme Studio](#). Please find the list of CSS classes in Kanban.

Css class	Purpose
<code>.e-kanban</code>	Customize the kanban.
<code>.e-kanban .e-kanban-table</code>	Header cells of kanban.
<code>.e-kanban .e-kanban-header .e-header-cells</code>	Header title of kanban.
<code>.e-kanban .e-kanban-header .e-header-cells .e-header-wrap .e-header-title</code>	Header cells minimum color of kanban.
<code>.e-kanban .e-kanban-header .e-header-cells.e-min-color</code>	Header cells maximum color of kanban.
<code>.e-kanban .e-kanban-header .e-header-cells.e-max-color</code>	Header cells of collapsed column minimum color in column constraint type of kanban.
<code>.e-kanban .e-kanban-header .e-header-cells.e-collapsed.e-min-color</code>	

| .e-kanban .e-kanban-header .e-header-cells.e-collapsed.e-max-color | Header cells of collapsed column maximum color in column constraint type of kanban. |

| .e-kanban .e-kanban-header .e-header-cells .e-header-text | Header text of Kanban. |

| .e-kanban .e-kanban-header .e-header-cells .e-item-count | Header cells Item count of Kanban. |

| .e-kanban .e-kanban-header .e-header-cells .e-limits | Header cells limits in column constraint type of kanban. |

| .e-kanban .e-kanban-header .e-header-cells .e-limits .e-min-count | Header cells minimum count of kanban. |

| .e-kanban .e-kanban-header .e-header-cells .e-limits .e-max-count | Header cells maximum count of kanban. |

| .e-kanban .e-kanban-content | Customize kanban Content. |

| .e-kanban .e-kanban-content .e-content-row .e-content-cells .e-limits | Content cells limits in swimlane constraint type of kanban. |

| .e-kanban .e-kanban-content .e-content-row .e-content-cells .e-limits .e-min-count | Content cells minimum count of kanban. |

| .e-kanban .e-kanban-content .e-content-row .e-content-cells .e-limits .e-max-count | Content cells maximum count of kanban. |

| .e-kanban .e-kanban-content .e-content-row .e-content-cells.e-min-color | Content cells minimum color of kanban. |

| .e-kanban .e-kanban-content .e-content-row .e-content-cells.e-max-color | Content cells maximum color of kanban. |

| .e-kanban .e-kanban-content .e-content-row .e-content-cells.e-collapsed .e-collapse-header-text | Content cells of collapsed header text. |

| .e-kanban .e-kanban-content .e-content-row .e-content-cells.e-collapsed .e-collapse-header-text .e-item-count | Content cells of collapsed header text Item count. |

| .e-kanban .e-kanban-content .e-content-row .e-content-cells .e-show-add-button | Add button in content cells of kanban. |

| .e-kanban .e-kanban-content .e-content-row .e-content-cells .e-show-add-button .e-show-add-icon | Customize content cells add icon of kanban. |

| .e-kanban .e-kanban-content .e-content-row .e-content-cells .e-card-wrapper .e-empty-card | Empty content cells of kanban. |

| .e-kanban .e-kanban-content .e-content-row .e-content-cells .e-card-wrapper .e-card | Customize cards in kanban. |

| .e-kanban .e-kanban-content .e-content-row .e-content-cells .e-card-wrapper .e-card .e-card-header .e-card-header-title | Cards header title of kanban. |

| .e-kanban .e-kanban-content .e-content-row .e-content-cells .e-card-wrapper .e-card .e-card-footer | Cards footer of kanban. |

| .e-kanban .e-kanban-content .e-content-row .e-content-cells .e-card-wrapper .e-card .e-card-content | Cards content of kanban. |

| .e-kanban .e-kanban-content .e-content-row .e-content-cells .e-card-wrapper .e-card.e-card-color |  
Cards color of kanban. |

| .e-kanban .e-kanban-content .e-content-row .e-content-cells .e-card-wrapper .e-card .e-card-tags |  
Customize Card tags of kanban. |

| .e-kanban .e-kanban-content .e-content-row .e-content-cells .e-card-wrapper .e-card .e-card-tag |  
Card tag of kanban. |

| .e-kanban .e-kanban-content .e-content-row.e-swimlane-row .e-content-cells .e-swimlane-header .e-swimlane-row-expand | Content cells of swimlane row expand of kanban. |

| .e-kanban .e-kanban-content .e-content-row.e-swimlane-row .e-content-cells .e-swimlane-header .e-swimlane-row-collapse | Content cells of swimlane row collapse of kanban. |

| .e-kanban .e-kanban-content .e-content-row.e-swimlane-row .e-content-cells .e-swimlane-header .e-swimlane-text | Content cells of swimlane header text of kanban. |

| .e-kanban .e-kanban-content .e-content-row.e-swimlane-row .e-content-cells .e-swimlane-header .e-item-count | Content cells of swimlane items count of kanban. |

| .e-kanban .e-kanban-content .e-content-row:not(.e-swimlane-row) .e-content-cells | swimlane content cells of kanban. |

| .e-kanban .e-kanban-content .e-content-row:not(.e-swimlane-row) .e-content-cells.e-dropping |  
Customize swimlane content cells card dropping of kanban. |

| .e-kanban .e-kanban-content .e-content-row:not(.e-swimlane-row) .e-content-cells .e-card-wrapper |  
Swimlane content cells of card wrapper. |

| .e-kanban .e-kanban-content .e-content-row:not(.e-swimlane-row) .e-content-cells.e-min-color |  
Swimlane content cells of minimum color of kanban. |

| .e-kanban .e-kanban-content .e-content-row:not(.e-swimlane-row) .e-content-cells.e-max-color |  
Swimlane content cells of maximum color of kanban. |Customize the kanban CSS theme. Please find the list of CSS classes in Kanban. |

| .e-kanban .e-kanban-table .e-header-cells | Header cells of kanban. |

| .e-kanban .e-kanban-table .e-header-cells .e-header-text | Header text of Kanban. |

| .e-kanban .e-kanban-table .e-header-cells .e-item-count | Header cells Item count of Kanban. |

| .e-kanban .e-kanban-table .e-header-cells .e-column-expand | Header cells of toggle icon in column expand. |

| .e-kanban .e-kanban-table .e-header-cells .e-column-collapse | Header cells of toggle icon in column collapse. |

| .e-kanban .e-kanban-table.e-content-table .e-content-row:not(.e-swimlane-row) .e-content-cells |  
swimlane content cells of kanban. |

| .e-kanban .e-kanban-table.e-content-table .e-content-row.e-swimlane-row .e-swimlane-text | Content cells of swimlane header text of kanban. |

| .e-kanban .e-kanban-table.e-content-table .e-content-row.e-swimlane-row .e-item-count | Content cells of swimlane items count of kanban. |

| .e-kanban .e-kanban-table.e-content-table .e-content-row .e-show-add-button .e-show-add-icon | Add icon in content cells of kanban. |

.e-kanban .e-kanban-table.e-content-table .e-card.e-selection	Selected card of kanban.
.e-kanban .e-kanban-table.e-content-table .e-card .e-card-header	Cards header in kanban.
.e-kanban .e-kanban-table.e-content-table .e-card .e-card-content	Cards content in kanban.
.e-kanban .e-kanban-table.e-content-table .e-card .e-card-tag.e-card-label	Cards label in kanban.

#### To set fixed position to the Kanban header

The Fixed header in Kanban control can be customized in following ways,

By setting a fixed height to the Kanban content,

```
`css
.e-kanban .e-kanban-content {
height: 500px;
}
`
```

By customizing the CSS for the Kanban header.

```
`css
.e-kanban-header {
position: -webkit-sticky;
position: sticky;
z-index: 100;
top: 0;
}
`
```

Note: It will not affect the Kanban content's height.

#### Accessibility in Vue Kanban component

The Kanban component has been designed, keeping in mind the WAI-ARIA specifications, and applies the WAI-ARIA roles, states, and properties. This component is characterized by complete ARIA accessibility support that makes it easy for people who use assistive technologies (AT) or those who completely rely on keyboard navigation.

The accessibility compliance for the Kanban component is outlined below.

Accessibility Criteria	Compatibility
[WCAG 2.2 Support](#)	![Yes](https://cdn.syncfusion.com/content/images/landing-page/yes.png)
[Section 508 Support](#)	![Yes](https://cdn.syncfusion.com/content/images/landing-page/yes.png)
[Screen Reader Support](#)	![Yes](https://cdn.syncfusion.com/content/images/landing-page/yes.png)
[Right-To-Left Support](#)	![Yes](https://cdn.syncfusion.com/content/images/landing-page/yes.png)

| [Color Contrast](#) | ![Intermediate](https://cdn.syncfusion.com/content/images/landing-page/intermediate.png) |

| [Mobile Device Support](#) | ![Yes](https://cdn.syncfusion.com/content/images/landing-page/yes.png) |

| [Keyboard Navigation Support](#) | ![Yes](https://cdn.syncfusion.com/content/images/landing-page/yes.png) |

| [Accessibility Checker Validation](#) |

![Intermediate](https://cdn.syncfusion.com/content/images/landing-page/intermediate.png) |

| [Axe-core Accessibility Validation](#) | ![Yes](https://cdn.syncfusion.com/content/images/landing-page/yes.png) |

```
<style>
```

```
.post .post-content img {
```

```
display: inline-block;
```

```
margin: 0.5em 0;
```

```
}
```

```
</style>
```

![Yes](https://cdn.syncfusion.com/content/images/landing-page/yes.png) - All features of the component meet the requirement.

![Intermediate](https://cdn.syncfusion.com/content/images/landing-page/intermediate.png) - Some features of the component do not meet the requirement.

![No](https://cdn.syncfusion.com/content/images/landing-page/no.png) - The component does not meet the requirement.

### WAI-ARIA attributes

The Kanban component followed the [WAI-ARIA](#) patterns to meet the accessibility. The following ARIA attributes are used in the Kanban component:

| Attributes | Purpose |

| --- | --- |

| `aria-label` | It helps to provides information about elements in a kanban component for assistive technology. |

| `aria-expanded` | Attributes indicate the state of a collapsible element. |

| `aria-selected` | This attribute is assigned to the Kanban component for the selection of elements, and its default value is `false`. The value changes to true when the user selects a Kanban card. |

| `aria-grabbed` | Indicates whether the attribute is set to true. It has been selected for dragging. If this attribute is set to false, the element can be grabbed for a drag-and-drop operation but will not be currently grabbed. |

| `aria-describedby` | This attribute contains the ID of the Kanban header column to indicate that the attribute establishes an association between the Kanban header column and the Kanban column body. |

| **aria-roledescription** | This attribute is assigned to the Kanban component and is used to provide alternative descriptions for card elements. |

#### Keyboard interaction

The Kanban component followed the [keyboard interaction](#) guideline, making it easy for people who use assistive technologies (AT) and those who completely rely on keyboard navigation. The following keyboard shortcuts are supported by the Kanban component.

| **Press** | **To do this** |

| --- | --- |

| **Home** | To select the first card in the kanban |

| **End** | To select the last card in the kanban |

| **Arrow Up** | Select the card through the up arrow |

| **Arrow Down** | Select the card through the down arrow |

| **Arrow Right** | Move the column selection to the right |

| **Arrow Left** | Move the column selection to the left |

| **Ctrl + Enter** | Used to select the multi cards |

| **Ctrl + Space** | Used to select the multi cards |

| **Shift + Arrow Up** | Used to select the multiple cards towards up |

| **Shift + Arrow Down** | Used to select the multiple cards towards down |

| **Shift + Tab** | Reverse order of the tab action |

| **Enter** | Open the selected cards |

| **Tab** | To navigate the Kanban column |

| **Delete** | To delete the selected cards |

| **ESC** | Escape from the modified details |

| **Space** | Used to open the card edit dialog based on the column selection |

#### Ensuring accessibility

The Kanban component's accessibility levels are ensured through an [accessibility-checker](#) and [axe-core](#) software tools during automated testing.

The accessibility compliance of the Kanban component is shown in the following sample. Open the [sample](#) in a new window to evaluate the accessibility of the Kanban component with accessibility tools.

{% previewsample "<https://ej2.syncfusion.com/accessibility/kanban.html>" %}

See also

- [Accessibility in Syncfusion Vue components](#)

## How To

### Header double click in Vue Kanban component

You can bind the header double click event by using the [dataBound](#) event at the initial rendering. You can get the column header text when you double click on the headers.

#### APP.VUE

```
<template>
  <div id="app">
    <ejs-kanban id="kanban" keyField="Status" :dataSource="kanbanData"
      :cardSettings="cardSettings" :dataBound="OnDataBound">
      <e-columns>
        <e-column headerText="To Do" keyField="Open"></e-column>
        <e-column headerText="In Progress" keyField="InProgress"></e-
column>
        <e-column headerText="Testing" keyField="Testing"></e-column>
        <e-column headerText="Done" keyField="Close"></e-column>
      </e-columns>
    </ejs-kanban>
  </div>
</template>
<script>
import Vue from "vue";
import { KanbanPlugin } from '@syncfusion/ej2-vue-kanban';
import { extend } from '@syncfusion/ej2-base';
import { DialogUtility } from '@syncfusion/ej2-popups';
import { kanbanData } from './datasource.js';
Vue.use(KanbanPlugin);
Vue.use(DialogUtility);
export default {
  data: function() {
    return {
      kanbanData: extend([], kanbanData, null, true),
      cardSettings: {
        contentField: "Summary",
        headerField: "Id"
      }
    };
  },
  methods: {
    OnDataBound: function() {
      var headerEle: HTMLElement = document.querySelector('.e-header-
row');
      headerEle.addEventListener("dblclick", function (e: Event) {
        var target = closest(e.target, '.e-header-cells');
        DialogUtility.alert({
          title: 'Header',
          content: "Double clicked on " + target.querySelector('.e-
header-text').innerText + " header",
          showCloseIcon: true,
          closeOnEscape: true,
          animationSettings: { effect: 'Zoom' }
        });
      });
    },
  },
}
```

```

}
</script>
<style>
@import '../node_modules/@syncfusion/ej2-base/styles/material.css';
@import '../node_modules/@syncfusion/ej2-buttons/styles/material.css';
@import '../node_modules/@syncfusion/ej2-layouts/styles/material.css';
@import '../node_modules/@syncfusion/ej2-dropdowns/styles/material.css';
@import '../node_modules/@syncfusion/ej2-inputs/styles/material.css';
@import '../node_modules/@syncfusion/ej2-navigations/styles/material.css';
@import '../node_modules/@syncfusion/ej2-popups/styles/material.css';
@import '../node_modules/@syncfusion/ej2-vue-kanban/styles/material.css';
</style>

```

{% previewsample "page.domainurl/code-snippet/kanban/card-header-cs4" %}

### Dynamically change columns in Vue Kanban component

You can dynamically change the Kanban columns by using the [columns](#) property.

In the below sample, you can dynamically change the [allowToggle](#) property at the particular column when you click on the button. You can also change the initially created columns to the new Kanban columns by using the [columns](#) property when you click on the button.

#### APP.VUE

```

<template>
  <div id="app">
    <ejs-button
      id="particular_column"
      class="e-btn"
      v-on:click.native="particularColumnClick"
    >Enable Allow Toggle</ejs-button>
    <ejs-button id="column" class="e-btn" v-on:click.native="columnClick"
    >Change Columns</ejs-button>
    <ejs-kanban
      ref="KanbanObj"
      id="kanban"
      keyField="Status"
      :dataSource="kanbanData"
      :cardSettings="cardSettings"
    >
      <e-columns>
        <e-column headerText="To Do" keyField="Open"></e-column>
        <e-column headerText="In Progress" keyField="InProgress"></e-column>
        <e-column headerText="Testing" keyField="Testing"></e-column>
        <e-column headerText="Done" keyField="Close"></e-column>
      </e-columns>
    </ejs-kanban>
  </div>
</template>
<script>
import Vue from "vue";
import { KanbanPlugin } from "@syncfusion/ej2-vue-kanban";
import { ButtonPlugin } from "@syncfusion/ej2-vue-buttons";
import { extend } from "@syncfusion/ej2-base";

```



```

import { kanbanData } from "../datasource.js";
Vue.use(KanbanPlugin);
Vue.use(ButtonPlugin);
export default {
  data: function () {
    return {
      kanbanData: extend([], kanbanData, null, true),
      cardSettings: {
        contentField: "Summary",
        headerField: "Id",
      },
    };
  },
  mounted: function () {
    this.kanbanObj = this.$refs.KanbanObj.ej2Instances;
  },
  methods: {
    particularColumnClick: function () {
      this.kanbanObj.columns[1].allowToggle = true;
    },
    columnClick: function () {
      this.kanbanObj.columns = [
        { headerText: "To Do", keyField: "Open" },
        { headerText: "Done", keyField: "Close" },
      ];
    },
  },
};
</script>
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-buttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-layouts/styles/material.css";
@import "../node_modules/@syncfusion/ej2-dropdowns/styles/material.css";
@import "../node_modules/@syncfusion/ej2-inputs/styles/material.css";
@import "../node_modules/@syncfusion/ej2-navigations/styles/material.css";
@import "../node_modules/@syncfusion/ej2-popups/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-kanban/styles/material.css";
</style>

```

{% previewsample "page.domainurl/code-snippet/kanban/card-header-cs2" %}

### Filter cards in Vue Kanban component

You can filter the collection of cards from the dataSource and display it on the Kanban board by using the [query](#) property.

In the below sample, you can filter the cards based on the 'where' query and display the filtered data to the Kanban board.

#### APP.VUE

```

<template>
  <div id="app">
    <ejs-dropdownlist
      ref="PriorityDrop"
      id="priority_filter"
    >

```

```

        :dataSource="priorityData"
        :change="change"
        value="None"
        placeholder="Select a priority"
    ></ejs-dropdownlist>
    <ejs-kanban
        ref="KanbanObj"
        id="kanban"
        keyField="Status"
        :dataSource="kanbanData"
        :cardSettings="cardSettings"
    >
        <e-columns>
            <e-column headerText="To Do" keyField="Open"></e-column>
            <e-column headerText="In Progress" keyField="InProgress"></e-column>
            <e-column headerText="Testing" keyField="Testing"></e-column>
            <e-column headerText="Done" keyField="Close"></e-column>
        </e-columns>
    </ejs-kanban>
</div>
</template>
<script>
import Vue from "vue";
import { KanbanPlugin } from "@syncfusion/ej2-vue-kanban";
import { DropDownListPlugin } from "@syncfusion/ej2-vue-dropdowns";
import { extend } from "@syncfusion/ej2-base";
import { Query } from "@syncfusion/ej2-data";
import { kanbanData } from "../datasource.js";
Vue.use(KanbanPlugin);
Vue.use(DropDownListPlugin);
export default {
    data: function () {
        return {
            kanbanData: extend([], kanbanData, null, true),
            cardSettings: {
                contentField: "Summary",
                headerField: "Id",
            },
            priorityData: ["None", "High", "Normal", "Low"],
        };
    },
    mounted: function () {
        this.kanbanObj = this.$refs.KanbanObj.ej2Instances;
        this.priorityObj = this.$refs.PriorityDrop.ej2Instances;
    },
    methods: {
        change: function (args) {
            let filterQuery = new Query();
            if (args.value !== "None") {
                filterQuery = new Query().where("Priority", "equal", args.value);
            }
            this.kanbanObj.query = filterQuery;
        },
    },
};
</script>
<style>

```

```
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-buttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-layouts/styles/material.css";
@import "../node_modules/@syncfusion/ej2-dropdowns/styles/material.css";
@import "../node_modules/@syncfusion/ej2-inputs/styles/material.css";
@import "../node_modules/@syncfusion/ej2-navigations/styles/material.css";
@import "../node_modules/@syncfusion/ej2-popups/styles/material.css";
@import "../node_modules/@syncfusion/ej2-vue-kanban/styles/material.css";
</style>
```

```
{% previewsample "page.domainurl/code-snippet/kanban/card-header-cs3" %}
```

### Search cards in Vue Kanban component

You can search the cards in Kanban by using the `query` property.

In the following sample, the searching operation starts as soon as you start typing characters in the external text box. It will search the cards based on the `Id` and `Summary` using the `search` query with `contains` operator.

### APP.VUE

```
<template>
  <div id="app">
    <table>
      <tbody>
        <td style="width: 200px">
          <ejs-textbox
            ref="SearchText"
            id="search_text"
            placeholder="Enter search text"
            showClearButton="true"
          ></ejs-textbox>
        </td>
        <td>
          <ejs-button
            id="reset_filter"
            class="e-btn"
            v-on:click.native="resetClick"
          >Reset</ejs-button>
        </td>
      </tbody>
    </table>
    <ejs-kanban
      id="kanban"
      ref="kanbanObj"
      keyField="Status"
      :dataSource="kanbanData"
      :cardSettings="cardSettings"
    >
      <e-columns>
        <e-column headerText="To Do" keyField="Open"></e-column>
        <e-column headerText="In Progress" keyField="InProgress"></e-column>
        <e-column headerText="Testing" keyField="Testing"></e-column>
        <e-column headerText="Done" keyField="Close"></e-column>
      </e-columns>
```

```

    </ejs-kanban>
  </div>
</template>
<script>
import Vue from "vue";
import { KanbanPlugin } from "@syncfusion/ej2-vue-kanban";
import { ButtonPlugin } from "@syncfusion/ej2-vue-buttons";
import { TextBoxPlugin } from "@syncfusion/ej2-vue-inputs";
import { extend } from "@syncfusion/ej2-base";
import { Query } from "@syncfusion/ej2-data";
import { kanbanData } from "../datasource.js";
Vue.use(KanbanPlugin);
Vue.use(TextBoxPlugin);
Vue.use(ButtonPlugin);
export default {
  data: function () {
    return {
      kanbanData: extend([], kanbanData, null, true),
      cardSettings: {
        contentField: "Summary",
        headerField: "Id",
      },
      priorityData: ["None", "High", "Normal", "Low"],
    };
  },
  mounted: function () {
    this.kanbanObj = this.$refs.kanbanObj.ej2Instances;
    this.textObj = this.$refs.SearchText.ej2Instances;
    document.getElementById("search_text").addEventListener("keyup", (e) =>
    {
      let searchValue = e.target.value;
      let searchQuery = new Query();
      if (searchValue !== "") {
        searchQuery = new Query().search(
          searchValue,
          ["Id", "Summary"],
          "contains",
          true
        );
      }
      this.kanbanObj.query = searchQuery;
    });
  },
  methods: {
    resetClick: function () {
      this.textObj.value = "";
      this.reset();
    },
  },
};
</script>
<style>
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-buttons/styles/material.css";
@import "../node_modules/@syncfusion/ej2-layouts/styles/material.css";
@import "../node_modules/@syncfusion/ej2-dropdowns/styles/material.css";
@import "../node_modules/@syncfusion/ej2-inputs/styles/material.css";

```

```
@import "../node_modules/@syncfusion/ej2-navigations/styles/material.css";  
@import "../node_modules/@syncfusion/ej2-popups/styles/material.css";  
@import "../node_modules/@syncfusion/ej2-vue-kanban/styles/material.css";  
</style>
```

{% previewsample "page.domainurl/code-snippet/kanban/card-header-cs5" %}